Wintersemester 2009/2010

Projekt: Multimodale Instruktion

Projektbericht

Anreicherung von CRIL-Netzen mit Hilfe von Beschreibungslogiken

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich WSV

Daniel von Poschinger-Camphausen
4poschin@informatik.uni-hamburg.de
Thomas Feldhaus
4feldhau@informatik.uni-hamburg.de
Julian Burkhart
4burkhar@informatik.uni-hamburg.de

Inhaltsverzeichnis

1 Einleitung

1.1 Der Geometrische Agent

Im Rahmen eines Projektes am Department Informatik der Universität Hamburg wurde ein Modell eines instruierten Agenten entwickelt, der geometrische Agent[1]. Ein instruierter Agent ist eine Variation des so genannten intelligenten Agenten, also eines autonomen, seine Umwelt beeinflussenden und beobachtenden Agenten, der sein Handeln auf das Erreichen von Zielen ausrichtet. Einem instruierten Agenten werden a priori Informationen über die zu erreichenden Ziele und die Umwelt, in der das Ziel erreicht werden soll, gegeben. Der Agent nutzt dann diese Instruktionen, um seine Ziele autonom zu erreichen.

Das Wissen des Agenten um die Umwelt und um seine Instruktionen ist in so genannten CRIL-Graphen kodiert (CRIL = Conceptual Route Instruction Language). Über eine vordefinierte Konzepthierarchie (Taxonomie) werden Vergleiche über die Ähnlichkeit von Objekten (die als CRIL-Teilgraphen repäsentiert werden) der Instruktion und der Perzeption ermöglicht. CRIL Graphen und ihr theoretischer Unterbau werden in Abschnitt 2 eingeführt und näher beschrieben.

Die Domäne des GA ist die Navigation bzw. die Verfolgung von Wegbeschreibungen in einer perzeptierten Umwelt. Dabei wird die wahrgenommene Umwelt vom GA in einen internen Modell repräsentiert. Über den Abgleich der intern repräsentierten Umwelt mit den gegebenen Instruktionen navigiert der GA in seiner Umwelt. Dabei werden Instruktionen in Form von natürlichsprachlichen oder graphischen Wegbeschreibungen gegeben. Hierfür enthält die Wissensbasis des GA linguistisches Wissen, um die natürlichsprachlichen Instruktionen zu analysieren wie auch räumliche und zeitliche Konzepte.

Der Ablauf der Simulation des GA ist in zwei Phasen eingeteilt: die Instruktions- und die Aktionsphase. Während der Instruktionsphase wird dem Agenten eine Wegbeschreibung gegeben mit deren Hilfe navigiert werden soll. Die Beschreibung wird interpretiert und in eine Repräsentation umgewandelt, die der Wissensbasis des GA hinzugefügt wird. In der Aktionsphase navigiert der Agent, indem er die einzelnen Teilstücke der Wegbeschreibung in seiner Wissensbasis mit seiner perzeptierten Umgebung abgleicht. Die Vergleiche erfolgen über ein Ähnlichkeitsmaß, welches die Entitäten der Wissensbasis miteinander vergleicht. Bei hinreichender Ähnlichkeit zwischen der Instruktion und der Perzeption wird der GA versuchen die vermeintlich

[1] des weiteren GA genannt

zu seiner Instruktion gehörende Landmarke anzusteuern. Der Prozess der Navigation des GA durch seine wahrgenommene Umwelt kann nun in drei unterschiedlichen Zuständen enden:

1. der GA hat erfolgreich alle Wegpunkte seiner Instruktion abgearbeitet und ist tatsächlich an seinem Ziel angekommen.

2. der GA hat aus seiner Sicht erfolgreich alle Wegpunkte abgearbeitet, ist aber nicht am gewünschten Ort angelangt.

3. der GA hat einige oder keinen Wegpunkt abgearbeitet und befindet sich in einen Zustand, in dem er keinen zu seiner Instruktion passende Landmarke findet.

In den Punkten 2) und 3) zeigen sich die Schwächen des GA bei den oben beschriebenen Vergleichen zwischen den Repräsentationen der Perzeption und der Instruktion. Auf diesen Aspekt wird in den folgenden Abschnitten noch näher eingegangen.

1.2 Aufgabenstellung

Die von unserer Projektgruppe erarbeitete Aufgabenstellung sieht vor, bestehende CRIL-Netze regelbasiert anzureichern, wobei die Regeln mit Hilfe einer Beschreibungslogik formuliert werden. Dies ermöglicht es, Eigenschaften (Konzepte) aus der Struktur des Netzes abzuleiten und diese Eigenschaften einzelnen Knoten des CRIL-Netzes als Attribute hinzuzufügen. Dabei stehen atomare Konzepte für Eigenschaften (bzw. Attribute) von CRIL-Knoten und beschreibungslogische Rollen entsprechen den Relationen zwischen den Knoten eines CRIL-Netzes.

Im Kontext des GA soll unser Ansatz als ein optionales Modul fungieren, welches in allen Verarbeitungsschritten, in denen CRIL-Netze verarbeitet werden, die Möglichkeit eröffnet, diese regelhaft mit neuen Informationen anzureichern.

Als Grundlage für die verwendete Beschreibungslogik soll auf einen, Tableau-Beweiser für Beschreibungslogiken zurück gegriffen werden, der am Fachbereich in einen anderen Projekt entwickelt wurde (siehe Abschnitt 4.1 Eingebundene Projekte). Wobei sich die Autoren auf die Nutzung des Aufbaus der internen Repräsentation, sprich auf den Parser des Tableau-Beweisers beschränken wollen.

Des Weiteren sollte es möglich sein, Regeln über eine Benutzerschnittstelle zu formulieren, zu validieren und diese in Regelmengen zusammenzufassen. Damit im Kontext des GA die Anreicherung automatisch geschieht,

sollte es Anwendungsfälle (folgend Usecase genannt) geben, denen einzelne Regelmengen zugeordnet werden können.

1.2.1 Motivation

Die beim Ablauf der Simulation des GA beobachtete Schwäche beim Abgleich der Wegpunkte der Instruktion mit seiner perzeptierten Umwelt führte die Autoren zu der Idee, CRIL-Netze regelhaft anzureichern. Eine Anreicherung ist als verfügbar machen von neuen Informationen und Zusammenhängen zu verstehen. Die Autoren erhoffen sich dadurch eine Verbesserung der Abgleichung und somit auch eine generelle Verbesserung des Navigationsverhalten des GAs.

Diese zusätzlichen Informationen sind in den Regeln enthalten und werden durch die Anreicherung das Netzes expliziert. Da, wie oben angedeutet, CRIL-Netze universell im GA eingesetzt werden (zur Repräsentation der Perzeption als auch der Instruktion), ist zu erwarten, dass viele Teilaspekte des GA von einer Anreicherung und der daraus resultierenden Verbesserung des Matchens profitieren. Ursprünglich wurde eine Anreicherung mithilfe von speziellen CRIL-Netzen erwogen, wegen folgender Gründe wurden aber Beschreibungslogiken favorisiert:

- Es konnte eine strukturelle Verbindung bzw. Ähnlichkeit ausgenutzt werden, um beschreibungslogische Konzepte auf Attribute von CRIL-Knoten und beschreibungslogische Rollen auf Relationen in CRIL-Netzen abzubilden. Diese Abbildung ist nicht vollständig, da in der Taxonomie des GA auch dreistellige Prädikate existieren, wovon bisher im GA aber nur ein Typ, `between`, interpretiert wird. Generell sollte es möglich sein auch dreistellige Prädikate durch Komposition zweistelliger Prädikate zu emulieren. Dieses wurde wegen eben genannter Gründe von den Autoren nicht weiter verfolgt.

- Die Autoren konnten auf die Implementation eines Tableaubeweisers (siehe [Ill08] und [Sol06]) für Beschreibungslogiken zurückgreifen. Wie bereits vorher erwähnt, nutzen die Autoren ausschließlich die Repräsentation beschreibungslogischer Ausdrücke. Es wird nicht über die CRIL-Netze unter Benutzung des Beweisers räsoniert.

- Das Formulieren von Regeln ist, für einen Menschen, verständlicher als der ursprünglich angedachte Ansatz, der die Anreicherung mit speziellen CRIL-Netzen vorsah. Diese sind umständlicher zu handhaben, da es, im Gegensatz zu einen beschreibungslogischen Ausdruck, mindestens zwei Informationsebenen in einen CRIL-Netz gibt: die Struktur

des Netzes, also die Existenz von Knoten und Relationen und die Typen der Relationen und die Attribute der jeweiligen Knoten. Der Vorteil eines CRIL-Netzes, Information in einer kompakten Form tragen zu können, macht sich durch eine gewisse Unübersichtlichkeit bemerkbar. In diesen Kontext erscheint ein beschreibungslogischer Ausdruck praktikabler. Dieser Umstand schlägt sich auch in den Aufwänden um die Benutzerschnittstellen, das Datenmodell und Ähnliches zu Implementieren, nieder.

1.2.2 Limitationen

In seiner jetzigen Form unterliegt der Ansatz einigen Limitationen, die hier kurz genannt werden:

- Quantitative, geometrische Informationen können nicht direkt abgefragt werden, da diese Informationen in den CRIL-Graphen nicht enthalten sind. Beispielsweise ist es durchaus möglich zu erfragen, ob eine Landmarke vor einer anderen Landmarke liegt, aber nicht wie weit davor, also ist eine Quantifizierung der Relation ,,vor" nicht möglich. Diese Limitation ist der Struktur der CRIL-Graphen geschuldet und weniger den Beschreibungslogiken im Allgemeinen. Allerdings ist der von uns gewählte Dialekt nicht in der Lage, quantifizierte Sachverhalte darzustellen.

- Die Möglichkeit, über Regelmengen und die Taxonomie zu räsonieren, wenn man beide in der T-Box der Beschreibungslogik zusammenfasst, wurde wegen des Aufwands nicht weiter verfolgt. Ebenso könnte man den Zustand des jeweiligen CRIL-Graphen als A-Box interpretieren, was zusammen mit oben beschriebener T-Box ein sehr mächtiges System hervorbringen würde. Dieses wurde ebenso unterlassen, da wir u.a. nicht dem Muster früherer Projektgruppen verfallen wollten, den Weltzustand in eine jeweils eigene Repräsentation (eigenes Format) zu überführen.

- Die Anreicherung mit Attributen bzw. Prädikaten kann nur an schon bestehende Knoten in einen CRIL-Graphen erfolgen, die Struktur des CRIL-Graphen wird nicht verändert. Es können keine Knoten oder Relationen hinzugefügt bzw. entfernt werden.

- Es können nur in der Taxonomie des GA vorhandene Konzepte augmentiert werden. Das bedeutet, dass so lange die Taxonomie des GA statisch vordefiniert ist, die Anreicherung und somit auch die Formulierung der Regeln auf Konzepte dieser Menge beschränkt sind. Sollte sich

die Taxonomie des GA eines Tages ändern, kann die Anreicherung unmittelbar davon Gebrauch machen. Das Benutzerinterface prüft beim Erstellen einer Regel, ob die verwendeten Konzepte und Rollen in der Taxonomie vorhanden sind und verhindern bei Nichtvorhandensein die Erstellung.

- Zyklische Regeln bzw. Regelmengen[2], werden nicht gesondert von unseren Ansatz behandelt. Die Autoren haben die Ausführung solcher Regeln bzw. Regelmengen ausgeschlossen, da nicht garantiert werden kann dass, nach einmaliger Ausführung die intendierten Änderungen sich im CRIL-Netz manifestiert haben. Um dieses zu garantieren müsste ein Abhängigkeitsbaum erstellt und die Regeln entsprechend der relativen Ordnung des Baumes Ausgeführt werden.

2 Theoretische Grundlagen

Die theoretischen Grundlagen, auf denen die regelhafte Anreicherung von CRIL-Netzen aufbaut, sollen in den nächsten Abschnitten kurz eingeführt werden. Dabei wird zuerst auf die Bedeutung und Eigenschaften der Taxonomien des GA eingegangen, um dann die *Semantic Representation Language* vorzustellen. Aufbauend auf diesen werden dann CRIL-Graphen und schlussendlich Beschreibungslogiken eingeführt.

2.1 Taxonomien

Zuerst werden wir auf Taxonomien eingehen, um später einige Bestandteile von SRL und CRIL-Graphen greifbarer zu machen.

Eine Taxonomie sei eine Vererbungshierachie für Konzepte, wie sie in [Hel03, S. 39] eingeführt wurde. Weiterhin können sich in dieser Taxonomie Konzepte auch gegenseitig ausschließen. So sind zum Beispiel sowohl Vögel als auch Menschen Lebewesen und erben deren Eigenschaften, aber es kann niemals ein Mensch ein Vogel sein oder umgekehrt. Diese Exklusion wird wiederum weiter vererbt. Demnach benötigt eine Taxonomie eine Menge von Konzepten, eine Vererbungsrelation (auch *Subsumptionsrelation* genannt) und eine Exklusionsrelation. In einer solchen Taxonomie sei das ,,Top"-Konzept ($\top$) enthalten, welches alle anderen Konzepte subsumiert.

[2]Zyklische Regelmenge: eine Menge von Regeln, in denen die Konzepte, die eine Regel nach Anwendung dem CRIL-Netz hinzufügt, von anderen Regeln in der selben Menge genutzt werden.

Zwei solche Taxonomien werden später als Grundlage für die Term- und Formel-Mengen in SRL benutzt. Die erste Taxonomie sei die Taxonomie von Attributen (Terme). In dieser befinden sich einstellige Operatoren, also essentielle[3] Eigenschaften. Hiermit können die sogenannten Sortenhierarchien (Verträglichkeitshierarchien) aufgebaut und Sorten derart verglichen werden, so dass man unsinnige Belegungen wie *hat_farbe ('Auto', 'Hans')* verhindern kann. Allerdings soll in diesem Bericht nicht weiter auf Sortenverträglichkeit eingegangen werden. Interessierte können dies in [Hab86, S. 65] nachschlagen. Die andere Taxonomie, die Taxonomie der Relationen (Formeln), enthält die Relationen, also mehrstellige Operatoren. Diese Trennung macht Sinn, zumal beim Vergleich von CRIL-Graphen diese Taxonomien in verschiedenen Kontexten verglichen werden (ist etwas sortenvertäglich / wie ähnlich sind einzelne CRIL-Knoten / wie ähnlich sind CRIL-Graphen).

Definition 2.1. *Taxonomien*[4]
Sei:

- $S :=$ *eine endliche Menge von Konzepten mit* $\top \in S$,
- $(\leq) \subseteq (S \times S) :=$ *die Subsumptionsrelation, welche eine Halbordnung, also reflexiv, antisymmetrisch und transitiv, jedoch nicht zwangsweise linear ist und Konzepte auf Elternkonzepte abbildet, wobei* $\top$ *ein Elternkonzept ist, von dem jedes andere Konzept erbt.*
- $(\oplus) \subseteq (S \times S) :=$ *eine Exklusionsrelation, welche irreflexiv ist und über die Subsumptionsrelation wie eine Eigenschaft vererbt wird,*

Dann ist:

- $T = (S, \leq, \oplus)$ *eine Taxonomie.*

Weiterhin sei:

- $S_A :=$ *eine endliche Menge von Attributen mit* $\top \in S_A$,
- $S_R :=$ *eine endliche Menge von Relationen mit* $\top \in S_R$,

Und damit ist:

- T_A *die Taxonomie der Attribute mit* S_A *als Grundmenge,*
- T_R *die Taxonomie der Relationen mit* S_R *als Grundmenge.*

[3] Was genau der Begriff „essentiell" bedeuten soll kann in [Hab86, S.163] nachgelesen werden.

[4] Vergl. [Hel03, Def. 3.2.1, S. 39].

Durch die Taxonomien kann nun festgestellt werden, wie ähnlich zwei zu vergleichende Konzepte sind, was wiederum notwendig ist, um die wahrgenommene Welt (in Form eines Perzeptions-CRIL-Graphen) mit den Instruktionen (in Form eines Instuktions-CRIL-Graphen) zu vergleichen. Da das ,,Top"-Konzept ($\top$) alle anderen Konzepte subsumiert, kann man alle Konzepte anhand ihrer Eltern-Konzepte vergleichen, denn $\top$ ist ein allen in den Taxonomien enthaltenen Konzepten gemeinsamer Vorfahre und dementsprechend gibt es immer mindestens einen Elternknoten, welcher beide zu vergleichenden Konzepte subsumiert. Am sinnvollsten ist ein solch angestellter Vergleich jedoch mit Hilfe des Eltern-Konzeptes E, das beide Konzepte subsumiert, aber selber kein anderes Konzept subsumiert, welches ebenfalls die beiden Vergleichs-Konzepte subsumiert. Dies hängt damit zusammen, dass Konzepte durch die Vererbungshierachie immer mehr Eigenschaften gewinnen je weniger sie dem ,,Top"-Konzept ähneln. Das Eltern-Konzept E besitzt also die maximale Anzahl an Eigenschaften, welche beiden Vergleichs-Konzepten gemeinsam sind. Von hier aus wird der Abstand zwischen den jeweiligen Kinder-Konzepten anhand eines Präzedenzgraphen (Hasse-Diagramm) gemessen. Je geringer der Abstand zwischen zwei Konzepten ist, desto mehr ähneln sie sich, da bei jedem Schritt zu einem allgemeineren Konzept Eigenschaften/Informationen verloren gehen und bei jedem Schritt zu einem spezielleren Konzept neue Informationen hinzukommen.

Zu guter Letzt wollen wir ein Beispiel für eine Taxonomie geben. Seien:

- $S_A = \{\top$, gebäude, hochhaus, einfamilienhaus, farbe, rot, bordeauxrot, blau$\}$ eine Menge von Attributen,
- $(\leq_A) = \{$(gebäude, $\top$), (farbe, $\top$), (hochhaus, gebäude), (einfamilienhaus, gebäude), (rot, farbe), (blau, farbe), (bordeauxrot, rot)$\}$ die Subsumptionsrelation, wobei das Vorhandensein von (A, B) bedeutet, dass das Konzept B das Konzept A subsumiert,
- $(\oplus_A) = \{$(gebäude, farbe)$\}$ die Exklusionsrelation.

Dann ist $T_A = (S_A, \leq_A, \oplus_A)$ eine Taxonomie von Attributen. In dieser Taxonomie wird unter anderem ausgedrückt, dass sowohl Einfamilienhäuser als auch Hochhäuser beide zu den Gebäuden zählen, Rot eine Farbe und Bordeauxrot zur Kategorie der Rottöne (und damit ebenfalls zu den Farben) gehört. Weiterhin wird festgehalten, dass Farben keine Gebäude und Gebäude keine Farben sind. Diese Eigenschaften werden mit der Subsumptionsrelation weitervererbt und damit steht fest, dass z.B. auch Bordeauxrot kein Einfamilienhaus und umgekehrt sein kann.

2.2 Semantic Representation Language

Die Semantic Representation Language (SRL) ist eine logik-orientierte[5], propositionale[6] Repräsentationssprache, die ursprünglich zur semantischen Repräsentation und Analyse von natürlich sprachlichen Texten entworfen wurde [Hab86, S. 56]. Die Unterscheidung zwischen Termen und Formeln wird in SRL allerdings implizit über die sogenannte Typfunktion vorgenommen.

SRL zu verwenden bietet sich aus zwei Gründen an: Erstens kann SRL als Repräsentationssprache offensichtlich dazu benutzt werden, um das Welt-Wissen des GA zu repräsentieren, also die Informationen, welche der GA aus seiner Umwelt ableitet. Weiterhin macht die Spezialisierung von SRL auf natürlichsprachliche Texte diese Sprache ideal zum Repräsentieren der natürlichsprachlichen Instruktionen.

Definition 2.2. *Die Syntax der Sprache SRL*[7]
Gegeben seien eine Menge S_A von Attributen und eine Menge S_R von Relationen. Dann sei das Alphabet von SRL definiert durch:

- *VAR := eine abzählbare Menge von (Term-) Variablen (mit $VAR \equiv S_A$)*
- *OP := eine endliche Menge von Operatoren (mit $OP \equiv S_R \cup S_A$)*
- *$EFF := \{t, f\}$ eine Menge von Effekten (mit $t \mathrel{\widehat{=}}$ termbildender Operator und $f \mathrel{\widehat{=}}$ formelbildender Operator)*

Weiterhin gehören auch die logischen Symbole

- *von logischen Junktoren $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$*
- *sowie die Quantoren $\exists, \forall$*
- *als auch Hilfssymbole wie Klammern und Kommata zum Inventar der Operatoren.*

Ferner wird die Typfunktion für Operatoren folgendermaßen definiert: $TYP : OP \rightarrow EFF \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Das von $TYP(OP)$ erzeugte Tupel $\langle$ eff, m, n, p$\rangle$ erklärt sich durch:

- *eff $\in \{t, f\}$ der Effekt, der dem Operator zugeordnet ist,*
- *m = Anzahl der durch den Operator gebundenen Variablen,*

[5] zwischen Termen und Formeln wird unterschieden
[6] propositional: interpretationsabhängig
[7] Vergl. [Hab86, S. 59 ff].

- *n = Anzahl der Terme und*
- *p = Anzahl der Formeln, die der Operator als Argument besitzt.*

Seien weiterhin über ein induktives Schema definiert:

- *Die Menge TER der Terme:*
  - *Jede Variable* v $\in VAR$ *ist ein Term.*
  - *Wenn* op $\in OP$ *und typ*(op) $= \langle$t,m,n,$p\rangle$*, wobei:*
    - t *anzeigt, dass* op *ein termbildender Operator ist,*
    - $x_1, \cdots, x_m \in VAR$,
    - $t_1, \cdots, t_n \in TER$ *und*
    - $f_1, \cdots, f_p \in FOR$ *sind,*

    dann ist auch op $(x_1, \cdots, t_1, \cdots, f_1, \cdots, f_p) \in TER$.
  - *Dies sind alle Terme.*
- *Die Menge FOR der Formeln:*
  - *Wenn* op $\in OP$ *und typ*(op) $= \langle$f,m,n,$p\rangle$*, wobei:*
    - f *anzeigt, dass* op *ein formelbildender Operator ist,*
    - $x_1, \cdots, x_m \in VAR$,
    - $t_1, \cdots, t_n \in TER$ *und*
    - $f_1, \cdots, f_p \in FOR$ *sind,*

    dann ist auch op $(x_1, \cdots, t_1, \cdots, f_1, \cdots, f_p) \in FOR$.
  - *Dies sind alle Formeln.*

Da sowohl Terme als auch Formeln als wohlgeformte SRL-Ausdrücke angesehen werden, aber trotzdem die Unterscheidung zwischen Termen und Formeln möglich sein soll, wird SRL als Paar von Mengen aufgefasst:

$$SRL = \langle TER, FOR \rangle$$

Wir haben schon vorher angedeutet, dass SRL dazu benutzt werden kann, um das Welt-Wissen des GA zu repräsentieren. Dabei stellt sich nun die Frage, wie das geschehen soll. Nach [Hab86, S. 61] repräsentieren *SRL*-Ausdrücke mentale Zustände (und damit Wissen/Annahmen über die reale Welt) und *SRL*-Operatoren Konzepte. Was mentale Zustände genau sind wird weder in [Hab86] noch in diesem Bericht geklärt. Habel verweist stattdessen auf Arbeiten aus psychologischen und philosophischen Bereichen. Konzepte hingegen

werden - ebenfalls weitgehend undefiniert - als Gesammtheit von Bedeutungen, inklusive Erwartungen und Standardannahmen, zu denen unter anderem auch Beziehungen zwischen den Konzepten selbst gehören, angesehen. Demnach werden auch Relationen als Konzepte behandelt und die Vereinigung der vorher eingeführten Taxonomien T_A und T_R deckt alle dem GA bekannten Konzepte ab.

Als Beispiel für ein Konzept in T_R sei hier *fliegen* gegeben. Der Satz ,,Hans fliegt nach Mallorca" kann durch *fliegen ('Hans', 'Mallorca')* repräsentiert werden. Hieran ist gut zu erkennen, dass sich Konzepte nicht nur zum Darstellen von physikalischen Objekten eignen, sondern auch Gegenstände des Denkens repräsentieren können. Des Weiteren können Konzepte Eigenschaften besitzen, über welche keine weiteren Aussagen gemacht werden sollen. Dies macht Sinn, da gewisse Eigenschaften in vielen Fällen nicht weiter erläutert zu werden brauchen, es jedoch möglich sein kann, auch diesen Eigenschaften wieder Eigenschaften zuzuweisen wie z.B. *besitzer_von ('Hans', 'Automobil534')* und weiterhin *farbe_von ('Automobil534', 'rot')*. Für die Wiedererkennung des Automobiles auf einem Parkplatz macht es Sinn, die Farbe des Fahrzeugs zu kennen, jedoch ist diese Information ohne Belang, wenn es darum geht, das Automobil mit dem korrekten Treibstoff zu betanken.

SRL ist nach [Hel03, S. 31] die Grundlage für referentielle Netze, welche weiterhin auf CRIL-Graphen abgebilded werden können und umgekehrt [Hel03, S. 45]. Daher kann man sagen, dass SRL auch als Grundlage für CRIL-Graphen dient.

Genauer möchten wir im Rahmen dieses Berichts nicht auf SRL oder referntielle Netze eingehen. Näheres findet man in [Hab86, S. 56 ff] bzw. [Hab86, S. 111 ff].

2.3 CRIL-Graphen

Bei der Definition von CRIL-Graphen orientieren wir uns an [Hel03, S. 45 ff]. Ein CRIL-Graph besteht aus einer Menge von CRIL-Knoten und gerichteten Kanten zwischen den Knoten. Die Knoten sind dabei gleichbedeutend mit Referenzobjekten, welche sowohl essentielle Eigenschaften aus T_A wie z.B. Namen, als auch Relationen aus T_R zu anderen CRIL-Knoten als Eigenschaften besitzen. Da diese Relationen aus T_R als Kanten des CRIL-Graphen angesehen werden und mehr als nur zweistellige Operatoren in T_R enthalten sein können, benötigt man eine Operatorfunktion, um zu repräsentieren, welcher Operator welche Kante darstellt.

Definition 2.3. *Ein CRIL-Graph und seine Teile*[8]
Gegeben seien eine Taxonomie von Attributen T_A mit einer Menge von Attributen S_A und eine Taxonomie von Relationen T_R mit einer Menge von Relationen S_R. Ein CRIL-Knoten $ck = (SOR, EIG, NAM) \in V$ ist ein Tripel, bestehend aus einer Menge SOR von Sorten, für die gilt $SOR \subseteq S_A$, einer Menge EIG von Eigenschaften, für die gilt $EIG \subseteq S_A$ sowie einer Menge NAM, deren Elemente Namen sind. Es werden ferner die folgenden Funktionen über einen CRIL-Knoten definiert, um Aussagen über Sorte, Eigenschaften und Namen machen zu können:

- $\mathcal{S}(ck) = SOR$
- $\mathcal{E}(ck) = EIG$
- $\mathcal{N}(ck) = NAM$

Ein CRIL-Graph $G = (V, E, T_A, T_R, op)$ besteht aus einer Menge V von CRIL-Knoten, einer Menge E von gerichteten Kanten mit $E \subseteq (V^2 \cup V^3 \cup \cdots)$, den Taxonomien T_A und T_R und der Operatorfunktion über die Kanten $op : E \to S_R$. Weiterhin wird definiert:

- $\mathcal{V}(G) = V$
- $\mathcal{E}(G) = E$

CRIL-Graphen werden an zwei Stellen im GA zur Wissensrepräsentation benutzt: Einmal zur Repräsentation der Instruktionen in Form eines Aktionsplans und weiterhin zur Repräsentation der Perzeption der Welt, in welcher sich der GA befinded. Die dabei entstehenden Abweichungen der realen Welt mit der internen Repräsentation sind erwünscht, denn auch Menschen benutzen zur Routenplanung eine unvollständige und manchmal von der Realität abweichende Repräsentation der realen Welt mit einigen markanten Landmarken anstatt eines möglichst vollständigen Welt-Modells.

Zwischen diesen beiden CRIL-Graphen wird mit Hilfe der Taxonomien verglichen um festzustellen, welche Aktion als nächstes ausgeführt werden sollte. Unser (CRIL-)Augmentor versucht nun anhand von voher definierten Regeln eben diese CRIL-Netze um ,,versteckte" Informationen zu erweitern bzw. implizite Informationen zu explizieren. Diese Regeln werden hierbei mit Hilfe von Beschreibungslogiken verfasst.

Abschließend wollen wir ein Beispiel für ein CRIL-Graphen geben. Dafür nutzen wir die in einem vorherigen Beispiel konstruierte Taxonomie T_A[9]. Weiterhin definieren wir:

[8] Vergl. [Hel03, Def. 3.3.1, S. 46].
[9] Siehe Abschnitt 2.1

- $S_R = \{\top, \text{vor}(2)\}$ eine Menge von Relationen,
- $(\leq_R) = \{(\text{vor}(2), \top)\}$ die Subsumptionsrelation,
- $(\oplus_R) = \{\}$ die Exklusionsrelation.

Dann ist $T_R = (S_R, \leq_R, \oplus_R)$ eine Taxonomie von Relationen. Durch die vorher erwähnten Sorten[10] ist es möglich, über eine Sortenhierarchie unsinnige Belegungen zu erkennen, indem man verlangt, dass nur *vor (gebäude, gebäude)* eine sinnvolle Belegung darstellen soll.

Erhält der GA nun Informationen über die Welt fasst er diese in einem CRIL-Graphen zusammen (entweder entsteht ein Perzeptions-Graph mit Hilfe der Informationen, die der GA aus seiner Perzeption erhält oder es entsteht ein Instruktions-Graph, erstellt aus den Informationen, welche der GA aus einer Instruktion erhalten hat), indem er diese Informationen gemäß seiner Möglichkeiten bezüglich der ihm zu Grunde liegenden Taxonomien interpretiert. Perzeptiert der GA nun ein rotes Einfamilienhaus und dahinter ein blaues Hochhaus, so wäre ein möglicher CRIL-Graph einer mit 2 CRIL-Knoten *ck1* und *ck2*. Hierbei besitze *ck1* als Sortenmenge $\mathcal{S}1 = \{einfamilienhaus\}$, als Namenmenge $\mathcal{N}1 = \{haus1\}$ und als Eigenschaftsmenge $\mathcal{E}1 = \{rot, einfamilienhaus\}$, während *ck2* die Eigenschaften $\mathcal{S}2 = \{hochhaus\}$, $\mathcal{N}2 = \{haus2\}$ und $\mathcal{E}2 = \{blau, hochhaus\}$ besitzt. Die einzige Kante des CRIL-Graphen wäre dann *e1* mit *vor(ck1, ck2)*. Weiterhin wäre in der Operatorfunktion op die Information enthalten, dass *vor(arg1, arg2)* bedeutet, dass das Objekt, welches durch *arg1* repräsentiert wird, sich räumlich vor dem befinded, welches durch *arg2* repräsentiert wird.

2.4 Beschreibungslogiken

Beschreibungslogiken sind eine Familie formaler Sprachen, die vornehmlich zur Wissensrepräsentation und zum Räsonieren über Wissensbasen eingesetzt werden. Sie zeichnen sich gegenüber ihren Vorgängern, Semantische Netzwerke und Frames, besonders durch eine klare, logik-basierte Semantik aus [NB03, Baa09].

Bis in die 1970er Jahre hinein lag der Fokus der Forschung auf dem Gebiet der Wissensrepräsentation und -verarbeitung vornehmlich auf Ansätzen, die nicht auf Logik basierten, sondern kognitiven Prozessen beim Menschen angelehnt waren [BL06]. Die entwickelten Systeme waren zumeist netzartige Strukturen, in denen Entitäten und Beziehungen zwischen Entitäten einer Wissensbasis modelliert wurden und regelbasierte Verfahren zur Manipulation und zur Schlussführung über Wissensbasen eingesetzt wueden.

[10] Siehe Abschnitt 2.1

Eines der Hauptprobleme dieser Methoden war das Zuweisen einer eindeutigen Semantik. In [Hay79] wurde festgestellt, dass die Semantik von Frames durch Prädikatenlogik 1. Ordnung (im Folgenden FOPC[11]) gegeben werden kann. Weitere Untersuchungen ergaben darüber hinaus, dass Frames, wie auch Semantische Netze, nicht zwangsläufig die volle Ausdrucksmächtigkeit von FOPC und daher auch keine vollen FOPC-Theorembeweiser benötigen. Vielmehr können sie als syntaktische Varianten von Fragmenten der Prädikatenlotgik angesehen werden [NB03].

Beschreibungslogiken sind die Konsequenz dieser Erkenntnisse. Der Name soll einerseits den Hauptzweck, die Beschreibung von Klassen und Objekten einer Domäne, verdeutlichen und andererseits den Fokus auf eine strikte, formale Semantik legen. Auf Basis einer Konzeptsprache[12], welche Syntax und Semantik der zulässigen, beschreibungslogischen Ausdrücke definiert, enthält eine Beschreibunglogik drei Komponenten: den terminologischen und den assertionalen Formalismus (im Folgenden auch TBox und ABox genannt) und eine Reasoning-Komponente. Die TBox kann als Taxonomie der modellierten Welt angesehen werden, während in der ABox Wissen über konkrete Individuen und ihre Beziehungen untereinander enthalten ist. In der Reasoning-Komponente können die ontologisch interessanten Standardinferenzen, wie Subsumption, Konsistenz, Erfüllbarkeit und das Instanzproblem enthalten sein, aber auch komplexere Schlüsse, wie die kleinste gemeinsame Oberbeschreibung (least common subsumer).

Wie üblich wird eine größere Ausdruckmächtigkeit seitens der Konzeptsprache meistens mit einer steigenden Komplexität von Inferenzproblemen erkauft. Seit den frühen 90er Jahren sind Komplexität und Berechenbarkeit von Standardinferenzproblemen in Beschreibungslogiken einer gründlichen Analyse unterzogen worden [BL06]. Einen guten Überblick bietet der Description Logic Complexity Navigator[13].

2.4.1 Konzeptsprachen

Eine Konzeptsprache besteht aus einer abzählbar unendlichen Menge von (atomaren) Konzept- und Rollenbezeichnungen und einer Menge von Konstruktoren, mit deren Hilfe komplexe Beschreibungen aus atomaren Konzepten und Rollen zusammengesetzt werden können. In der Literatur häufig diskutierte Konstruktoren umfassen u.a. Konjuktion, Disjunktion, Negation, Existenz- und Allquantoren, sowie spezielle Konzepte wie das allumfassende und das leere Konzept. Im Folgenden bezeichnen A, B üblicherweise atomare

[11] FOPC = First-Order Predicate Calculus

[12] in der Literatur oft auch als „Beschreibungssprache" bezeichnet

[13] http://www.cs.man.ac.uk/∼ezolin/dl/

Name	Syntax	Semantik
Top-Konzept	$\top$	$\Delta^{\mathcal{I}}$
Bottom-Konzept	$\bot$	$\emptyset$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Konjunktion	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunktion	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Werterestriktion	$\forall r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \forall e.(d,e) \in r^{\mathcal{I}} \rightarrow e \in C^{\mathcal{I}}\}$
Existenzrestriktion	$\exists r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$
Transitive Rollen	r	$r^{\mathcal{I}}$ ist transitiv
Inverse Rollen	r^-	$\{(d,e) \mid (e,d) \in r^{\mathcal{I}}\}$
Nominale	I	$I^{\mathcal{I}}$ ist ein Singleton
Qualifizierte Anzahl-restriktion	$\leq n\, r\, C$ $\geq n\, r\, C$	$\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}} \mid e \in C^{\mathcal{I}}\} \leq n\}$ $\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}} \mid e \in C^{\mathcal{I}}\} \geq n\}$
(nicht qualifizierte) Anzahlrestriktion	$\leq n\, r$ $\geq n\, r$	$\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}}\} \leq n\}$ $\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}}\} \geq n\}$

Tabelle 1: Semantik von Konzept- und Rollenkonstruktoren. Hier soll r für eine atomare Rolle, C und D für komplexe Konzeptausdrücke und n für eine natürliche Zahl stehen.

Konzepte, r und s atomare Rollen, während C und D für komplexe Ausdrücke stehen.

Die Semantik beschreibungslogischer Ausdrücke wird durch *Interpretationen* gegeben. Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, bestehend aus einer nichtleeren Menge $\Delta^{\mathcal{I}}$ (Domäne) und der Interpretationsfunktion $\cdot^{\mathcal{I}}$, bildet atomare Konzepte auf Mengen $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ und atomare Rollen r auf binäre Relationen $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ ab. Die Tabelle 1 (entnommen aus [BL06]) zeigt die induktive Erweiterung der Interpretationsfunktion auf komplexe Ausdrücke. Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ wird als *Modell* eines Konzeptausdrucks C bezeichnet, falls $C^{\mathcal{I}} \neq \emptyset$ gilt.

In der Betrachtung von propositional abgeschlossenen Konzeptsprachen wird häufig die von Schmidt-Schauß und Smolka in [SSS91] eingeführte elementare Konzeptsprache $\mathcal{ALC}$ als Grundlage genommen. Sie enthält neben Konjunktion, Disjunktion und Negation noch Existenzrestriktion, Werterestriktion sowie die Konzepte Top und Bottom. Darauf aufbauend hat sich ein Namensschema entwickelt, welches die enthaltenen Konstruktoren durch Buchstaben abkürzt:

Symbol	Syntax
$\mathcal{S}$	$\mathcal{ALC}$ mit transitiven Rollen
$\mathcal{H}$	$r \sqsubseteq s$
$\mathcal{I}$	r^-
$\mathcal{O}$	Nominale
$\mathcal{N}$	$(\leq n\, r), (\geq n\, r)$
$\mathcal{Q}$	$(\leq n\, r\, C), (\geq n\, r\, C)$

In der Tabelle fallen die Rollenhierarchien eigentlich nicht in den Bereich der Konzeptsprache, sondern eher zum terminologischen Formalismus. Durch sie werden zulässige Interpretationen weiter eingeschränkt, da $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ gefordert wird. Sie verhalten sich dementsprechend eher wie ,,General Concept Inclusions", die im nächsten Abschnitt genauer besprochen werden und eine Form von TBoxen darstellen.

2.4.2 Terminologischer Formalismus

Mit Hilfe einer Konzeptsprache sind wir in der Lage, komplexe Rollen- und Konzeptausdrücke zu formulieren. Um diese jedoch in Beziehung zueinander zu setzen benötigt man den in diesem Abschitt eingeführten Teil einer Beschreibungslogik.

Im simpelsten Fall ist eine TBox $\mathcal{T}$ eine Sammlung von *Konzeptdefinitionen* (auch *Axiome*). Es handelt sich dabei um Ausdrücke der Form $A \equiv C$, wobei A ein atomarer Konzeptname und C eine (komplexe) Konzeptbeschreibung ist. Dadurch muss nun zwischen *primitiven* und *definierten Konzepten* unterschieden werden. Ein Konzeptname wird *definiert* genannt, wenn er auf der linken Seite einer Konzeptdefinition auftritt. Ein Konzeptname darf innerhalb einer TBox nicht mehrfach definiert werden, alle anderen Konzeptnamen heißen *primitiv*. Desweiteren muss zwischen *zyklischen* und *azyklischen TBoxen* unterschieden werden:

Auf der Menge der definierten Konzepte in einer TBox $\mathcal{T}$ sei die Relation $\prec_{\mathcal{T}}$ gegeben durch ,,$A \prec_{\mathcal{T}} B$ gdw. $A \equiv C$ in $\mathcal{T}$ enthalten ist und B in der Konzeptbeschreibung C vorkommt". Sei nun $\prec_{\mathcal{T}}^+$ die transitive Hülle von $\prec_{\mathcal{T}}$, dann enthält $\mathcal{T}$ einen *terminologischen Zyklus*, falls $A \prec_{\mathcal{T}}^+ A$ für ein definiertes Konzept A in $\mathcal{T}$ gilt.

Es gibt verschiedene Typen von terminologischen Zyklen, die in [Neb91] genauer aufgeschlüsselt werden. Man kann sie z.B. benutzen um rekursive Strukturen und Beziehungen zu beschreiben:

$$\mathsf{BinaryTree} \equiv \mathsf{Tree} \sqcap \leq 2\, \mathsf{branch} \sqcap \forall \mathsf{branch}.\mathsf{BinaryTree} \tag{1}$$

$$\mathsf{Human} \equiv \mathsf{Mammal} \sqcap \forall \mathsf{parent}.\mathsf{Human} \tag{2}$$

Die Konzeptdefinitionen 1 und 2 illustrieren den Nutzen von terminologischen Zyklen. Die Definition 1 ist ein Beispiel für die Modellierung einer rekursiven Struktur. Sie besagt, ein binärer Baum sei ein Baum mit maximal zwei Ästen und alle Äste müssen wiederum binäre Bäume sein. Im zweiten Beispiel werden Menschen als Säugetiere mit ausschließlich menschlichen Eltern definiert, was evolutionsbiologisch anfechtbar ist, aber zumindest für Menschen im Alltag als Definition ausreichend ist.

Eine TBox ohne terminologische Zyklen kann auch als Sammlung von Abkürzungen für komplexe Konzeptausdrücke angesehen werden. Diese können explizit gemacht werden, indem alle definierten Konzepte A mit $A \equiv C \in \mathcal{T}$ in allen Konzeptdefinitionen durch ihre komplexen Ausdrücke C ersetzt werden. Dieser Vorgang wird *Expansion* genannt. Durch diese Eigenschaft ist es möglich für azyklische TBoxen *beschreibende Semantiken* zu verwenden. Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *erfüllt* ein Axiom $A \equiv C$ einer TBox $\mathcal{T}$, falls $A^{\mathcal{I}} = C^{\mathcal{I}}$. Erfüllt eine Interpretation alle Axiome einer TBox ist sie ein *Modell* der TBox.

Sei $\mathcal{I}$ eine Interpretation einer azyklischen TBox $\mathcal{T}$, die nur die primitiven Konzepte und Rollen in $\mathcal{T}$ interpretiert, dann wird $\mathcal{I}$ als *primitive Interpretation* bezeichnet. Eine (vollständige) Interpretation $\mathcal{J}$ von $\mathcal{T}$, die mit $\mathcal{I}$ in allen primitiven Konzepten und Rollen übereinstimmt, wird *Erweiterung von* $\mathcal{I}$ genannt. Eine TBox $\mathcal{T}$ heißt *definierend*, falls es zu jeder primitiven Interpretation nur eine Erweiterung gibt, die ein Modell von $\mathcal{T}$ ist. Es ist leicht einzusehen, dass azyklische TBoxen mit beschreibenden Semantiken immer definierend sind: Expandiert man die definierten Konzepte und Rollen einer azyklischen TBox, ist ihre Interpretation in Gänze durch die Interpretation der primitiven Konzepte und Rollen und der Konstruktoren der Konzeptsprache gegeben. So lässt sich eine primitive Interpretation $\mathcal{I}$ einer azyklischen TBox nur auf eine einzige Weise zu einer vollen Interpretation erweitern, die gleichzeitig auch ein Modell der TBox ist.

Die Eigenschaft, definierend zu sein, lässt sich mit beschreibenden Semantiken nicht aufrecht erhalten, sobald terminologische Zyklen in der TBox zugelassen werden [BL06]. Bereits für sehr einfache zyklische TBoxen lassen sich primitive Interpretationen auf verschiedene Weisen zu (vollen) Interpretationen erweitern, die Modelle der TBoxen sind. Eine vereinfachte Version des vorigen Beispiels der Definition von „Human" soll hier ausreichen:

$$\mathsf{Human} \equiv \forall \mathsf{parent}.\mathsf{Human} \tag{3}$$

Sei $\mathcal{I} = (\{d\}, \cdot^{\mathcal{I}})$ eine primitive Interpretation der TBox $\mathcal{T}$, die nur Beispiel 3 als Axiom enthält und $\cdot^{\mathcal{I}}$ sei gegeben durch $\mathsf{parent}^{\mathcal{I}} = \{(d, d)\}$. $\mathcal{I}$ lässt sich nun zu zwei verschiedenen Interpretationen von $\mathcal{T}$ erweitern, die Modelle von $\mathcal{T}$ sind.

Um definierende TBoxen bei zulässigen terminologischen Zyklen zu erhalten, muss die Interpretation durch Fixpunktsemantiken gegeben werden. Allerdings unterscheiden sich die Modelle einer TBox, je nach dem, ob man den größten oder den kleinsten Fixpunkt einer Konzeptdefinition verwendet. Die Verwendung beider Fixpunkte hat jeweils ihre Berechtigung und es ist bislang dem Konstrukteur einer Wissensbasis überlassen, zu entscheiden, wann welche Fixpunktsemantik verwendet werden soll. Ohne auf antropologische Feinheiten eingehen zu wollen, wäre bei einer Definition vom ,,menschlich" gemäß Beispiel 3 vielleicht der kleinste Fixpunkt als Semantik vorzuziehen, um nicht die Elemente verschiedener unter einer primitiven Interpretation von parent abgeschlossener, Teilmengen automatisch als Human zu klassifizieren. Auf der anderen Seite könnte für das Axiom

$$\mathsf{TopResearcher} \equiv \mathsf{Researcher} \sqcap \mathsf{Renowned} \sqcap \forall \mathsf{collaborates}.\mathsf{TopResearcher} \quad (4)$$

eine unter der collaborates-Relation abgeschlossene Gruppe von renomierten Forschern durchaus als Teilmenge der Interpretation von TopResearcher erwünscht sein. Durch Verwendung von Fixpunkt-Semantiken ist es also möglich, zyklische TBoxen definierend zu machen, allerdings muss von Fall zu Fall entschieden werden, welcher Fixpunkt zu den erwünschten Modellen führt.

Eine Möglichkeit beschreibende Semantiken für TBoxen zu verwenden, die nicht auf Konzeptdefinitionen beruht, sind *g concept inclusion axioms (GCIs)*. Diese Sicht auf eine TBox als Werkzeug, um zulässige Modelle einzuschränken und nicht Konzepte zu definieren, macht es möglich die Forderung nach definierenden TBoxen fallen zu lassen und somit beschreibende Semantiken zu verwenden:

Ein GCI-Axiom hat die Form $C \sqsubseteq D$, wobei C und D auch komplexe Konzeptbeschreibungen sein können. In dieser Form sind keine Einschränkungen für Syntax und Eindeutigkeit der rechten Seiten der Axiome notwendig. Eine Interpretation $\mathcal{I}$ erfüllt das Axiom $C \sqsubseteq D$, falls $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ gilt und ist weiterhin Modell einer TBox $\mathcal{T}$, falls sie alle Axiome in $\mathcal{T}$ erfüllt. Eine Konzeptdefinition $A \equiv C$ kann offensichtlich durch die Axiome $A \sqsubseteq C, C \sqsubseteq A$ als GCIs formuliert werden, wobei sich die Äquivalenz auf die Gleichheit der zulässigen Modelle beider Formen bezieht. Folglich sind auf GCIs beruhende TBoxen eine Verallgemeinerung von zyklischen und azyklischen TBoxen. Sie werden häufig auch als *generelle TBoxen* bezeichnet und werden von den meisten modernen DL-Systemen unterstützt [BL06, S. 9].

2.4.3 Assertionaler Formalismus

Der assertionale Formalismus enthält Wissen über die einen Weltausschnitt bevölkernden Individuen. Die einzelnen Individuen werden dabei durch Konzept- und Rollenzuordnungen beschrieben und untereinander in Verbindung gebracht. Man nennt Einträge im assertionalen Formalismus auch die *Fakten* einer Wissensbasis, wobei sich der Term *Wissensbasis* im Allgemeinen auf die Kombination aus einer ABox und einer TBox bezieht.

Eine ABox basiert auf einer abzählbar unendlichen Menge von Individuenbezeichnungen, im Folgenden a, b und c, die Konzept- und Rollenbeschreibungen zugeordnet werden:

$$\begin{aligned} &C(a) && (Konzeptzuordnung)\\ &r(a,b) && (Rollenzuordnung) \end{aligned}$$

Für eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ muss nun die Interpretationsfunktion auf Individuen erweitert werden, sodass jedem Individum der ABox ein Element der Domäne zugeordnet wird, also $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ für alle Individuen einer ABox. Hierbei ist es üblich unter der *unique name assumption* zu arbeiten und für die Interpretationsfunktion zu fordern, dass $a \neq b$ impliziert, dass auch $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. So werden verschiedenen Individuen der ABox auf jeden Fall auch verschiedene Elemente der Domäne zugeordnet.

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *erfüllt* eine Konzeptzuordnung $C(a)$, falls $a^{\mathcal{I}} \in C^{\mathcal{I}}$, und sie erfüllt eine Rollenzuordnung $r(a,b)$, falls $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. Sie wird *Modell* einer ABox $\mathcal{A}$ genannt, wenn sie alle Zuordnungen in $\mathcal{A}$ erfüllt.

2.4.4 Reasoning

Die Reasoning-Komponente ist der letzte Bestandteil einer Beschreibungslogik. Sie enthält Algorithmen zum Lösen von Inferenzproblemen, die mehreren Zielen dienen. Zum einen kann implizites Wissen in einer Wissensbasis expliziert werden, zum anderen helfen sie auch bei der Erstellung und Pflege von Wissensbasen. Unter den Standardinferenzproblemen wird zwischen den TBox-Inferenzen und den ABox-Inferenzen unterschieden. Die Standard-TBox-Inferenzen umfassen *Subsumption*, *Äquivalenz* und *Erfüllbarkeit*, die Standardinferenzprobleme der ABox sind *Konsistenz* und das *Instanzproblem*.

Gegeben seien eine TBox $\mathcal{T}$, eine ABox $\mathcal{A}$, zwei Konzeptbeschreibungen C und D und ein Individum a aus $\mathcal{A}$, dann sind die Standardinferenzen definiert als:

- C *subsumiert* D in Bezug auf $\mathcal{T}$ ($C \sqsubseteq_\mathcal{T} D$), falls $C^\mathcal{I} \subseteq D^\mathcal{I}$ für alle Modelle $\mathcal{I}$ von $\mathcal{T}$ gilt.
- C ist *äquivalent* zu D in Bezug auf $\mathcal{T}$ ($C \equiv_\mathcal{T} D$), falls $C^\mathcal{I} = D^\mathcal{I}$ für alle Modelle $\mathcal{I}$ von $\mathcal{T}$ gilt.
- C ist *erfüllbar* in Bezug auf $\mathcal{T}$, falls C und $\mathcal{T}$ ein gemeinsames Modell besitzen.
- $\mathcal{A}$ ist *konsistent* in Bezug auf $\mathcal{T}$, falls $\mathcal{A}$ und $\mathcal{T}$ ein gemeinsames Modell besitzen.
- a ist eine *Instanz* von C in Bezug auf $\mathcal{A}$ und $\mathcal{T}$ ($\mathcal{A} \models_\mathcal{T} C(a)$), falls $a^\mathcal{I} \in C^\mathcal{I}$ für alle Modelle $\mathcal{I}$ von $\mathcal{A}$ und $\mathcal{T}$ gilt.

Zwischen den Inferenzproblemen bestehen einige interessante Beziehungen, genauer gesagt können sie vielfach in polynomieller Zeit auf einander abgebildet werden [Baa09]. Für diese Reduktionen ist lediglich Konjunktion und Negation von komplexen Ausdrücken in der Beschreibungslogik erforderlich:[14]

- Subsumption und Äquivalenz

$$C \sqsubseteq_\mathcal{T} D \ \leq_p \ C \sqcap D \equiv_\mathcal{T} C$$
$$C \equiv_\mathcal{T} D \ \leq_p \ C \sqsubseteq D \text{ und } D \sqsubseteq C$$

- Subsumption und Erfüllbarkeit

$$C \sqsubseteq_\mathcal{T} D \ \leq_p \ C \sqcap \neg D \text{ unerfüllbar i.B.a. } \mathcal{T} \text{ ist}$$
$$C \text{ ist erfüllbar i.B.a. } \mathcal{T} \ \leq_p \ C \not\sqsubseteq_\mathcal{T} \bot$$

- Konsistenz und Instanzproblem

$$\mathcal{A} \text{ ist konsistent i.B.a. } \mathcal{T} \ \leq_p \ \mathcal{A} \not\models_\mathcal{T} \bot(a)$$
$$\mathcal{A} \models_\mathcal{T} C(a) \ \leq_p \ \mathcal{A} \cup \{\neg C(a)\} \text{ inkonsistent i.B.a. } \mathcal{T} \text{ ist}$$

Die Folge aus diesen Reduktionen ist, dass bei der Implementierung eines beschreibungslogischen Systems nicht für jedes Inferenzproblem ein eigener Algorithmus entworfen werden muss. Zumindest für die Standardinferenzen

[14] $\mathcal{T}$, $\mathcal{A}$, C, D und a wie zuvor, $\leq_p$ stehe hier und im Folgenden für eine Reduktion in polynomieller Zeit

ist es ausreichend, jeweils einen Algorithmus für die TBox- und die ABox-Inferenzen vorzuhalten und die übrigen Inferenzprobleme auf diese zu reduzieren. Dieser Ansatz wird in der Tat von vielen modernen DL-Systemen verwendet [BL06].

Es ist für viele Beschreibungslogiken möglich, sowohl azyklische TBoxen, als auch TBoxen mit GCIs zu *eliminieren*. Bereits in Kapitel 2.4.2 wurde angesprochen, dass azyklische TBoxen lediglich als Abkürzungen komplexer Konzeptausdrücke angesehen werden und durch Expansion explizit gemacht werden können. Für TBoxen mit GCIs ist das Eliminieren etwas komplizierter und gilt nur für sehr mächtige Beschreibungslogiken. In diesem Fall spricht man auch von *Räsonieren mit der leeren TBox* oder auch *ohne TBox* und Indizes für die TBox werden bei den formalen Symbolen für Subsumption, Äquivalenz und dem Instanzproblem weggelassen ($\sqsubseteq, \equiv, \models$ anstatt $\sqsubseteq_\mathcal{T}, \equiv_\mathcal{T}, \models_\mathcal{T}$). Obgleich die Elimination der TBox zu einem Anstieg der Komplexität der Konzeptbeschreibungen führt, können die Inferenzprobleme je nach Beschreibungslogik wesentlich leichter ohne TBox zu verarbeiten sein [Baa09]. Da das Elimieren von TBoxen für diese Arbeit jedoch irrelevant ist, wird an dieser Stelle nicht weiter darauf eingegangen.

3 Ansatzbeschreibung

Dieser Abschnitt soll einen Überblick über den Kern des Ansatzes geben, in dem genauer beschrieben wird, welche Teile der Beschreibungslogik und des GA Verwendung finden. Es soll die Wahl von Beschreibungslogiken für die Anreicherung motiviert und kurz die, durch den Ansatz eingeführten, Objekte bzw. Datenstrukturen beschrieben werden. Eine genauere Beschreibung der vorliegenden Implementation gibt Abschnitt 4 *Vorarbeiten und Implementation*.

Um eine Verbesserung des Navigationsverhaltens des geometrischen Agenten zu erreichen, wurde von den Autoren der Ansatz gewählt, CRIL-Netze mit zusätzlichen Informationen anzureichern. Da CRIL-Netze sowohl sämtliche Informationen über die Umwelt als auch über die Instruktionen des GA repräsentieren, erschien ein Vorgehen, welches auf dieser Ebene ansetzt, sinnvoll. Die Anreicherung mit zusätzlichen Informationen wird über Ausdrücke einer Beschreibungslogik formuliert. Dies hat mehre Vorteile, zuerst genannt sei, dass Beschreibungslogiken eine eindeutige Semantik besitzen, im Gegensatz zu den zuerst, zur Beschreibung einer Anreicherung, favorisierten CRIL-Netzen. Hier hätte erst eine, für diesen Zweck dienliche, Semantik definiert werden müssen. Zudem sind auch wegen ihrer klaren Semantik beschreibungslogische Ausdrücke sowohl für Menschen einfacher zu lesen, zu

formulieren und zu interpretieren als auch die algorithmische Verarbeitung, da wegen der semantischen Eindeutigkeit keine Unschärfen existieren und ein rekursives Abarbeiten der Teilausdrücke genügt. Schließlich sei noch erwähnt, das die Autoren sich einen, schon vorhandenen, Tableaubeweiser für Beschreibungslogiken zunutze machen konnten. So musste die Infrastruktur, um beschreibungslogische Ausdrücke zu verarbeiten, nicht implementiert, sondern die bereits Vorhandene nur an die Bedürfnisse der Autoren angepasst werden.

Folgend soll gezeigt werden, welche Komponenten der Beschreibungslogik genutzt werden, auf welche Weise diese auf Teile des GA Bezug nehmen und wie der Ansatz der Anreicherung von CRIL-Netzen durch beschreibungslogische Ausdrücke formuliert wird. Eine Beschreibungslogik besteht formal aus einer Konzeptsprache, einen assertionalen (*A-Box*), einen terminologischen Formalismus (*T-Box*) und einer Reasoning-Komponente (s. Abschnitt 2.5). Die *A-Box* und *T-Box* sowie die Reasoning-Komponente des Beweisers werden nicht genutzt, sondern nur der Parser und die Konzeptsprache. Dabei fungiert die statische Taxonomie des GA als *T-Box* und die CRIL-Netze als *A-Box* der Beschreibungslogik. Die statische Taxonomie des GA darf als Ersatz für die *T-Box* des Beweisers verwendet werden, da für eine *T-Box* nur gefordert wird, das sie terminologisches Wissen, also im einfachsten Fall, eine Sammlung von Konzeptdefinitionen enthält. Eine Taxonomie ist eine Sammlung von Konzeptdefinitionen, geht aber darüber hinaus, da sie zusätzlich eine Hierarchisierung, der durch sie beschriebenen Konzepte, bereitstellt. Für den Ansatz ist eine Hierarchisierung der Konzeptdefinitionen nicht erforderlich, aber auch nicht hinderlich, da diese zusätzlichen Informationen schlicht ignoriert werden. Schließlich darf ein vorliegendes CRIL-Netz als *A-Box* der Beschreibungslogik angesehen werden, da von einer *A-Box* gefordert wird, das sie Wissen über konkrete Individuen der Welt und ihre Relationen untereinander enthält. Dies ist genau das, was ein CRIL-Netz im Kontext des GA leistet. Somit ist die Substitution der Komponenten des Beweisers mit entsprechenden Teilen des GA verträglich.

Stellt man einen Ausdruck in der Beschreibungssprache, der aus Konstruktoren (Operatoren die Teilausdrücke verknüpfen), Konzepten und Rollen besteht, einen CRIL-Netz gegenüber, welches aus Knoten mit Attributen und Relationen zwischen diesen Knoten besteht, so kann folgende Zuordnung getroffen werden: Die Attribute (Eigenschaften) der Knoten des Netzes entsprechen dabei den atomaren Konzepten der Beschreibungslogik und die Relationen der Knoten des Netzes den Rollen der Beschreibungslogik. Diese strukturelle Übereinstimmung zwischen Beschreibungslogiken und CRIL-Netzen erlaubt es nun Ausdrücke in der Beschreibungssprache zu Formulieren, um Muster bzw. Zusammenhänge in beliebigen CRIL-Netzen zu beschreiben. Die Beschreibung manifestiert sich durch die Interpretation der

Konstruktoren, den Konzepten und Rollen eines Ausdrucks. Den Kern der Interpretation bilden dabei Suchoperationen, die alle Knoten aus einen gegebenen CRIL-Netz auswählen, welche die durch den Ausdruck beschriebenen Attribute bzw. Relationen besitzen. Konstruktoren der Beschreibungssprache können als Mengenoperationen definiert werden[15], da das Ergebnis der Interpretation eines Ausdrucks eine Menge von Knoten ist.

Um eine Anreicherung durch einen Ausdruck beschreiben zu können muss eine Zuordnung eines Ausdrucks, der die zu manipulierende Knotenmenge beschreibt, zu einen Konzept existieren, welches dem ausgewählten Knoten als Attribut hinzugefügt wird. Die Autoren nutzen hierfür das, in der Beschreibungssprache enthaltene, *T-Box* Axiom *implies*. Eine Anreicherung wird somit durch den Ausdruck *(implies A B)* beschrieben. Durch die Interpretation des Ausdrucks wird jeden Knoten der, durch A beschriebenen, Knotenmenge ein durch B bestimmtes Attribut hinzugefügt (siehe Abschnitt 4.7). Die Autoren verzichteten auf eine weitergehende Anreicherung des Netzes. Das Hinzufügen weiterer Attribute an schon bestehende Knoten verändert die Struktur des Netzes nicht. Prinzipiell sind aber Anreicherungen denkbar, die die Struktur eines Netzes erweitern, in dem weitere Knoten oder Relationen dem Netz hinzugefügt werden.

Um nun die regelhafte Anreicherung von Netzen zu implementieren führt der Ansatz drei Objekte bzw. Datenstrukturen ein. Dies sind: Regeln, Regelmengen und Usecases. Regeln werden, wie oben angedeutet, benötigt, um eine Anreicherung formulieren zu können, Regelmengen und Usecases um das Arbeiten mit Regeln zu vereinfachen und die Integration in den GA zu erleichtern. Folgend werden die einzelnen Objekte kurz beschrieben:

Eine Regel zur Anreicherung besteht, unter anderem, aus einem beschreibungslogischen Ausdruck, der der Form nach, wie oben beschrieben, eine Implikation ist (siehe Abschnitt 4.4 für eine genaue Beschreibung).

Regelmengen fassen eine beliebige Anzahl von Regeln zusammen und ermöglichen dadurch das Gruppieren von Regeln nach unterschiedlichen Gesichtspunkten. Die Anreicherung eine Netzes wird durch Regelmengen erleichtert, da nicht jede Regel gesondert spezifiziert werden muss. Auch das Formulieren von komplexen Aspekten mit vielen Einzelregeln wird durch die Existenz von Regelmengen handhabbarer.

Usecases definieren Kontexte der Anreicherung, denen jeweils eine Regelmenge zugeordnet werden kann. Ein Kontext kann dabei auf den Typ eines Netzes Bezug nehmen (Perzeption oder Instruktion) oder den Typ und einen konkreten Zeitpunkt in dem Verarbeitungsprozess der CRIL-Netze des GA benennen (Perzeption vor Abgleich mit Instruktion). Weitere beliebige

[15] eine genaue Aufstellung findet sich in Abschnitt 4.7

Kontexte sind denkbar[16]. Usecases erleichtern die Integration des Ansatzes in den GA, da sie Details vor dem GA verbergen und somit eine einfache Schnittstelle definieren, mit deren Hilfe der GA seine CRIL-Netze anreichern kann. Zudem erlauben sie eine automatisierte Anreicherung, ohne das der Benutzer des GA dazu eingreifen oder Entscheidungen treffen müsste, da die Zuordnung eines Kontextes zu einer Regelmenge stets gegeben ist.

Die oben beschriebenen Datenstrukturen des Ansatzes sollen über eine Benutzeroberfläche editiert werden können, die neben dem Erstellen, Editieren und Löschen auch das Laden und Speichern von Regeln, Regelmengen und Usecases ermöglicht. Zusätzlich werden die Regeln auf Korrektheit geprüft, bevor sie in die Daten des Ansatzes aufgenommen werden[17].

4 Vorarbeiten und Implementation

Im nun folgenden Teil soll die Implementierung unserer Erweiterung des Geometrischen Agenten (GA) beschrieben werden. Zu Beginn werden die eingebundenen Projekte und die relevanten Teile des GA-Codes vorgestellt, da diese den Ausgangspunkt unserer Arbeit darstellen und zumindest grundlegend bekannt sein sollten. Dann wird auf die Modellierung von Regeln und Regelmengen auf CRIL-Netzen und deren Serialisierung in einer XML-basierten Datenstruktur eingangen, woraufhin schließlich der wesentliche Teil des Projekts folgt, nämlich die Anwendung von Regeln auf ein CRIL-Netz. Am Schluss dieses Kapitels wird kurz die grafische Oberfläche (GUI) vorgestellt werden, über welche Regeln und Regelmengen erstellt, bearbeitet und verwaltet werden können.

4.1 Eingebundene Projekte

Extern zu diesem Projekt wurde das Ergebnis eines früheren Projekts dieses Fachbereichs („Reasoning Services") eingebunden. Es stellte für uns eine Repräsentation beschreibungslogischer Terme in Java und einen Parser für selbige zur Verfügung. Aus dem GA wurde lediglich das Package zur Repräsentation von CRIL-Netzen und der Taxonomie benötigt, da diese Arbeit gewissermaßen unabhängig vom GA ist und gänzlich auf der Ebene von CRIL arbeitet. Auch auf Änderungen an der vorhandenen Codebasis, die für dieses Projekt von Nöten waren, wird an dieser Stelle eingegangen. Waren die Änderungen im Fall des GA nur sehr gering, mussten Teile des eingebun-

[16] siehe Abschnitt 1.2.1 Motivation für weitere Erläuterungen

[17] siehe hierzu die Abschnitte 4.6 bis 4.8

denen Tableaubeweisers tiefer aufgebohrt werden, um die darin enthaltenen Klassen für diese Arbeit sinnvoll ableiten zu können.

4.1.1 Tableaubeweiser für Beschreibungslogiken

Einleitung Im Wintersemester 2006/2007 fand das Projekt „Reasoning Services: Tableaubeweiser für Beschreibungslogiken" statt, mit dem Ziel einen in Java geschriebenen Tableaubeweiser für Aussagenlogik auf Beschreibungslogikgen zu erweitern. Es galt einerseits die Repräsentation von Symbolen und Termen auf beschreibungslogische Ausdrücke und den Tableaubeweiser um entsprechende Expansionsregeln zu erweitern, aber auch das aussagenlogischen Tableauverfahren zu optimieren, um Schlüsse in Beschreibungslogiken effizienter zu machen [Sol06].

Operator	Formale Syntax	DLCS-Syntax
Top-Konzept	$\top$	(top)
Bottom-Konzept	$\bot$	(bottom)
Negation	$\neg C$	(not C)
Subsumption	$C \sqsubseteq A$	(implies C A)
Konjunktion	$C \sqcap \ldots \sqcap D$	(and C ... D)
Disjunktion	$C \sqcup \ldots \sqcup D$	(or C ... D)
Werterestriktion	$\forall r.C$	(all r C)
(Qualifizierte) Existenzrestriktion	$\exists r.C$ $\exists r$	(some r C) (some r)
Anzahlrestriktion	$(\geq n\ r)$ $(\leq n\ r)$ $(= n\ r)$	(at-least n r) (at-most n r) (exactly n r)
Qualifizierte Anzahlrestriktion	$(\geq n\ r.C)$ $(\leq n\ r.C)$ $(= n\ r.C)$	(at-least n r C) (at-most n r C) (exactly n r C)

Tabelle 2: Überblick über die relevanten DL-Operatoren und ihre Form in DLCS-Syntax. In der Tabelle stehen A für ein atomares Konzept, r für eine atomare Rolle, C, D für komplexe Konzeptausdrücke und n für eine beliebige natürliche Zahl.

Der Parser für Beschreibungslogiken akzeptiert die sogenannte *concrete syntax*, welche im Appendix des Description Logic Handbook [Baa03] vorgeschlagen wird. Sie ist ein Versuch formallogische Symbole zu vermeiden, um Ausdrücke über einer Beschreibungslogik ohne Verwendung von Sonderzeichen darstellen zu können. Es ist leicht einzusehen, dass dies eine erhebliche

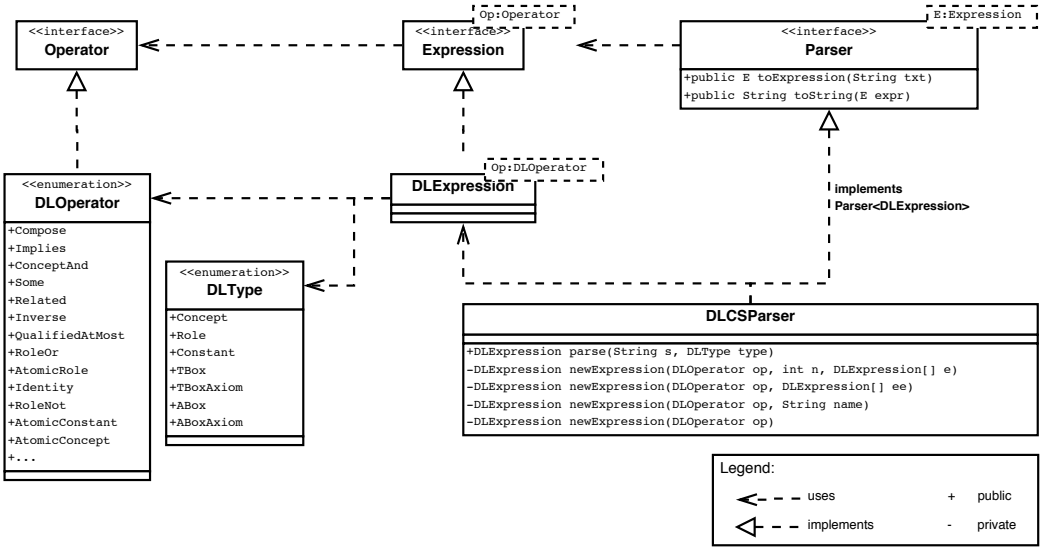

Abbildung 1: Ausschnitt aus dem Package expressions. Methoden und Eigenschaften wurden weitestgehend weggelassen, um die Übersicht zu verbessern.

Vereinfachung der Eingabe über eine Standardtastatur bedeutet. In der Tabelle 2 befindet sich eine Übersicht über die grundlegenden Beschreibunglogischen Konstruktoren und ihrer Entsprechungen in *concrete syntax*.

Implementierung Der Tableaubeweiser besteht aus sieben Packages und Subpackages, welche die Repräsentation von Ausdrücken in Aussagenlogik (AL) und Beschreibungslogiken (DL), zwei Tableaubeweiser für AL und DL und die dazugehörigen Expansionsregeln, Infrastruktur für Performanztests, und eine graphische Benutzerschnittstelle enthalten. Für die Arbeit der Autoren dieses Textes war lediglich die Repräsentation von DL-Termen und deren Parsing erforderlich, weshalb sich dieser Abschnitt ausschließlich mit dem Package expressions beschäftigt, welches beides beinhaltet. Für einen tiefergehenden Einblick in den Tableaubeweiser seien dem Leser der Projektbericht von Roland Illig [Ill08] und die Baccalaureatsarbeit von Arved Solth [Sol06] empfohlen.

Die Funktionalität des Packages expressions wird grundlegend durch drei Interfaces beschrieben: Operator und Expression⟨Operator⟩ für die Darstellung von Ausdrücken einer Logiksprache und Parser⟨Expression⟩ zur Überführung solcher Ausdrücke als Text in ihre Java-Repräsentation und umgekehrt. Die Abbildung 1 zeigt die drei Interfaces mit den implementierenden Klassen für

Listing 1: Das Interface Parser (ohne Kommentare):

```
public interface Parser<E extends Expression>
{
  public E toExpression(String txt) throws ParserException;
  public String toString(E expression);
}
```

Beschreibungslogiken aus dem Packages

An diesem Ausschnitt wird auch eine der wesentlichen Designentscheidungen des Projekts deutlich, nämlich sowohl beschreibungslogische Atome, als auch Konstanten, komplexe Ausdrücke und sogar ganze TBoxen, ABoxen und ihre Axiome durch die Klasse DLExpression zu modellieren. Dies soll der Minderung von Coderedundanz zugute kommen [Ill08], bedingt jedoch, dass nicht vom Objekttyp auf das modellierte Objekt geschlossen werden kann. Der Ausweg aus diesem Dilemma kommt in Form der zwei Aufzählungen DLType und DLOperator. Dem Konstruktor von DLExpression wird je ein Element der beiden Aufzählungen übergeben, um den modellierten beschreibungslogischen Ausdruck zu klassifizieren. Die Aufzählung DLType ist dabei für die grobe Einteilung in Konzepte, Rollen, Konstanten und TBox, ABox, bzw. deren Axiome zuständig. Die ersten beiden Fälle teilen beschreibungslogische Ausdrücke in zwei Klassen für Konzepte und Rollen, verbergen jedoch, ob es sich um Atome oder komplexe Ausdrücke handelt. Aufschluss darüber gibt der Wert der Aufzählung DLOperator. Er gibt entweder den (beschreibungslogischen) Konstruktor auf der obersten Ebene eines zusammengesetzten Ausdrucks an, oder steht mit den Werten AtomicConcept oder AtomicRole für atomare Konzepte und Rollen.

Neben Typ und Operator beschränken sich die entscheidenden Membervariablen auf einen ganzzahligen Wert für Anzahlrestriktionen, eine Zeichenkette für einen Namen und ein Array von Teilformeln, welches wiederum vom Typ DLExpression[] ist. Der Name wird ausschließlich für atomare Rollen und Konzepte genutzt und ist ansonsten null. Das Array von Teilformeln enthält entweder alle Teilausdrücke, die durch den Operator miteinander verknüpft werden, oder aber alle Axiome einer T- oder ABox. Alle Instanzen von DLExpression sind immutabel, weshalb alle Membervariablen als final deklariert sind und alle getter-Methoden lediglich sogenannte *shallow copies* der eigentlichen Referenzen zurück geben. Da alle Teilformeln wiederum vom Typ DLExpression sind, setzt sich die Eigenschaft der Unveränderlichkeit nach Instanziierung rekursiv in allen Teilformeln fort.

Das Interface Parser⟨Expression⟩ (s. Listing 1) ist allerdings für den Beschreibungslogikparser unzureichend und wird anscheinend nur aus ,,histo-

rischen" Gründen auch von der Klasse DLCSParser implementiert. Es stellt zwei Methoden bereit, die den Transfer zwischen Text und Java-Objekten in beide Richtungen abbilden sollen. Der Grund für die Unzulänglichkeit des Interfaces liegt in der akzeptierten Syntax begründet. Die *concrete syntax* ist nicht spezifisch genug, um einen Term in jedem Fall eindeutig klassifizieren zu können. Daher wird der Typ des zu parsenden Objekts als Element der Enumeration DLType mit übergeben und die toExpression(String txt)–Methode des Interfaces Parser⟨Expression⟩ reicht folglich ohne weitere Argumente nicht aus.

Änderungen im Rahmen dieses Projekts Aus dem Tableaubeweiser war für diese Arbeit wie oben bereits erwähnt lediglich die Repräsentation beschreibungslogischer Formeln in Java interessant, inklusive deren Parsen aus der *concrete syntax*. Die dafür verantwortlichen Klassen sind DLExpression und DLCSParser, mit ihren Abhängigkeiten DLOperator und DLType.

Gegenstand dieser Arbeit waren Regeln über CRIL-Netzen und somit über der Taxonomie des Geometrischen Agenten (GA). In Beschreibungslogik formuliert sollten sie demnach lediglich Atome mit Entsprechungen in der Taxonomie enthalten. Somit musste also zunächst die Klasse DLExpression abgeleitet werden, sodass bei Erstellung eines atomaren Konzepts oder einer atomaren Rolle deren Namen mit der Taxonomie abgeglichen wird. Danach sollte der DLCSParser abgeleitet werden, um nicht mehr Objekte des Typs DLExpression, sondern unseres abgeleiteten Typs zurück zu geben. Für die Klasse DLExpression war dies unproblematisch, jedoch musste der Code des DLCSParsers leicht umstrukturiert werden, um ein Ableiten zu ermöglichen.

Ein schöner Ansatz wäre generische Programmierung gewesen, so dass der DLCSParser je nach übergebenem Typ entweder Instanzen von DLExpression oder CrilDLExpression parst. Im Verlauf des Parsens würden jedoch neue Objekte des generischen Typs neu instanziiert, was nicht möglich ist. Der Grund hierfür ist, dass es keine Zusicherungen über Vorhandensein oder Signatur von Konstruktoren des generischen Typs gibt. Um dies zu umgehen wurde eine abstrakte Klasse eingeführt, die fast den gesamten Code des alten Parsers enthält und lediglich die Erstellung neuer Instanzen von DLExpression in abstrakte Methoden kapselt (Abbildung 2). Die neue Klasse AbstractDLCSParser arbeitet auf einem generischen Typ, der CrilDLExpression ableiten muss und verschiebt das Problem der Instanziierung also in die sie ableitenden Klassen. Es handelt sich hierbei um eine reine Umstrukturierung, da die Instanziierung im Code des DLCSParsers bereits in einzelne Methoden gekapselt war.

Nach dem Refactoring liefen alle JUnit-Tests des Packages expressions

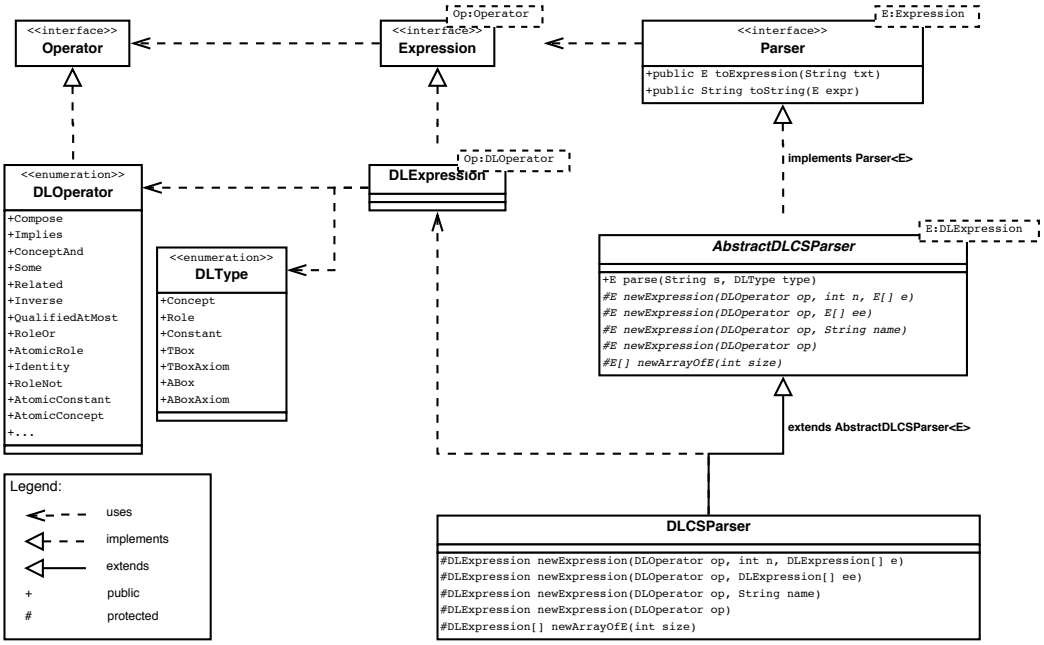

Abbildung 2: Ausschnitt aus dem Package expressions nach dem Refactoring.

fehlerlos durch, jedoch ist anzumerken, dass ihr Quelltext an vielen Stellen auf Unvollständigkeit hinweist.

4.1.2 Geometrischer Agent

Dieses Projekt greift kaum in den Code des GA ein, sondern setzt im Grunde eine Erweiterung oben drauf. Durch diesen ,,Add-on-Charakter" der Arbeit gibt es schlanke, gut definierte Schnittstellen zum GA und umgekehrt gibt es nur ein paar wenige Stellen, an denen die Erweiterung aus dem GA heraus aufgerufen wird. Der Hauptgegenstand der Schnittstelle ist das *Interface CrilInterface*, zusammen mit den Klassen für CRIL–Netze und die Taxonomie des GA. Als essenzielle Grundlage dieser Arbeit, soll hier kurz auf die Grundzüge dieses Interfaces und der Taxonomie eingegangen werden. Außerdem wurden im Rahmen dieser Arbeit als Nebenprodukt ein paar Änderungen am Code der Taxonomie vorgenommen, welche ebenfalls skizziert werden.

Taxonomie Ziel der Taxonomie ist es, das gesamte semantische Wissen des GA über die Welt zu repräsentieren. Sie liegt in Form einer Prolog-Wissensbasis vor, da die Textinstruktion des GA in Prolog verarbeitet wird,

kann aber über ein Skript in eine Javaklasse exportiert werden. Sie ist unveränderlich und besteht aus einstelligen Prädikaten für Entitäten und Konzepte des Weltausschnitts und mehrstelligen Prädikaten für Beziehungen unter den Entitäten. Für die Navigation wichtige Prädikate sind z.B. Baum, Gebäude, Weg, Wegpunkt und mögliche Beziehungen, wie vor, hinter, rechts von, zwischen.

Implementiert wird die Taxonomie des GA im Package geoAgent.cril durch die Klassen Taxonomies und GATaxonomies. Erstere enthält zwei Listen mit Objekten des Typs TaxonomyElement, welche einstelligen und mehrstelligen Prädikaten entsprechen und erstellt grundlegende Elemente, wie z.B. die im letzten Paragraphen genannten. Die Klasse GATaxonomies, erbt von Taxonomies und fügt weitere Prädikate aus der Prolog-Wissensbasis in die Listen ein. Sowohl Taxonomies als auch GATaxonomies sind nach dem Singleton-Pattern aufgebaut, so dass die Taxonomie jederzeit abrufbar ist, jedoch nur ein einziges Mal erstellt wird.

Dieser Aufbau als Singleton ist das oben angesprochene Nebenprodukt dieser Arbeit. Auch wenn der Aufbau nicht duch diese Arbeit bedingt wird, stieß der vorherige Aufbau dem Autor dieser Zeilen (zu) unangenehm auf. Der vorherige Aufbau der Klassen Taxonomies und GATaxonomies, ließ beide Klassen ihre Taxonomie jeweils im Konstruktor erstellen, wobei GATaxonomies bereits von Taxonomies abgeleitet wurde. Die Listen der Taxonomieelemente waren jedoch statisch und alle Elemente werden beim Einfügen auf Vorhandensein geprüft. Der Grund hierfür war, dass der Quellcode der Klasse GATaxonomies aus Prolog heraus generiert wird und daher so wenig Code wie möglich enthalten sollte. Daraus folgte jedoch, dass vor jeder Nutzung der Taxonomie im GA zuerst eine Instanz von GATaxonomies erstellt wurde, wodurch beim ersten Mal die Taxonomie erstellt wurde und bei allen weiteren Instanziierungen nochmals alle Elemente durchlaufen wurden ohne ein einziges zu erstellen. Es muss allerdings angemerkt werden, dass der Umgang mit der Taxonomie im GA bislang so sauber war, dass es nie zu einer mehrfachen Instanziierung kam und eine solche erst durch dieses Projekt notwendig wurde.

Im neuen Aufbau stellt GATaxonomies nun lediglich eine Initialisierungsmethode bereit, die im Konstruktor von Taxonomies aufgerufen wird. Dies umgeht nicht ganz das Problem eines mehrfachen Aufrufs, da die Initialisierung theoretisch mehrfach aufgerufen werden kann, folgt aber dem Ziel der Codeminimierung in GATaxonomies. Zusammen mit der Verwendung des Singleton-Patterns ist die intendierte Optimierung jedoch dahingehend sichergestellt, dass die Erstellung der Taxonomie über den privaten Konstruktor von Taxonomies nur ein einziges Mal vorgenommen wird. Mittels einer getter-Methode ist die statische Instanz der Taxonomie dann von verschie-

denen Stellen aus direkt erreichbar.

Wichtig für den Rahmen dieses Projekts ist, dass in der Navigation des GA nur ein kleines Fragment der Taxonomie Verwendung findet. Dieses Fragment besteht mit einer Ausnahme ausschließlich aus ein- und zweistelligen Prädikaten, die die wichtigsten Elemente von Skizzen und der abstrahierten simulierten Welt des GA beschreiben: Landmarken, wie Häuser und Bäume, Wegsegmente und mögliche Pfade auf Wegsegmenten und dazu verschiedene räumliche Relationen wie ,,rechts von", ,,links von", ,,vor", ,,neben", etc. Grund hierfür ist die derzeitige Ausbaustufe des GA und der simulierten Welt, in der er sich bewegt. Viele Elemente der Taxonomie sind in der GA-Welt schlicht nicht modelliert und es werden nur grundlegende, meist spatiale Relationen zwischen Objekten vom GA erkannt. Die Ausnahme von dieser Einschränkung ist die dreistellige Relation ,,zwischen", welche die Position eines Objekts zwischen zwei anderen Objekten ausdrückt. Sie ist jedoch die einzige genutzte Relation mit einer Stelligkeit > 2, wodurch die Nähe zu Beschreibungslogiken deutlich wird.

CRIL-Netze im GA CRIL-Netze oder auch CRIL-Graphen werden im GA zur Repräsentation von jeglicher geometrischen Informationen eingesezt. Sie sind in der Lage geometrische Zusammenhänge zwischen Objekten darzustellen. Beispielsweise kann die Aussage, das der Kirchturm links von der Tankstelle und zwischen dem Aldi-Markt und dem großen Parkplatz gelegen ist, leicht über ein CRIL-Netz repräsentiert werden. Da sowohl die Wegbeschreibung als auch die vom GA wahrgenommene Umwelt, miteinander in Beziehung gesetzt bzw. Abgeglichen werden müssen ist es naheliegend, beides mithilfe von CRIL-Netzen zu repräsentieren. Diese können miteinander auf Ähnlichkeit verglichen werden, welches die Grundlage für die Navigation des GA in seiner perzeptierten Umwelt darstellt. Bis auf die Sprachverarbeitung und die statische Taxonomie, auf die zum Vergleichen von Objekten zurückgegriffen werden muss, ist alles Wissen des GA um seine Umwelt und seine Instruktionen durch CRIL-Netze repräsentiert.

4.2 Die Implementierung im Überblick

Für die Aufteilung der verschiedenen Aspekte unseres Ansatzes haben wir uns an dem MVC (Model View Controller) Paradigma orientiert. Das MVC Paradigma unterteilt eine Applikation in drei Bereiche die lose gekoppelt sind und so im Idealfall austauschbar sind. In unserer Implementation sind die einzelnen Komponenten nicht lose gekoppelt, können aber leicht abstrahiert werden falls dies nötig sein sollte. Wie der Name MVC andeutet, wird eine Applikation in drei Teile unterteilt: das (Daten-)Modell, welches die

Daten und alle zu diesem Aspekt zugehörigen Funktionalitäten der Applikation enthält, den Controller, in dem die Programm- und Ablauflogik spezifiziert werden und ein View, welches eine Schnittstelle zum Benutzer anbietet. Somit besteht das Modell unseres Ansatzes aus den Regeln, Regelmengen, Usecases, Funktionen zum Laden und Speichern von Daten und einen Parser, der String-Repräsentationen von beschreibungslogischen Ausdrücken in Objekte wandeln kann. Der Controller enthält Funktionen, um die Regeln auszuführen und zwischen dem Datenmodell und dem Benutzer bzw. GA zu vermitteln. Und schliesslich fungiert das von uns entwickelte graphische Benutzerinterface, um die Regeldaten zu Verwalten, in diesem Kontext als View.

Die Implementierung ist in dem Package geoAgent.cril.augmenting enthalten. Sie besteht aus den Klassen CrilAugmentor, CrilDLExpression, CrilRule, CrilRuleExecutor, CrilRuleManager, CrilRuleParser, CrilRuleSet, CrilRulesetUsecase und einigen Exception- und Testklassen. Die Klasse CrilAugmentor fasst die gesamte Funktionalität zusammen und bietet sozusagen den ,,Augmenting Service" an. CrilRuleManager enthält Funktionalitäten zum Laden und Speichern der Regeldaten und anderer Einstellungen, verwaltet also das Datenmodell unseres Projekts. CrilRuleExecutor ist für das Ausführen der Regeln zuständig, CrilRule, CrilRuleSet und CrilRulesetUsecase repräsentieren die Regeln, Regelmengen und Usecases. Die Klasse CrilDLExpression enthält einen Beschreibungslogischen Ausdruck und der CrilRuleParser ist für das Parsen der String-Repräsentation beschreibungslogischer Formeln zuständig. Falls das Parsen erfolgreich ist wird eine Instanz der Klasse CrilDLExpression instanziert, die der beschreibungslogischen Formel entspricht.

4.3 Beschreibungslogik über CRIL

Wie bereits geschildert, wird zur Repräsentation beschreibungslogischer Ausdrücke auf Klassen aus dem Tableaubeweiser für Beschreibungslogiken zurückgegriffen. Es sollte allerdings lediglich die Syntax von Beschreibungslogiken genutzt werden, um Regeln über der Taxonomie des GA formulieren zu können. Diese Regeln sollen als (beschreibungslogische) Atome lediglich Elemente der Taxonomie enthalten dürfen, um sinnvoll auf CRIL-Netze über der Taxonomie angewendet werden zu können. Dazu wurden die Klasse DLExpression in unserem Package durch CrilDLExpression abgeleitet, um bei Erstellung eines atomaren Konzepts oder einer atomaren Rolle einzugreifen. An dieser Stelle wird der Name eines Atoms mit der Taxonomie abgeglichen, in der Art, dass atomare Konzepte als Prädikate und atomare Rollen als zweistellige Relation in der Taxonomie bekannt sein müssen. Wird der Name nicht in der Taxonomie gefunden schmeißt der Konstruktor von CrilDLExpression

eine entsprechende Exception.

Nach den oben beschriebenen Änderungen am Code des Parsers, war es möglich die abstrakte Klasse AbstractDLCSParser abzuleiten und die abstrakten Methoden zur Erstellung von geparsten Objekten neu zu implementieren. Somit hält dieses Projekt seinen eigenen Parser, um Objekte vom Typ CrilDLExpression zu parsen, ohne Code des Tableaubeweisers zu duplizieren und ist sauber vom Tableaubeweiser getrennt. Die Implementation gleicht der von DLCSParser – es wurden lediglich Instanziierungen von DLExpression durch Instanziierungen von CrilDLExpression ersetzt.

An dieser Stelle soll darauf hingewiesen sein, dass von Seiten der Repräsentation keine Grenzen für die beschreibungslogischen Ausdrücke über der Taxonomie gesetzt werden (außer durch die vorhandenen Konstruktoren in der Aufzählung DLOperator). Damit ist die Repräsentation und der Parser gewappnet für einen weiteren Ausbau, der über die zum Zeitpunkt dieser Arbeit unterstützten Regeln, bzw. Ausdrücke, hinausgeht.

4.4 Aufbau einer CRIL-Regel

Bei der Implementation von Regeln über CRIL muss grundlegend zwischen zwei Verwendungen des Wortes „Regel", bzw. der englischen Übersetzung „rule", unterschieden werden. Es steht für formaltextuelle beschreibungslogische Ausdrücke und deren Repräsentation in Java-Objekten einerseits und für die eigentlichen Regel, einem Objekt mit Namen, Beschreibung, einem beschreibungslogischen Ausdruck und weiteren Attributen, andererseits. Die Verwendung ist trotz einiger Anstrengungen seitens der Autoren weder im Code noch in diesem Dokument immer klar getrennt, sollte jedoch aus dem Kontext hervorgehen.

Als Repräsentation einer Regel über CRIL-Netzen dient die Klasse CrilRule. Sie enthält vier grundlegende Attribute: einen Namen und eine Beschreibung als Strings und den eigentlichen beschreibungslogischen Ausdruck in formaltextueller Darstellung als String und als CrilDLExpression-Objekt. Die Doppelung des Regelinhalts war eine Designentscheidung, um einerseits das Speichern/Laden von unfertigen, korrupten oder aus anderen Gründen invaliden Regeln zu erlauben und andererseits jede korrekte Regel nur ein einziges Mal parsen zu müssen. Die entsprechenden Setter- und Getter-Methoden stellen dabei sicher, dass die beiden Darstellungsformen keine widersprüchlichen Informationen enthalten.

Als fünfte Eigenschaft enthält die Klasse eine statische Referenz auf ein Objekt vom Typ CrilRuleParser. Die Verwendung als statische Eigenschaft ermöglicht es ohne großen Verwaltungsaufwand eine einzige Parserinstanz für alle CrilRule-Instanzen zu verwenden. Dafür wird lediglich die Threadsi-

cherheit geopfert, welche auf Grund der derzeitigen Architektur des GA kein Erfordernis darstellt.

Zu den wichtigen Methoden der Klasse gehören isValid() und parse(). Erstere prüft, ob die Instanz eine gültige Regel repräsentiert und gibt lediglich einen booleschen Wert zurück, während parse() versucht, die textuelle Darstellung des beschreibungslogischen Ausdrucks zu parsen und reicht Exceptions aus dem Parser weiter, um sie z.B. in Fehlermeldungen zu verwenden. Ein erfolgreicher Durchlauf von parse() speichert das gewonnene CrilDLExpression-Objekt als Referenz, solange die formaltextuelle Darstellung nicht neu gesetzt wird.

Diese Fehlertoleranz soll dem Benutzer ermöglichen, unfertige, inkorrekte Regeln zu speichern und korrupte Daten zu laden. Eine striktere Verwendung von Regeln hätte den Programmcode an einigen Stellen erheblich vereinfacht, würde aber zu Lasten der Benutzerfreudlichkeit gehen.

Die Klasse verfügt über einen umfassenden JUnit-Test, der zum Zeitpunkt des Verfassens dieses Dokuments fehlerfrei durchläuft.

4.5 Regelmengen

Gültige CRIL-Regeln können in Mengen zusammengefasst werden, um alle Regeln dieser Menge in einem Lauf auszuführen. In unserer Implementation werden die Regeln einer Regelmenge nur einmal auf das gesammte CRIL-Netz angewandt. Dies garantiert eine Ausführung der Regelmengen in endlicher Zeit. Dabei sollen zusätzlich folgende Beschränkungen für die Ausführung einer Regelmenge gelten:

- Das Ergebnis der Ausführung der Regelmenge ist determiniert.
- Das Ergebnis der Ausführung der Regelmenge ist vollständig, d.h. auch bei mehrmaliger Anwendung der Regelmenge auf den gleichen CRIL-Graphen werden keine neuen Informationen nach dem ersten Durchlauf hinzugefügt.

Um dies zu gewährleisten ist es bei unserer Implementation notwendig, zyklische Regelmengen zu unterbinden und eine bestimmte Ausführungsreihenfolge (AR) zu finden. Eine zyklische Regelmenge ist eine Menge, in der eine Regel als Bedingung das Ergebnis einer anderen Regel benötigen könnte, welche (eventuell über eine Kette von anderen Regeln) wiederum das Ergebnis eben dieser Regel benötigen könnte, so dass bei einer Anwendung der Regelmenge die Vollständigkeit der Ausführung nicht mehr garantiert werden kann. Ist die Regelmenge nicht zyklisch, so gibt es mindestens eine AR, deren Ergebnis determiniert und vollständig ist.

Um nun zu erkennen, ob eine Regelmenge zyklisch ist, werden alle Regeln dahingehend untersucht, ob sie sich selber eventuell benötigen. Da eine Regelmenge auch unabhängige Regelgruppen enthalten kann, welche nur Regeln innerhalb ihrer Gruppe benötigen oder von Regeln ihrer Gruppe benötigt werden, müssen sämtliche Regeln der Menge getestet werden. Demnach sind n Untersuchungen notwendig. Um festzustellen, ob eine Regel A eine Regel B benötigt, wird geprüft, ob Teile des Antezedens der Regel A das Sukzedens einer anderen Regel B beinhalten. Weiterhin benötigt die Regel A auch alle Regeln, welche die Regel B benötigt.

Pro Untersuchung wird wiederum für jede Regel die Menge der benutzten Regeln rekursiv erstellt, womit jede Teiluntersuchung schlimmstenfalls $n!$ Durchläufe hat. Die Gesammtuntersuchung würde demnach schlimmstenfalls $n * n!$ Schritte benötigen. Weiterhin wird die Untersuchung erfolgreich abgebrochen, wenn schon ein Zyklus entdeckt wurde und der Benutzer bekommt eine Meldung, dass die Regelmenge Zyklen enthält. Diese Regelmenge kann dann nicht ausgeführt werden und die schlimmeren Untersuchungen werden frühzeitig abgebrochen.

Die Zyklenerkennung könnte stark verbessert werden, wenn Knoten, welche schon in einer Rekursion geprüft worden sind, nicht nochmal geprüft würden. Dann wäre die Anzahl der Schritte der Teiluntersuchung nur noch höchstens n, da jeder Knoten nur genau einmal geprüft werden müsste, was die Gesammtuntersuchung auf n^2 Schritte bringt.

Eine binäre Matrix mit transitiver Hülle sollte in der Lage sein, schnell und effizient die Abhängigkeiten darzustellen und durch Prüfen der Hauptdiogonalen festzustellen, ob es Zyklen gibt. In diesem Falle ist die Komplexität nur noch $n^2 + O_{TH} + n$ (n^2 für die Erstellung der Matrix, O_{TH} sei die Komplexität der Bildung der Transitiven Hülle und n für die Untersuchung der Hauptdiagonalen). Nutzt man zu Erstellung der transitiven Hülle den Algorithmus von Tarjan zur Bestimmung starker Zusammenhangskomponenten[18], welcher die Komplexität $2A_K + n$ hat, wobei A_K die Anzahl der Kanten im Graph ist, also höchstens n^2, dann ist die Komplexität der Erfassung der transitiven Hülle $2n^2 + n$ und weiterhin die Komplexität der Findung von Zyklen $n^2 + 2n^2 + n + n = 3n^2 + 2n$.

Wissen wir, dass die auszuführende Regelmenge keine Zyklen enthält, müssen wir noch eine gute AR finden, die garantiert, dass wir ein vollständiges Ergebnis bekommen. Ausführbare Regeln nennen wir Regeln, welche aktuell in die AR genommen werden dürfen, also welche keine andere Regel aus der Regelmenge eventuell benötigen oder bei denen alle eventuell benötigten Regeln sich schon in der AR befinden.

[18] nachzulesen in http://www.cs.hut.fi/~enu/thesis.html

Da wir eine azyklische Regelmenge haben und die Regelmenge endlich ist, gibt es solange mindestens eine Regel, die ausführbar und noch nicht in der AR ist, wie es Regeln gibt, die noch nicht in der AR sind. Solche Regeln werden zufällig in die AR aufgenommen. Regeln, die eine Regel benötigen, welche noch nicht in der AR enthalten sind, können auch nicht in die AR genommen werden. Jedoch werden zwangsläufig eben diese Regeln irgendwann in die AR aufgenommen werden können, denn wenn dies nicht möglich wäre, weil mindestens eine benötigte Regel nie reingenommen werden kann, dann nur dann, wenn eben diese Regel sich irgendwann selber benötigt, also wenn die Regelmenge zyklisch war. Dies haben wir aber im Vorschritt ausgeschlossen.

Wir iterieren solange über die Menge der noch nicht in die AR aufgenommenen Regeln bis diese leer ist. Dies sind im schlimmsten Falle $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Versuche die AR zu vervollständigen.

Da wir schon vorher geprüft haben, dass die Regeln auf unserer Taxonomie arbeiten und zulässige Operatoren verwenden, sind auch die Ausführungen von einzelnen Regeln in unserer Regelmenge zulässig und somit ebenfalls die Ausführung einer ganzen Regelmenge in Form einer AR.

Daher macht es Sinn ,,korrekten" Regelmengen eine automatische Ausführung zu erlauben. Dabei wird an bestimmten Stellen im GA die Ausführung einer speziell gekennzeichneten Regelmenge vorgenommen. Diese Kennzeichnungen sind im **CrilRulesetUsecase** vorgeschrieben und hier auch erweiterbar. Jedoch müssen die entsprechenden Stellen im GA selber gefunden und von dort Aufrufe mit entsprechenden Parametern gestartet werden.

Bestimmt man nun eine ausführbare Regelmenge zum automatischem Ausführen für eine bestimmte Stelle, so wird diese Regelmenge immer an dieser Stelle ausgeführt und kann somit als ,,Defaultset" für diese Stelle angesehen werden. Nur nichtzyklische Regelmengen können als Defaultset ausgewählt werden und immer nur höchstens eines pro Stelle. Verändern sich diese Sets und werden dadurch zyklisch verlieren sie wieder ihren Default-Status.

4.6 Serialisierung der Regeldaten

Die Funktionalität der Serialisierung der Regeldaten wird von der Klasse CrilRuleManagers bereitgestellt. Diese Klasse ist für das Verwalten, Laden und Speichern der Regeln und all ihrer zugehörigen Daten zuständig. Alle Daten des CrilRuleManagers werden in einen XML-Format serialisiert. Dies umfasst die einzelnen Regeln, die Regelmengen und die so genannten CrilRulesetUsecases. Die Zuordnungen von Regeln zu Regelmengen und Regelmengen zu Usecases werden über inkrementierte Indexwerte realisiert. Diese werden

aufsteigend vergeben und sind nur in der serialisierten Form eindeutig. Das bedeutet, dass nach einer Deserialisierung und erneuten Serialisierung, die Indexe nicht die selben sein müssen, die logischen Zuordnungen aber erhalten bleiben.

Die vorliegende Implementation weicht von diesen Muster ab, da dort fälschlicherweise angenommen wurde, dass über das Bilden von Hashwerten eindeutige Zuordnungen erstellt werden könnten, was bei kleiner Anzahl von Regeln und Regelmengen durchaus berechtigt ist, diese Vorgehensweise bei großer Anzahl von Regeln aber unweigerlich zu Kollisionen führt. Eine Prüfung auf Eindeutigkeit bzw. auf Kollisionsfreiheit findet in der vorliegenden Implementation nicht statt, was eingangs erwähnte Kollisionen nicht ausschließt. Die nun von den Autoren favorisierte inkrementelle Vergabe von Indexwerten ist aber trivial zu implementieren.

Nachfolgend wird eine beispielhafte Beschreibung das XML-Formats zur Serialisierung der Regeldaten gegeben: Eine *Regel* wird über vier Werte serialisiert: Der Name der Regel, eine optionale Anmerkung, einen eindeutigen Integer-Wert, um die Regel einer Regelmenge zuordnen zu können, sowie eine Zeichenkette, die die formaltextuelle Repräsentation des beschreibungslogischen Ausdrucks in concrete Syntax enthält. Analog dazu, wird eine *Regelmenge* durch einen Namen, einen Kommentar, einen Integer-Wert, der die Regelmenge eindeutig repräsentiert und einer beliebigen Anzahl von Integer-Werten, die die zu dieser Menge gehörenden Regeln identifizieren. Schließlich wird ein *Usecase* über zwei Werte serialisiert: Einen Integer-Wert, der den jeweiligen Usecase-Typ repräsentiert und einen weiteren Integer-Wert, der auf eine bestehende Regelmenge verweist. Zum Zeitpunkt der Serialisierung wird davon ausgegangen, dass die Datenbasis in einen korrekten Zustand und damit jeder Usecase-Typ maximal einmal vorhanden ist.

Listing 2: beispielhafte Serialisierung der Regeldaten in einen Xml-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <rules>
        <rule>
          <name>Kreuzung</name>
          <annotation>ein Kommentar</annotation>
          <id>17</id>
          <rulestr>
                (implies (at-least 3 EPT TRACK) kreuzung)
          </rulestr>
        </rule>
        ...
  </rules>
  <rulesets>
        <ruleset>
```

```
            <name>Eine Regelmenge mit zwei Regeln</name>
            <annotation>und ein Kommentar</annotation>
            <id>158</id>
            <rule_id>17</rule_id>
            <rule_id>95</rule_id>
          </ruleset>
  </rulesets>
  <usecases>
          <case>
            <case_id>1</case_id>
            <ruleset_id>158</ruleset_id>
          </case>
  </usecases>
</data>
```

4.7 Ausführung von Regeldaten

Unter der Ausführung, bzw. der Anwendung, einer Regel verstehen die Autoren die Manipulation eines CRIL-Netzes durch die in einer Regel implizit beschriebenen Such- und Mengenoperationen. Der Prozess der Ausführung lässt sich grob in zwei Teile gliedern: Erstens das Parsen, also das Umwandeln der textuellen Repräsentation des beschreibungslogischen Ausdrucks, in eine CrilDLExpression und zweitens die Verarbeitung der Regel, in der die Regel abgearbeitet und das vorliegende CRIL-Netz entsprechend manipuliert wird. Der beschreibungslogische Ausdruck dient somit als Gerüst für die Such- und Mengenoperationen auf den Knoten eines gegebenen CRIL-Netzes. Das Ergebnis dieses Prozesses, eine Menge von Knoten aus dem vorliegenden CRIL-Netz, wird dann, mit dem in der Regel implizierten und in der Taxonomie das GA vorhandenen atomaren Konzept, angereichert. Dieser Prozess wird bei einer Menge von Regeln für jede Regel einzeln wiederholt, bis alle Regeln abgearbeitet worden sind.

Das Parsen Da sich die Autoren dieses Textes entschieden haben einer Regel (siehe Abschnitt 3.4 Aufbau einer CRIL-Regel) zwei Repräsentationsarten zu geben, eine formaltextuelle als String und eine weitere, in Form von Instanzen von CrilDLExpression, muss die String-Repräsentation in korrespondierende CrilDLExpression Instanzen umgewandelt werden. Während der Umwandlung wird die Korrektheit der Regel geprüft wird. Die formaltextuelle Repräsentation einer Regel ist Korrekt, wenn sie den folgenden Bedingungen gehorcht:

- die Struktur der Regel entspricht (implies expr AtomicConcept), wobei expr ein beliebig komplexer Ausdruck sein darf

- alle in der Regel vorkommenden atomaren Konzepte sind in der statischen Taxonomie des GA enthalten
- alle im Dialekt der Regel enthaltenen Operatoren werden vom CrilRuleExecutor unterstützt

Nachdem eine Regel erfolgreich geparst wurde, liegt sie als Baumstruktur von Instanzen des Typs CrilDLExpression vor. Das Parsen geschieht nur einmalig beim Deserialisieren der Regeldaten. Wird eine bestehende Regel verändert, d.h. Ihre formaltextuelle Repräsentation, wird die Regel erneut geparst und, nach erfolgreicher Überprüfung der Korrektheit, in eine CRILDLExpression umgewandelt.

Die Ausführung / Anwendung Die oben schon erwähnte Abbildung von beschreibungslogischen Ausdrücken auf Such- und Mengenoperationen auf Knoten in CRIL-Netzen, ergibt sich implizit durch das rekursive Abarbeiten der jeweiligen Teil-Ausdrucke einer Regel. Jeder in einer Teilregel vorkommende Operator steht für eine bestimmte Such- bzw. Mengenoperation (vergleiche hierzu Tabelle 3 im selben Abschnitt). Wobei sich die Wirkung eines Operators in die zwei eingangs erwähnten Klassen (Such- und Mengenoperationen) gliedern lässt:

Suchoperationen wählen aus einer gegebenen Knotenmenge, diejenigen Knoten aus, welche die durch den DL-Ausdruck spezifizierten Konzepte bzw. Rollen besitzen. Dabei entsprechen beschreibungslogische Konzepte den Eigenschaften (Properties) und beschreibungslogische Rollen den Relationen, die ein Knoten eines CRIL-Netzes besitzen kann. Die Suchoperationen auf Knoten gliedern sich in zwei Typen, den Suchen nach Knoten die ein gegebenes Konzept besitzen (Konzeptsuche) und die Suche nach Knoten die eine gewisse Anzahl von Rollen (Relationen zu anderen Knoten im CRIL-Netz) besitzen (Rollensuche). Die Konzeptsuche wird über den beschreibungslogischen Ausdruck AtomicConcept realisiert. Wird ein atomares Konzept in einen Teilausdruck gefunden, werden alle Knoten aus den CRIL-Graphen ausgewählt, die das gegebene Konzept erfüllen (oder genauer als Eigenschaft besitzen). Die Rollensuche manifestiert sich in den Operatoren AtLeast, AtMost, Exactly und ihren qualifizierenden Entsprechungen, sowie Some, LimetedSome und All. Beispielsweise werden bei den Operatoren AtLeast, AtMost und Exactly alle Knoten ausgewählt, die jeweils mindestens, höchstens, oder exakt so viele Rollen des gegebenen Typs besitzen. Ein qualifizierender Operator erwartetet zusätzlich, dass die über die Rollen verbundenen Knoten ebenfalls ein gegebenes qualifizierendes Konzept erfüllen müssen.

Operator	DLCS Ausdruck	Suchoperation
Top-Konzept	(top)	alle Knoten
Bottom-Konzept	(bottom)	keine Knoten - leere Menge
Atomares-Konzept	(D)	alle Knoten die Konzept D besitzen
Negation	(not C)	Komplement der durch C bestimmten Knotenmenge.
Subsumption	(implies C A)	jedem Knoten der durch C bestimmten Knotenmenge wird A hinzugefügt.
Konjunktion	(and C_1 $C_2 \ldots C_i$)	Schnittmenge aller durch C_i bestimmten Knotenmengen
Disjunktion	(or C_1 $C_2 \ldots C_i$)	Vereinigung aller durch C_i bestimmten Knotenmengen
Werterestriktion	(all r C)	alle Knoten die Rolle r besitzen
Anzahlrestriktion	(at-least n r) (at-most n r) (exactly n r)	alle Knoten die mindestens, höchstens oder exakt, die Anzahl der Rolle r besitzen
Qualifizierte Anzahlrestriktion	(at-least n r C) (at-most n r C) (exactly n r C)	Wie Anzahlrestriktion, wobei die durch r bestimmten Knoten zusätzlich das Konzept C besitzen müssen
(Qualifizierte) Existenzrestriktion	(some r C) (some r)	alle Knoten die mindestens eine durch r spezifizierte Rolle besitzen und (wenn qualifiziert) zusätzlich der durch r verbundene Knoten das Konzept C besitzen muss

Tabelle 3: Operatoren und Konzepte und ihre jeweiligen Auswirkungen auf Knoten im CRIL-Netzen. Der Ausdruck C ist entweder als atomares Konzept oder als beliebig komplexer beschreibungslogischer Ausdruck zu lesen.

Mengenoperationen auf Knoten des CRIL-Netzes werden durch die beschreibungslogischen Operatoren ConceptAnd = Mengendurchschnitt, ConceptOr = Mengenvereinigung und Not = Mengenkomplement eingeführt. Die Mengenoperationen dienen dem Verknüpfen der von Teilausdrücken ausgewählten Knotenmengen und entsprechen dabei in ihrer Semantik den aus

Operator / Ausdruck	Funktionsklasse
at-least, at-most, exactly, (und ihre qualifizierenden Entsprechungen), all, some, not	$\in O(n^2)$
atomic-concept, and*, or*	$\in O(n)$
top, bottom	$\in O(1)$

Tabelle 4: Operatoren und Konzepte und ihr jeweiliges Laufzeitverhalten mit $n :=$ Anzahl der Knoten eines gegeben CRIL-Netzes

der Logik bekannten Operatoren. Das Mengenkomplement wird gegenüber der Gesamtheit der Knoten eines CRIL-Netzes gebildet und unterliegt damit einer Closed-World-Assumption.

Die Baumstruktur einer beschreibungslogischen Regel legt eine rekursive Implementation der Abarbeitung einer Regel nahe. Da die Struktur einer Regel einer Subsumption (vergl. hierzu Tabelle 3) entspricht, also der Form (implies C A) genügt, wird nur der Teil-Ausdruck C rekursiv ausgewertet. Die Blattknoten des Regelbaums entsprechen dabei den oben beschriebenen Suchoperationen, alle anderen inneren Knoten entsprechen Mengenoperationen. Der Ablauf der Abarbeitung beginnt mit der Wurzel des Baums und arbeitet sich rekursiv bis zu den jeweiligen Blättern vor. Die Auswertung jedes beschreibungslogischen Teil-Ausdrucks resultiert auf der Seite der CRIL-Netze jeweils in einer Menge von Knoten die, je nach Typ des Operators, eingeschränkt bzw. erweitert werden können. Die durch die Suchoperationen ausgewählten Knotenmengen werden im *Backtracking* an ihre Eltern-Knoten weitergereicht, wo diese (falls vorhanden) mit den Knotenmengen der Geschwister-Knoten in der durch den Operator des Knotens vorgegebenen Art verknüpft werden. Jede zu einen Knoten des Baums gehörende Knotenmenge wird nach Verarbeitung an seine Eltern-Knoten weitergereicht. Erreicht der Prozess im *Backtracking* den Wurzelknoten ist der Prozess beendet. Die resultierende Knotenmenge wird nun mit dem Konzept A angereichert (allen Knoten der Menge wird das Konzept A als Eigenschaft hinzugefügt).

Laufzeitverhalten Folgend eine Betrachtung des Laufzeitverhaltens der einzelnen Operatoren und Konzepte. Zuerst wird eine Einteilung in Funktionsklassen gegeben (s. Tabelle 4 in diesen Abschnitt), die nachfolgend motiviert wird. Abschließend wird eine rekursive Funktion definiert, welche die Abschätzung der Laufzeit eines beliebigen Ausdrucks ermöglicht und zur Betrachtung des Laufzeitverhaltens des Gesamtprozesses genutzt wird.

Die Werte-, Anzahl- und Existensrestriktions-Operatoren (s. Tabelle 4,

erste Zeile) haben ein quadratisches Laufzeitverhalten, da über eine Rollensuche das gegebene CRIL-Netz durchsucht werden muss. Das bedeutet, dass für jeden Knoten jede seiner Relationen (Rollen) betrachtet wird. Bei einen Netz in welchem, im schlimmsten Fall, jeder Knoten mit jeden anderen Knoten des Netzes verbunden ist, muss dies folglich $n * (n - 1)$ mal geschehen, was in einem quadratischen Laufzeitverhalten resultiert. Der bei qualifizierenden Operatoren zusätzlich anfallende konstante Aufwand, durch Abgleich der abgearbeiteten Knoten mit den qualifizieren Konzept, fällt in dieser Betrachtung nicht ins Gewicht.

Die Konzepte und Operatoren mit linearer Laufzeit (Tabelle 4, Zeile 2), wie atomic-Concept, and und or müssen, im schlimmsten Fall, jeden Knoten des CRIL-Netzes einmal abarbeiten. Es sei noch darauf hingewiesen, dass die Operatoren and und or nicht allein von der Größe der Knotenmenge des CRIL-Netzes abhängen, sondern auch von der Anzahl der jeweils zu Verknüpfenden Knotenmengen. Der Grund hierfür ist, dass die Stelligkeit (Anzahl ihrer Argumente) dieser Operatoren beliebig groß sein darf. Dies ändert aber an der Betrachtung aus Sicht der Grösse von CRIL-Netzen nichts an ihrem linearen Verhalten. Die richtige Komplexität wäre $O(m * n)$, mit $m =$ Anzahl der Argumente (entspr. Anzahl der Knotenmengen) $\in \mathbb{N}_0$.

Schließlich haben die top und bottom Konzepte ein konstantes Laufzeitverhalten, da entweder alle Knoten des CRIL-Netzes, oder eine leere Menge zurück gegeben werden. In beiden Fällen ist der Aufwand konstant zur Anzahl der Knoten im CRIL-Netz.

Die Autoren geben nun, aufbauend auf den obigen Klassifizierungen, eine rekursive Formel zur Abschätzung des Gesamtlaufzeitverhaltens eines beliebigen Ausdrucks. Es werden drei disjunkte Mengen definiert, die zusammen alle beschreibungslogischen Operatoren und Konzepte ergeben, die bisher unterstützt werden. Die Aufteilung erfolgt anhand der in Tabelle 4 getroffenen Klassifizierung des Laufzeitverhaltens. Darauf aufbauend wird die rekursive Funktion k definiert, welche, je nach Zugehörigkeit des Operators, einen Ausdruck produziert, der dem Laufzeitverhalten des Operators entspricht. Der resultierende Ausdruck ist dann die Summe der einzelnen Laufzeiten der Teilausdrücke. Folgend die Definitionen mit:

$n \in \mathbb{N}_0 :=$ Mächtigkeit der Knotenmenge des gegebenen CRIL-Netzes,
$\mathcal{K} :=$ {top, bottom},
$\mathcal{L} :=$ {and, or, atomic-concept},
$\mathcal{P} :=$ {at-least, at-most, exactly, qualified-at-least, qualified-at-most, qualified-exactly, top, bottom, all, some, not, atomic-role},
$\mathcal{E} := \mathcal{K} \cup \mathcal{L} \cup \mathcal{P}$ ist die Menge der unterstützten CRIL-DL-Expressions,
$k : \mathcal{E} \to \mathbb{N}_0$, Klassifikationsfunktion, mit $e \in \mathcal{E}$ und $e_i \in \mathcal{E}$ für die Unteraus-

drücke einer Formel:

$$k(e) = \begin{cases} 1, \text{ falls } e \in \mathcal{K} \\ n + k(e_1), \ldots, k(e_i), \text{ falls } e \in \mathcal{L} \\ n^2 + k(e_1), \ldots, k(e_i), \text{ falls } e \in \mathcal{P} \end{cases}$$

Das Laufzeitverhalten des Gesamtprozesses wird damit in polynomieller Zeit erfolgen, da der durch k produzierte Ausdruck eine Summe maximal quadratischer Summanden ergibt. Zählt man die in einer Regel vorkommenden Operatoren und nimmt als pessimistische Abschätzung für jeden Operator die Komplexität $\in O(n^2)$ an, so kann man eine Abschätzung des Laufzeitverhaltens des Gesamtprozesses geben: Mit den Problemgrößen $m :=$ Anzahl der Operatoren des Ausdrucks und $n :=$ Anzahl der Knoten des CRIL-Netzes, ist der Gesamtprozess folglich $\in O(m * n^2)$, da die Komplexität quadratisch zu der Größe des CRIL-Netzes wächst und linear zur Anzahl der Operanden eines Ausdruckes.

4.8 GUI

Zur Konfiguration dieser Erweiterung stehen zwei neue JPanels zur Verfügung, über die sich Regeln und Regelmengen erstellen, editieren und speichern lassen und Regelmengen zu Usecases zugeordnet werden können. Die Implementation als JPanel lässt einen flexiblen Einbau in den GA zu. Desweiteren orientiert sich die Implementation an dem Model-View-Controll-Schema. Sämtliche Funktionalität, die nicht rein GUI-spezifisch ist, wurde in eine Pufferklasse ausgelagert - GUIData. Sie ist dazu gedacht alle Änderungen eines Benutzers zwischenzuspeichern, bis sie übernommen werden sollen und diverse Prüfungen vorzunehmen, bzw. falls möglich/sinnvoll an weitere Klassen wie CrilRuleManager und CrilAugmentor zu delegieren. Die beiden neuen Panels haben wir in einem ersten Entwurf in einer JTabbedPane zusammengefasst (Abb. 3).

Die Abbildung 3 zeigt das JPanel zur Konfiguraion von Regeln. Es beinhaltet eine Liste aller vorhandenen Regeln, Felder zur Anzeige und zum Editieren der Eigenschaften von Regeln und Buttons für verschiedene Aktionen. Das Verhalten ist recht simpel. Zu jeder Zeit werden im Panel entweder die Daten des in der Liste der vorhandenen Regeln ausgewählten Elements angezeigt, oder die Daten einer neuen noch nicht gespeicherten Regel. Um eine neue Regel eingeben zu können, muss zuvor der Button ,,Neu" betätigt werden. Hierdurch werden auch alle Felder für Regeldaten geleert.

Sobald Daten im Panel editiert werden (einer vorhandenen oder neuen Regel), werden die Buttons ,,OK" und ,,Abbrechen" aktiviert und alle anderen Buttons deaktiviert. So müssen die Änderungen entweder übernommen

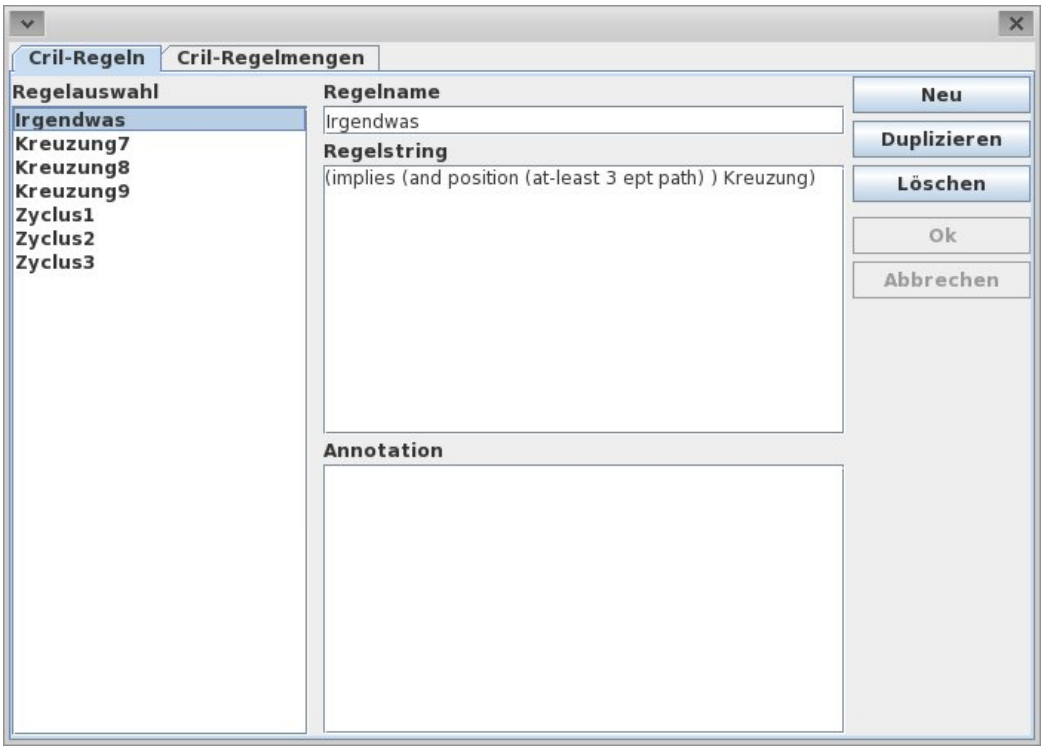

Abbildung 3: Das JPanel zur Erstellung und Modifikation von Regeln.

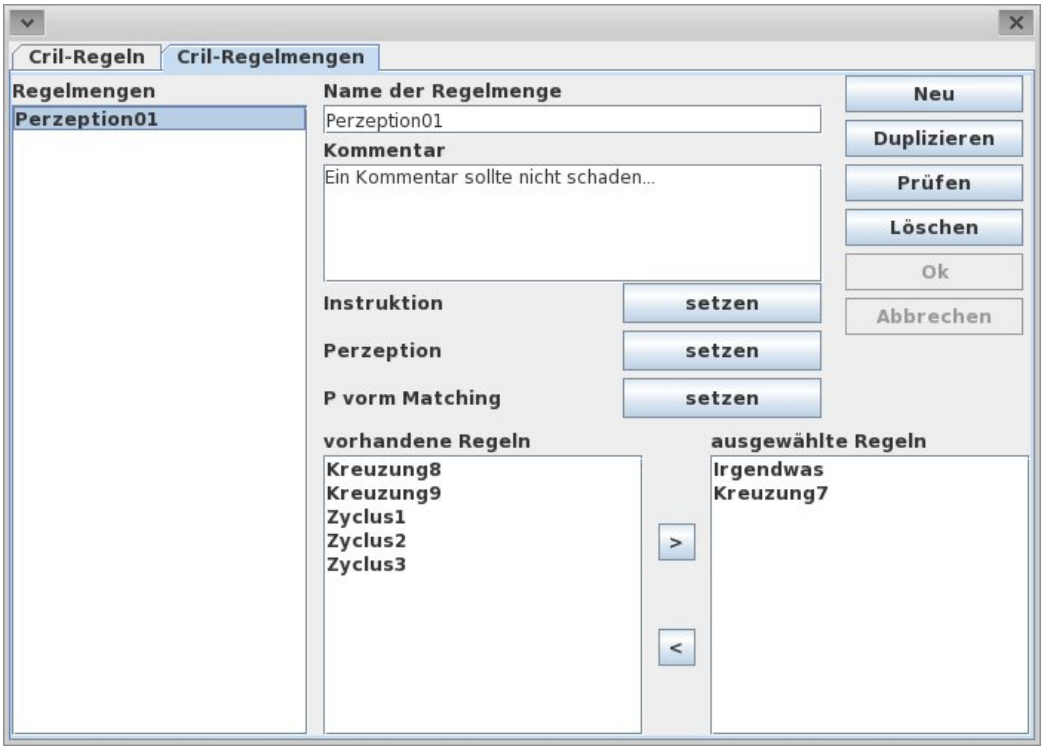

Abbildung 4: Das JPanel zur Erstellung und Modifikation von Mengen von Regeln.

oder verworfen werden, bevor zu einer anderen Regel gewechselt werden kann. Durch zusätzliches Deaktivieren der JTabbedPane wurde sichergestellt, dass während des Editierens einer Regel nicht gleichzeitig deren Zugehörigkeit zu Regelmengen verändert werden kann. So wird im Regelmengenpanel nicht auf Regeln mit offenen Änderungen gearbeitet.

Der Aufbau des Panels zur Manipulation von Regelmengen gleicht dem des Regel-Panels (vgl. Abbildung 4). Links eine Liste aller vorhandenen Regelmengen, rechts Buttons für verschiedene Aktionen und in der Mitte die Daten der angezeigten Regelmenge. Neben Standardfeldern für Namen und Beschreibung der Regelmengen, die keiner weiteren Erklärung bedürfen, gibt es im unteren Bereich des Panels zwei Listen für die Zuordnung von Regeln zu einer Regelmenge. Die rechte Liste enthält dabei alle Regeln, die der Regelmenge hinzugefügt wurden, die linke enthält eine Auflistung aller noch vorhandenen Regeln. Über zwei Buttons zwischen den Listen können einzelne oder mehrere Einträge zwischen den beiden Listen hin und her geschoben werden.

In der Mitte des Panels ist ein weiteres Panel eingebunden, um eine Regelmenge zu einem Usecase zuzuordnen. Das Panel DefaultsetConfigPanel erzeugt auf Basis der Enumeration CrilRulesetUsecase automatisch Paare aus je einem Label und einem Button für jeden Eintrag in der Enumeration. Wie in Abbildung 4 erkennbar ist, sind derzeit drei Standardeinsatzgebiete für die Anreicherung angedacht, zu denen eine Regelmenge als automatisch benutzter Standard gesetzt werden kann. Einem Usecase kann lediglich eine Regelmenge zugeordnet werden, daher wird bei Betätigung eines ,,setzen"-Buttons für einen Usecase eine vorhendene Zuordnung ggf. überschrieben. Die Zuordnungen einer Regelmenge werden durch eine grüne Einfärbung des Labels und einen Änderung der Beschriftung des Buttons in ,,entfernen" kenntlich gemacht.

Bei Änderungen der Daten verhält sich das Regelmengen-Panel wie das für einzelne Regeln: Bei der ersten Änderung werden die Buttons zum Duplizieren, Löschen der Regelmenge und zum Leeren aller Felder deaktiviert und die Buttons ,,OK" und ,,Abbrechen" aktiviert. Werden die Änderungen durch die aktiven Buttons angenommen oder verworfen, wird der Ursprungszustand der Buttons wieder hergestellt. Über den Button ,,Prüfen" kann zu jeder Zeit die Zyklenfreiheit der Regelmenge überprüft werden, da zur Zeit keine zyklischen Regelmengen verarbeitet werden können.

5 Stand des Projekts

Das in den vorherigen Kapiteln Beschriebene, wurde vollständig implementiert, allerdings nicht in den GA integriert. Da das Crilinterface die einzige Schnittstelle zum GA darstellt, sollten die nötigen Schritte, um den Augmentor zu integrieren, überschaubar sein. Aus unserer Sicht stellen sich die noch ausstehenden Arbeiten wie folgt dar:

- In der Klasse CrilRuleExecutor, wird von der Methode intersec ein Algorithmus, mit einen, im schlechtesten Fall, $O(n!)$ Laufzeitverhalten implementiert, der durch eine $O(n)$ Variante ersetzt werden kann (mit n = Anzahl der zu schneidenden Mengen). Die Linearität ergibt sich aus den Umstand, das es genügt, eine beliebige der gegebenen Mengen, mit allen anderen zu schneiden, da ja die Ergebnismenge der Operation nicht größer sein kann, als jede der gegebenen Mengen. Dieser Umstand wurde auch in die Betrachtungen zum Laufzeitverhalten der Operatoren in Abschnitt 4.7 einbezogen.

- Die Implementation des Serialisierers, in der Klasse CrilRuleManager, benutzt die Java Hashfunktion um die Zuordnungen innerhalb der Re-

geldaten sicherzustellen. Dies kann durch einen inkrementierenden Index sehr viel einfacher realisiert werden, ohne die bei großen Datenmengen zu erwarteten Kollisionen. Alternativ kann bei Berechnung des Hashwertes geprüft werden ob dieser schon existiert, was eine Buchführung der schon berechneten Hashwerte bedingt.

- Das Definieren von Menueinträgen, um die GUI des Augemtors in den GA zu Integrieren.
- Die Konfigurationseinstellungen das Augmentors in die des GA einbinden.
- Die Definition von Regelmengen für alle Usecases, die die entsprechenden CRIL-Graphen sinnvoll anreichern.

6 Evaluation

Wie schon in der Motivation (siehe Abschnitt 1.2.1) dargelegt sollten einige Schwächen des Matchens durch regelhafte Anreicherung der CRIL-Graphen ausgeglichen werden.

- Vor der Integration des (CRIL-)Augmentors im GA war das Perzeptierte alle verfügbare Information im Perzeptions-Graphen des GA. Eventuell ableitbare Information des CRIL-Graphen musste explizit enthalten sein, um später beim Matchen zur Verfügung zu stehen. Selbst einfach ableitbare Information war nicht allein aus der Perzeption zu gewinnen.
- Beim Nutzen des Karteneditors für Skizzen gilt dies ebenfalls. Alle Informationen, welche dem GA später beim Matchen zur Verfügung stehen sollen, mussten von Hand angegeben bzw. annotiert werden, welches die Erstellung möglichst vollständiger Skizzen umständlich machte.
- Der GA verhielt sich beim Navigieren anhand der von ihm perzeptieren Welt entgegen dem menschlichem Vorbild, dessen Verhalten er simulieren sollte, da Menschen automatisch weitere Informationen aus dem Gesehenen/den Instruktionen ableiten.

Nach erfolgreicher Eingliederung des Augmentors im GA ist es nun möglich die CRIL-Graphen des GA um Informationen zu erweitern.

- Jetzt ist regelbasiertes Schließen möglich. Die Regeln können jederzeit einzeln aktiviert und deaktiviert werden, sollte eine Regel unerwünschte Resultate hervorbringen. Auch ganze Regelmengen können derart angepasst und an bestimmten Stellen angewendet werden, bis sich optimale Ergebnisse erziehlen lassen.

- Implizierte Informationen, d.h. implizites Wissen über die Welt, können nun mit Hilfe entsprechender Regeln expliziert werden. Perzeptiert der GA eine Kreuzung indem er z.B. sieht, dass mindestens 3 Wege an einem Punkt zusammenlaufen (siehe *Netz A* in Abbildung 5), so kann er diesen Punkt als Kreuzungspunkt deklarieren (siehe *Netz B* in Abbildung 5).

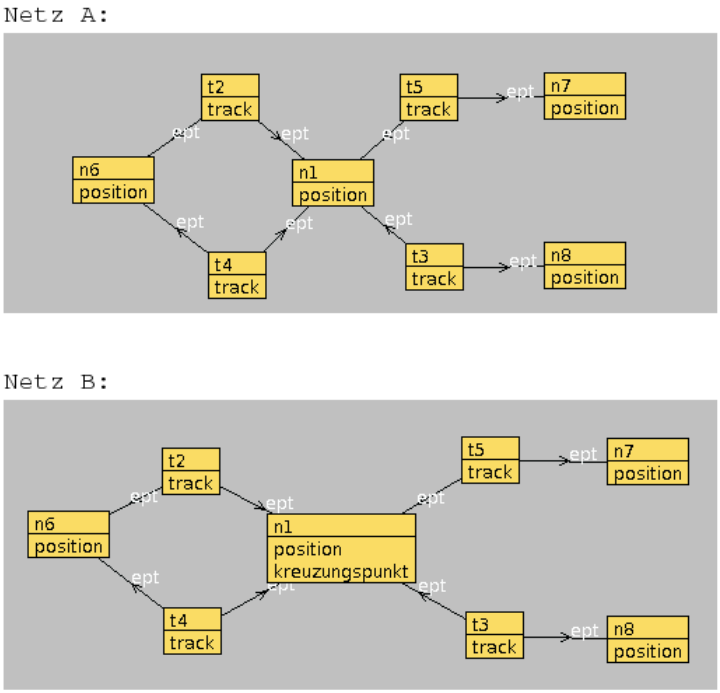

Abbildung 5: Ein CRIL-Netze vor (Netz A) und nach der Anreicherung von Informationen bezüglich eines Kreuzungspunktes (Netz B).

- Auch bei der Kartenerstellung kann man automatisiert Regeln anwenden und so dem GA mehr mögliche Bezugspunkte zum Matchen der CRIL-Graphen geben.

- Auch Informationen aus den Taxonomien, in denen Beziehungen zwischen Konzepten festgehalten sind, können nun dem CRIL-Graphen hinzugefügt werden. Hat der GA z.B. die Anweisung erhalten, beim nächsten Baum rechts abzubiegen, so kann er nach dem Perzeptieren einer Esche mit Hilfe der Taxonomie schließen, dass die Esche ein Baum ist, den Perzeptions-Graphen entsprechend anreichern und dann später beim Matchen auf diese Informationen zurückgreifen.

Diese Änderungen machen einerseits das Matchen robuster gegenüber verschiedenen Instruktionen indem Synonyme, Verallgemeinerungen und Verfeinerungen ebenfalls beachtet werden können, andererseits kann durch das regelbasierte Erweitern implizit gegebene Information expliziert und somit später zum Matchen zur Verfügung gestellt werden. Dabei werden die CRIL-Graphen nicht blind mit allen impliziten Informationen angereichert, welche sich im CRIL-Graphen oder der Taxonomie verbergen, sondern nur mit denen, welche vom Benutzer zu einem von ihm gewählten Zeitpunkt während der Abarbeitung der Instruktionen und in einem vom Benutzer festgelegtem Maße bestimmt wurden.

7 Ausblick

Auf Basis dieser Arbeit sind mehrere Erweiterungen denkbar. Zum Einen ist es möglich die Grenzen der jetzigen Implementation anzugehen, zum Anderen könnte der Geometrische Agent (GA) auch – mit erheblich mehr Aufwand – stärker an diese Arbeit angenähert werden. Auszughalber sollen hier folgende Möglichkeiten kurz diskutiert werden:

1. Rollenkonstruktoren in Regeln
2. Zyklen in Regelmengen
3. weitere Regeltypen erlauben (z.B. Konzeptdefinitionen)
4. Ableitung von Suchmustern aus Regeln
5. Redesign des GA auf Basis von Beschreibungslogiken

Einige dieser Punkte sind fast schon trivial und nur aus dem einen Grund nicht bereits Bestandteil dieser Arbeit, nämlich dem Zeitfaktor. Von den Grenzen des zur Verfügung gestandenen Zeitkontinuums einmal abgesehen, standen die Planung und Implementierung von Grundbausteinen und Infrastruktur im Vordergrund, um regelbasiertes Schließen über CRIL-Netzen mit Hilfe von Beschreibungslogiken überhaupt zu ermöglichen.

Rollenkonstruktoren/Rollensubsumption Der Dialekt von Beschreibungslogiken, der derzeit akzeptiert wird, umfasst weder Rollenkonstruktoren[19], noch Rollensubsumption. Die Abwesenheit von Rollenkonstruktoren ist einer der Punkte, die einem begrenzten Zeitfenster des Projekts geschuldet sind, wohingegen Rollensubsumption bewusst außen vor gelassen wurde:

Alle bislang akzeptierten Regeln haben lediglich die Anreicherung vorhandener Knoten mit zusätzlichen Attributen zur Folge und verändern nicht die Struktur des Netzes über dem sie interpretiert werden. Rollensubsumption hingegen würde neue Relationen zwischen einzelnen Knoten erschließen und somit die Netzstruktur verändern. Eine Regel $r \sqsubseteq s$, wobei r eine atomare Rolle und s eine komplexe Rollenbeschreibung sei, würde bedeuten, dass zwei Knoten a, b im CRIL-Netz, die s erfüllen, auch r erfüllen. Bei der Anreicherung sollte dann die Relation $r(a, b)$ in das Netz eingefügt werden. Insbesondere die Deklaration von transitiven und reflexiv-transitiven Rollen hätte potentiell eine zwar aufgrund der Endlichkeit von CRIL-Netzen nur eine endliche, aber dennoch signifikante Steigerung der Komplexität eines Netzes zur Folge.

Zyklen in Regelmengen sind aus Komplexitätsgründen nicht in der derzeitigen Ausbaustufe enthalten. Die logische Konsequenz zyklenbehafteter Regelmengen ist, dass ihr Ergebnis nicht nach einmaliger Ausführung erschlossen werden kann. Hier müsste der Fixpunkt der Interpretation einer Regelmenge gefunden werden, der jedoch bedingt durch die Endlichkeit aller relevanten Komponenten (Taxonomie, Regelmengen, CRIL-Netze) bereits durch eine simple Iteration der Ausführung in endlicher Zeit erreicht werden könnte.

Als weiterführende Literatur sei auf [Neb91] verwiesen, welches sich mit zyklischen TBoxen in Beschreibungslogiken und deren Komplexität i.B.a. Inferenzprobleme beschäftigt.

Weitere Regeltypen Auf eine Erweiterung um Rollensubsumptionsaxiome wurde bereits im vorherigen Abschnitt eingegangen, so dass als weitere Regeltypen noch Konzeptdefinitionen und komplexere GCIs (*general concept inclusion axioms*[20]) in Frage kommen. Beide könnten die Mächtigkeit des Ableitungsmechanismus steigern, bergen jedoch auch gewisse Probleme im Kontext von CRIL-Netzen.

Wenn A ein atomares Konzept ist und C und D komplexe Konzeptbeschreibungen sind, dann haben die bisher akzeptierten Regeln die Form

[19] z.B. Vereinigung, Komplement, Negation, Komposition

[20] siehe Kapitel 2.4.2 (Seite 17)

$C \sqsubseteq A$. Man könnte diese eingeschränkte Form von GCIs etwas aufweichen und die Implikation bestimmter komplexer Konzeptausdrücke zulassen, also $C \sqsubseteq D$. Da jedoch jede Regel einzeln bearbeitet wird und nicht über der Regelbasis räsoniert wird, muss sichergestellt werden, dass D auch in CRIL auch formulierbar ist. Dies gilt insbesondere nicht für die Disjunktion und Negation. Keine Probleme ergeben sich dagegen bei der Konjunktion atomarer Konzepte. Es wäre also genau zu prüfen, welche Struktur für die implizierten Konzeptbeschreibungen D sinnvoll sind.

Für Konzeptdefinitionen $A \equiv C$ gilt gewissermaßen das Gleiche. Die eine Richtung dieser Äquivalenz wird bereits durch den zulässigen Regeltyp abgedeckt ($C \sqsubseteq A$) und die andere Richtung ($A \sqsubseteq C$) ist ein Spezialfall der allgemeinen GCIs $C \sqsubseteq D$ mit den gleichen Problemen in Bezug auf das Schließen über CRIL-Netzen. Somit muss bei der Betrachtung von Konzeptdefinitionen als Regeltyp entweder bei der Anreicherung geprüft werden, welche Implikationsrichtungen sinnvoll sind, oder die Mächtigkeit der rechten Seite der Äquivalenz soweit eingeschränkt werden.

Ableitung von Suchmustern aus Regeln Zu Beginn des Projekts wurde auch die Möglichkeit diskutiert, aus den Regeln Suchmuster für die Instruktionsphase des GA abzuleiten. Insbesondere für die Sprachinstruktion wurde erwogen, durch Regeln beschriebene Konzepte im Instruktionstext zu suchen und für diese aus den Regeln die erwartete Netzstruktur abzuleiten. Der Vorteil dieser Methode läge darin, dass u.U. aus den Instruktionen spezifischere CRIL-Netze abgeleitet würden und eine höhere Genauigkeit beim Abgleich mit den aus der Perzeption hervorgegangenen Netzen erzielt werden könnte.

Grundlage für diesen Denkansatz ist die Idee eines *minimalen Netzes*, das eine Regel $C \sqsubseteq A$, bzw. deren Konzeptbeschreibung C, erfüllt. Ein CRIL-Netz N soll in diesem Sinne minimal zu einer Konzeptbeschreibung C heißen, wenn es Teilnetz jedes zu C passenden CRIL-Netzes ist und es kein CRIL-Netz N' gibt, das nach einer nicht näher bestimmten Gewichtsfunktion kleiner ist als N.

Streng genommen funktioniert dies lediglich für Konzeptdefinitionen, oder falls zwei Regeln $C \sqsubseteq D$ und $D \sqsubseteq C$ in der Regelmenge existieren, damit eine Biimplikationsbeziehung zwischen dem in der Instruktion gefundenen Konzept und der aus einer entsprechenden Regel stammenden Konzeptbeschreibung besteht. Darüber hinaus trifft man auch hier auf die Problematik aus dem letzten Absatz, dass Disjunktionen und Negationen in CRIL nicht formuliert werden können. Deshalb ist die Existenz eines minimalen CRIL-Netzes zu einer Konzeptbeschreibung keinesfalls sichergestellt und müsste

wieder von Fall zu Fall geprüft werden. Bei diesem Unterfangen sollte also genau geprüft werden, ob der Nutzen den Implementierungsaufwand rechtfertigt, da derzeit nicht einmal Konzeptdefinitionen oder GCIs in ihrer allgemeinen Form als Regeltypen zulässig sind.

Stärkere Einbindung von Beschreibungslogiken in den GA Mit dieser Arbeit finden Beschreibungslogiken erstmals Verwendung im GA. Allerdings wird im Grunde genommen lediglich die Syntax von Beschreibungslogiken ausgeliehen. Als Schlussmechanismus kommt derzeit eine graphenbasierte Suche in CRIL-Netzen anstatt des bereits existierenden Tableaubeweisers zum Einsatz. Eine sinnvolle Überlegung wäre jedoch dieses Projekt als Anstoß zu nehmen, den GA grundlegend oder in Teilen auf Basis von Beschreibungslogiken neu aufzusetzen. Es handelt sich bei Beschreibungslogiken um ein aktives und gut verstandenes Forschungsgebiet und sie passen technisch gesehen gut zur Architektur des GA. Derzeit gibt es schon eine Zweiteilung der Wissensbasis des GA in einen terminologischen und assertionalen Teil in Form der Taxonomie und der CRIL-Netze aus den Instruktionen und der Perzeption. Ein Redesign als TBox und ABox einer Beschreibungslogik liegt also nah.

Zu den Problemen, die sich im Zuge eines Umbaus ergeben würden, gehören unter anderem der Umgang mit drei- und mehrstelligen Prädikaten in der Taxonomie, da die Rollen einer Beschreibungslogik per Definition zweistellig sein müssen und die Erkennung von unvollständigen Matches zwischen Perzeption und Instruktion, bzw. die Klassifizierung und Auswahl bei mehreren Matches. Die Aufgabe des GA, sich in unbekannter Umgebung und mit unvollständigen und inkorrekten Instruktionen fortzubewegen, benötigt eine gewisse Nachlässigkeit beim Abgleich der Instruktionen mit der Perzeption. Die logische Strenge beschreibungslogischer Inferenzen müsste also ggf. aufgeweicht werden.

Der Nutzen von Beschreibungslogiken als Form der Wissensrepräsentation und Reasoningkomponente des GA läge, neben der Flexibilität und Mächtigkeit des Formalismus, vor allem in der Vereinigung der Wissensrepräsentation in einem einzigen Formalismus. Taxonomie, Wissen über konkrete Weltzustände und die neu hinzugekommenen Anreicherungsregeln könnten in TBox und ABox vereint und gewissermaßen gemeinsam betrachtet werden. Besonders die Anreicherung würde davon profitieren, die Beziehungen unter Prädikaten und Relationen in der Taxonomie berücksichtigen zu können. Obendrein könnten vorhandene Tools zum Design und Verarbeiten von beschreibungslogischen Wissensbasen eingebunden und Algorithmen verwendet werden, deren verarbeitungseigenschaften untersucht und bewiesen sind.

Diese grundlegenden Änderungen wären ein guter Ansatz für ein dringend notwendiges Refactoring vieler Komponenten des GA, das vielleicht in eine Version 2.0 des Geometrischen Agenten münden könnte.

Literatur

[Baa03] Franz Baader. Appendix: Description Logic Terminology. In *[BCM+03]*, pages 485–495. 2003.

[Baa09] Franz Baader. Description logics. In *Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009*, pages 1–39. Springer-Verlag, 2009.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[BL06] Franz Baader and Carsten Lutz. Description Logic. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *The Handbook of Modal Logic*, pages 757–820. Elsevier, 2006.

[Hab86] Christopher Habel. *Prinzipien der Referentialität*. Springer-Verlag, 1986.

[Hay79] Patrik J. Hayes. The Logic of Frames. In Dieter Metzing, editor, *Frame Conceptions and Text Understanding*, Research in Text Theory, pages 46–61. Walter de Gruyter and Co., 1979.

[Hel03] Jan Hendrik Helwich. Graphenbasierte Navigation eines geometrischen Agenten: Integration von Perzeption und Instruktion. Master's thesis, Universität Hamburg, 2003.

[Ill08] Roland Illig. Optimierung und Messung Beschreibungslogischer Beweisverfahren. Projektbericht, 2008.

[NB03] Daniele Nardi and Ronald J. Brachman. An Introduction to Description Logics. In *[BCM+03]*, pages 5–44. 2003.

[Neb91] Bernhard Nebel. Terminological Cycles: Semantics and Computational Properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–362, San Mateo, CA, 1991. Morgan Kaufman.

[Sol06] Arved Solth. Effiziente Expansion durch Redundanzvermeidung in einem beschreibungslogischen Tableau-Beweiser. Projektbericht (Universität Hamburg), 2006.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Complemts. *Artificial Intelligence*, 48(1):1–26, 1991.