

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik



Diplomarbeit

DDD*query*:
Anfragen an komplexe Korpora

Thorsten Vitt

18. Oktober 2005

Betreuung:
Dr. Lukas C. Faulstich
Prof. Dr. Ulf Leser

Inhaltsverzeichnis

1	Einführung	11
1.1	Hintergrund	11
1.2	Zielsetzung	12
1.3	Architektur und Workflow	12
1.4	Überblick über diese Arbeit	14
2	Datenmodell des Korpus	15
2.1	Einführung	15
2.2	Grundlegende Elemente	15
2.3	Alignments	19
2.3.1	Greediness	21
2.3.2	Relationen zwischen Alignments	22
2.3.3	Alignments von Annotationselementen	22
3	Benutzungsschnittstellen	25
3.1	Webschnittstellen	25
3.1.1	Digitales Mittelhochdeutsches Textarchiv	25
3.1.2	Uni Bonn	26
3.1.3	Thesaurus Indogermanischer Text- und Sprachma- terialien (TITUS)	26
3.1.4	Codices Electronici Ecclesiae Coloniensis (CEEC)	28
3.1.5	Mittelhochdeutsche Begriffsdatenbank	28
3.1.6	Lancaster Newsbook Corpus	29
3.1.7	Corpus of Middle English Prose and Verse	31
3.1.8	Corpus del Español	32
3.1.9	Annotated Corpora of Text (ACT)	32
3.1.10	IMS Corpus Workbench / CQP	34
3.1.11	A Glossarial DataBase of Middle English	36
3.1.12	The Canterbury Tales Project	37
3.1.13	Corpus Scriptorum Latinorum (CSL)	37
3.2	Nicht-Web-Tools	37

3.2.1	ICECUP III	37
3.2.2	TIGERSearch und TIGERin	39
3.2.3	WordCruncher	39
3.3	Sonstiges	40
3.3.1	Interlinearer Text	40
3.4	Auswertung existierender grafischer Benutzungsschnittstellen	41
3.4.1	Textbelege und Volltext	41
3.4.2	Annotationen	41
3.4.3	Keyword in Context (KWIC)	42
3.4.4	Alignments	42
3.4.5	Frequenztafel	43
3.4.6	Verzeichnisse	43
3.4.7	Metadaten	44
3.4.8	Navigation	44
4	Datenschema für Anfrageergebnisse	45
4.1	Entwurfskriterien	45
4.2	Daten für die GUI-Darstellung	46
4.2.1	Textbelege und Volltext	46
4.2.2	Annotationen	46
4.2.3	Keyword in Context (KWIC)	47
4.2.3.1	Umsetzung der Markierungen	48
4.2.4	Alignments	51
4.2.5	Frequenztafel / Aggregierte Daten	53
4.2.6	Verzeichnisse	54
4.2.7	Metadaten	54
4.2.8	Navigation und Query Refinement	55
4.3	Entwurf des Ergebnisschemas	55
4.3.1	gXDF	55
4.3.2	Erweiterungen und Anpassungen für <i>DDDquery</i>	56
4.3.2.1	Selektierte Elemente	56
4.3.2.2	Benannte Markierungen	57
4.3.2.3	Aggregierte Inhalte	57
4.3.2.4	Texte, Spans und Alignments	57
4.3.2.5	Weiternavigation mit IDs	58
4.3.3	Beispiel	58

5	Linguistische Anfragesprachen	61
5.1	Emu	61
5.2	XPath-Derivate	62
5.2.1	LPath	62
5.2.2	Generalized XPath (Cassidy)	62
5.2.3	EXPath	63
5.3	TGrep2	65
5.4	CQP	65
5.5	TIGERSearch	66
5.6	CorpusSearch	67
5.7	Xaira	68
5.8	Weitere Arbeiten	68
5.9	Zusammenfassung	70
6	Anforderungen an die Anfragesprache	71
6.1	Anforderungsanalyse für Baumbanken von Lai und Bird (2004)	71
6.2	Datenmodell	72
6.3	Volltextsuche und Pattern Matching	73
6.4	Relationen zwischen Spans	73
6.5	Relationen zwischen Elementen	74
6.6	Spezielle Anforderungen an linguistische Anfragesprachen	74
6.7	Elemente, Spans etc. in Verbindung	74
6.8	Sequenzoperatoren	74
6.9	Aggregierte Inhalte	75
6.10	Anfrageergebnis	75
6.11	Eigenschaften der Sprachsyntax	75
6.12	Übersicht	76
7	Anfragesprache	79
7.1	Grundlagen	79
7.1.1	Orientierung an und Abgrenzung von XPath . . .	79
7.1.2	Ausführliche und abgekürzte Syntax	80
7.2	Anfrageergebnisse	81
7.3	Bibliographische Daten, Headerdaten	81
7.4	Sprachelemente	82
7.4.1	Pfadmuster	82
7.4.1.1	Semantik	82
7.4.1.2	Auswertung: Kontext und Focus	83
7.4.1.3	Syntax	83

Inhaltsverzeichnis

7.4.2	Schritte	84
7.4.2.1	Syntax	85
7.4.3	Typen entlang eines Pfades	86
7.4.3.1	Abbildungen von Annotationselementen zu Spans	87
7.4.3.2	Abbildungen von einem Span zu zugehö- rigen Annotationsobjekten	87
7.4.3.3	Automatisch eingefügte Konvertierungs- schritte	89
7.4.4	Achsen	90
7.4.4.1	Typen einer Achse	91
7.4.4.2	Achsen im Vergleich zu XPath	91
7.4.4.3	Verfügbare Achsen	92
7.4.4.4	Die layer-Achse: Assoziationen zwischen Annotationsebenen	95
7.4.5	Knotentests	96
7.4.5.1	Volltextsuche	97
7.4.5.2	Verfügbare Knotentests	98
7.4.6	Kontextauswahl und Treffermarkierungen (Marker)	99
7.4.7	Variablen / Joins	101
7.4.8	(Aggregations-)funktionen	102
7.4.9	Alignments	102
7.4.10	Boolesche Ausdrücke und Prädikate	103
7.4.11	Ausdrücke, Vergleiche und Arithmetik	105
7.4.12	Anfragen	105
7.4.13	Reguläre Pfadausdrücke	105
7.4.14	Leerraum (Whitespace)	106
7.5	Erweiterbarkeit	107
8	Implementierung	109
8.1	Parser und abstrakter Syntaxbaum	110
8.2	Normalisierung des abstrakten Syntaxbaums	111
8.3	Umsetzung von Anfragen und Ausgaben in einem RDBMS	112
8.3.1	Datenbankschema	112
8.3.1.1	Graphstruktur	112
8.3.1.2	Spans	113
8.3.1.3	Texte	114
8.3.1.4	Headerdaten etc.	114
8.3.1.5	Layer	114
8.3.1.6	Alignments	114

8.3.2	Rekonstruktion des Anfrageergebnisses in XML / gXDF	115
8.3.2.1	Strukturdaten	115
8.3.2.2	Textabschnitte	115
8.3.3	Umsetzung der SQL-Anfragen	116
8.3.3.1	Rückgabeformat	116
8.3.3.2	Schritte und Pfade	116
8.3.3.3	Umsetzung der Markierungen	118
8.3.3.4	Suche / Span-Tests	119
8.3.3.5	Variablen	119
8.3.3.6	Einfache Prädikate	119
8.4	Umsetzung mit LoToS	119
9	Zusammenfassung und Ausblick	123
A	Grammatik	127
A.1	Schritte und Pfade	127
A.1.1	Reguläre Pfadausdrücke	128
A.1.2	Detaillierte Bestandteile eines Schrittes	128
A.2	Ausdrücke	129
A.3	Startsymbol	131
	Literaturverzeichnis	133

Inhaltsverzeichnis

Abbildungsverzeichnis

1.1	Modell einer Sitzung mit <i>DDDquery</i>	13
1.2	Struktur dieser Arbeit	14
2.1	Grundlegende Elemente des Korpusdatenmodells	16
2.2	Datenmodell des Korpusgraphen (Entwurf)	18
2.3	Zwei Umschriften desselben Texts	19
2.4	Datenmodell für Alignments (Ausschnitt)	20
2.5	Elemente eines Alignments am Beispiel (Ausschnitt)	20
3.1	Volltextsuche im Digitalen Mittelhochdeutschen Textarchiv	27
3.2	Physische und logische Struktur in TITUS' Webschnittstelle.	28
3.3	Die Webschnittstelle der Mittelhochdeutschen Begriffsdatenbank	30
3.4	Alignments im Lancaster Newsbook Corpus	31
3.5	Webschnittstelle des Corpus del Español	33
3.6	Websuche in ACT	34
3.7	CQP-Webdemo	35
3.8	Verteilungs- und Frequenzsuche mit CQP	36
3.9	Anfrageergebnisse in ICECUP III	38
3.10	Syntaxbaum in ICECUP III	38
3.11	Korpusmetainformationen in TIGERSearch	40
3.12	Alignierungen und Interlinearübersetzung. Beispiele nach (Lüdeling et al., 2003).	41
3.13	Alignierung von eng- und weitdiplomatischer Fassung	43
4.1	Suchtypen	53
5.1	Beispiel für einen GODDAG	64
5.2	Beispielanfragen für Xaira CQL 2	69
6.1	Sekundäre Kanten	72

7.1	Grammatik für Knotentests	97
8.1	Ausschnitt aus dem Modell des abstrakten Syntaxbaums .	110
8.2	Grundlegendes Datenbankschema	113
8.3	Ergebnisse der SQL-Anfragen	116

Tabellenverzeichnis

7.1	Achsen für den Weg von Span nach Element	90
7.2	Syntax zur Ausgabe und Markierung von Knoten	100
7.3	Beispiele für Ausdrücke mit Markern	100

Listings

4.1	Integration von Textabschnitten in Standoff-Annotationen	47
4.2	Ergebnis einer Anfrage	60
7.1	Aggregierte Inhalte in einem Anfrageergebnis	103
8.1	Umsetzung der Anfrage page/line	117

1 Einführung

1.1 Hintergrund

Das Projekt *Deutsch Diachron Digital (DDD)*¹ entwickelt ein umfangreiches diachrones Korpus des Deutschen: Eine Sammlung zahlreicher historischer und zeitgenössischer deutscher Texte, teils in unterschiedlichen, miteinander alignierten Versionen und mit verschiedenen Annotationen versehen.

Deutsch Diachron Digital hebt sich durch einige Eigenschaften von anderen Korpusprojekten ab:

- Texte werden mit Annotationen in *mehreren, zueinander orthogonalen Annotationsebenen* versehen.

So kann ein Text etwa gleichzeitig mit seiner physischen (Zeilen, Seiten, ...) und seiner logischen (Wörter, Sätze, ...) Struktur annotiert sein.

- Texte können in verschiedenen Ausgaben vorliegen, die *miteinander aligniert* sind.

Beispielsweise kann ein Text in einer engdiplomatischen (d. h. nahe an der physischen Gestalt orientierten) und einer weiter normalisierten Fassung vorliegen, die aneinander orientiert sind; oder es ist denkbar, einen Text und seine Übersetzung in eine andere Sprache miteinander zu alignieren.

- Das Korpus soll in einer (objekt-) relationalen Datenbank mit XML- und Volltextunterstützung verwaltet werden.
- Die Ergebnisse von Anfragen sollen maschinelle Auswertungsmöglichkeiten eröffnen, aber auch in einer für Menschen sinnvoll rezipierbaren Form dargestellt werden können.

¹ <http://www.deutschdiachrondigital.de>

1 Einführung

So sind bei einer Anfrage z. B. nach Wörtern mit bestimmten Eigenschaften nicht nur die passenden Wörter, sondern die sie enthaltenden Sätze darzustellen und die relevanten Wörter darin hervorzuheben.

1.2 Zielsetzung

Die von Dipper et al. (2004) vorgestellte grundlegende Systemarchitektur sieht vor, dass Benutzer über lokale Clients oder über ihren Webbrowser mit einer *Middleware* agieren, die Anfragen an den Datenbankserver stellt und deren Ergebnisse sinnvoll für die Benutzer aufbereitet.

Das System soll dabei soweit wie möglich auf Standard-XML-Techniken und -Werkzeuge zurückgreifen: Insbesondere bieten sich zur Erzeugung der diversen Ausgabeformate XML-Transformationstechniken wie XSLT (Kay et al., 2003) an. Dazu müssen die Anfrageergebnisse zunächst in ein *Datenschema* (auf XML-Basis) gebracht werden, das eine geeignete Basis für die erwünschten Ausgabeformate ist, indem es für alle unterstützten Ausgaben die benötigten Informationen enthält.

Obschon die Anfragen an das RDBMS in SQL erfolgen werden und als Benutzungsschnittstelle für die Endanwender des Korpus aus den Sprachwissenschaften Webformulare unterschiedlicher Komplexität vorgesehen sind, ist die Entwicklung einer *Anfragesprache* sinnvoll: Zum einen kann diese Sprache als Schnittstelle für Experten oder das Korpus automatisch bearbeitende Anwendungen dienen, zum anderen als Interface zwischen formularartigen oder grafischen Anfrageschnittstellen und dem RDBMS. Dies ist auch dadurch motiviert, dass bereits einfache Anfragen auf der Korpusstruktur recht komplexen und schwer verständlichen SQL-Code erfordern (vgl. Vitt, 2004).

Ziele der Diplomarbeit sind Entwurf und Implementierung einer Anfragesprache für DDD sowie der Entwurf des XML-Datenschemas für die Ergebnisse des Anfragemoduls.

1.3 Architektur und Workflow

Abbildung 1.3 auf der nächsten Seite demonstriert einen typischen Ablauf einer Sitzung mit der zu entwickelnden Anfragesprache *DDDquery*:

Ein Benutzer stellt eine Anfrage über eine Benutzungsschnittstelle. Diese leitet die Anfrage in der in dieser Arbeit entwickelten Anfragesprache weiter. Das Anfragesystem führt diese Anfrage dann auf der

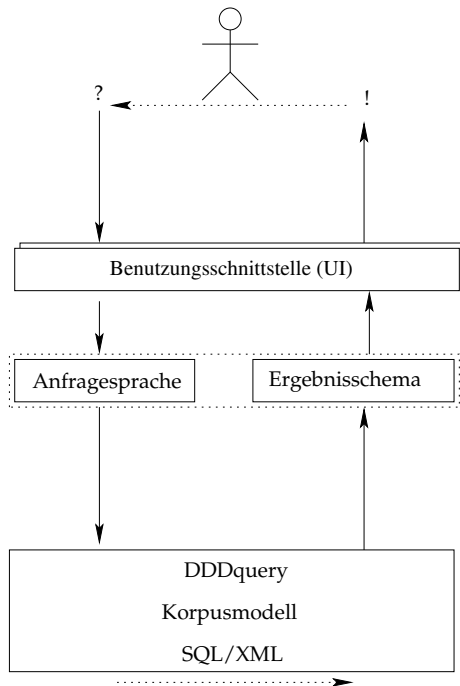


Abbildung 1.1: Modell einer Sitzung mit DDDquery

Datenbank aus und liefert die Ergebnisse im in Kapitel 4 entwickelten Ergebnisschema zurück an die Benutzungsschnittstelle, die die Ergebnisse visuell aufbereitet und dem Benutzer präsentiert. Dieser kann dann seine Anfrage überarbeiten und neu stellen, und der Kreislauf beginnt von neuem.

Bezüglich des Einflusses der Benutzungsschnittstellen auf die Anfrage sind drei Varianten denkbar:

1. Die Benutzer formulieren ihre Anfragen komplett selbst in der Anfragesprache DDDquery, das UI leitet die Anfrage lediglich weiter und übernimmt die Visualisierung der Ergebnisse.
2. Die Benutzer formulieren ihre Anfragen selbst in DDDquery, das UI modifiziert sie jedoch, etwa in dem es Angaben zu zusätzlich zurückzulieferndem Kontext ergänzt.
3. Die DDDquery-Anfragen werden vollständig vom UI generiert, etwa indem es von Benutzern auszufüllende Formulare auswertet.

Alle drei Varianten sollen grundsätzlich unterstützt werden.

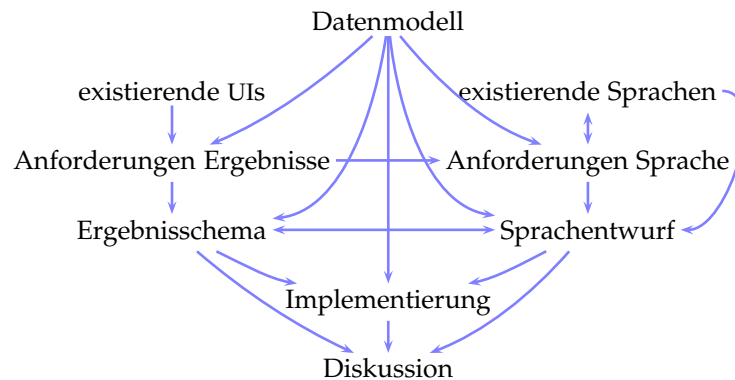


Abbildung 1.2: Struktur dieser Arbeit

1.4 Überblick über diese Arbeit

In dieser Arbeit soll zunächst, in Kapitel 2, das Datenmodell vorgestellt werden, das dem Korpus zugrunde liegt und auf das entsprechend auch Anfragesprache und Ergebnisschema dieser Arbeit aufbauen werden.

Dann gilt es, zu ermitteln, welcher Art die Ergebnisse der Anfragen sein sollen. Da die Ergebnisse den Benutzern über grafische Benutzungsschnittstellen (GUIs) präsentiert werden sollen, wird zunächst analysiert, welche Daten für übliche GUI-Präsentationsformen notwendig sind. Dafür werden in Kapitel 3 existierende Benutzungsschnittstellen für diachrone und andere Korpora untersucht und auf ihre anzeigerelevanten Elemente hin ausgewertet. In Kapitel 4 wird dann analysiert, welche Daten für diese Darstellungsformen nötig sind. Auf diese Weise werden Anforderungen an ein Ergebnisschema aufgestellt und schließlich das Ergebnisschema entwickelt. Dabei wird zum Teil auch auf die Anforderungen an die Anfragesprache aus späteren Kapiteln Bezug genommen, da sich Aspekte der Modellierung der Anfragesprache auf das Ergebnisschema auswirken.

Kapitel 5 gibt eine Übersicht über existierende linguistische Anfragesprachen. In Kapitel 6 werden die Anforderungen an die Anfragesprache zusammengestellt. Da sich zeigen wird, dass keine der existierenden Sprachen den Anforderungen genügt, wird schließlich in Kapitel 7 eine eigene Anfragesprache, *DDDquery*, entwickelt.

Kapitel 8 schließlich stellt die Implementierung des Sprachentwurfs vor, und Kapitel 9 schließt die Arbeit mit einer Zusammenfassung der Ergebnisse und einem Ausblick ab.

2 Datenmodell des Korpus

In diesem Abschnitt wird das Modell des Korpus getroffen, das den Konzepten des Anfragesystems zugrunde liegt.

2.1 Einführung

Bereits von Dipper et al. (2004) ist das grundlegende Datenmodell des DDD-Korpus skizziert worden, weiteres dazu ist bei Vitt (2004) und Faulstich (2005) zu finden

Aufgrund der frühen Projektphase stehen jedoch noch Präzisierungen und Überarbeitungen aus.

Im Folgenden soll ein Datenmodell vorgestellt werden, dass präzise genug ist, um den in Abschnitt 6 vorgestellten Anforderungen zu genügen und als Grundlage zur Definition der Anfragesprache zu dienen, jedoch allgemein genug, um noch keine Vorgaben für die endgültige Form der Speicherung zu machen und um künftigen Entwicklungen Rechnung zu tragen.

2.2 Grundlegende Elemente

Abbildung 2.1 auf der nächsten Seite zeigt die zentralen Elemente dieses Korpusdatenmodells: Das Korpus ist ein Graph aus **Knoten (Node)**. Knoten können **Elemente**, **Spans** oder **Attribute** sein. Die Kanten des Korpusgraphen ergeben sich durch die verschiedenen möglichen Relationen zwischen den Knoten.

Definition 1 (Span)

Ein Span s bezeichne einen zusammenhängenden Abschnitt eines Textes.

Ein Span ist eindeutig definiert durch ein Tripel $(tid, left, right)$, wobei $s.tid$ eine Referenz auf einen Text t , $s.left$ die Startposition des Textabschnitts innerhalb von t (d. h. die fortlaufende Nummer des ersten Zeichens von s in t) und $s.right$ die Endposition (die Nummer des letzten Zeichens von s) darin bezeichne.

Zu jedem Span s lässt sich eindeutig ein Textwert $s.content()$ bestimmen, dies ist der Inhalt des durch $s.left$ und $s.right$ gekennzeichneten Texts.

2 Datenmodell des Korpus

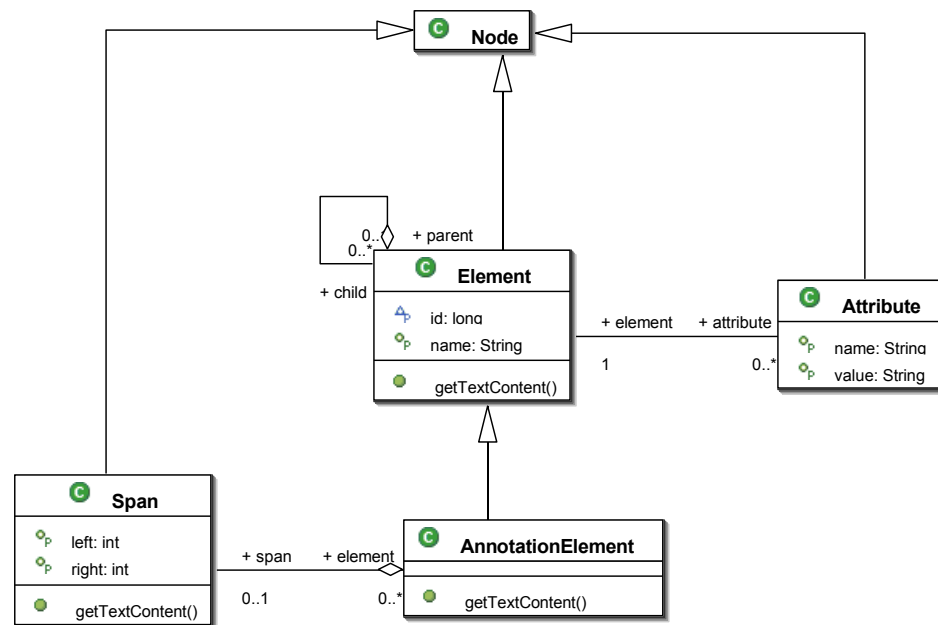


Abbildung 2.1: Grundlegende Elemente des Korpusdatenmodells

In diesem Zusammenhang sollte beachtet werden, dass Spans der Länge 0 durchaus möglich sind. Sie werden in Alignments (vgl. Abschnitt 3.4.4) benutzt, um Einfügungen bzw. Löschungen zu modellieren.

Elemente sind die anderen grundlegenden Bausteine des Korpus:

Definition 2 (Element)

Ein Element e besitzt eine eindeutige ID $e.id$ sowie einen Namen $e.name$, der den Typ des Elements (z. B. *word*) bezeichnet.

Definition 3 (Beziehungen von Elementen)

Zwei Elemente e_p und e_c können in einer Kindbeziehung stehen: e_p / e_c heißt, e_c ist Kind von e_p . Ein Element e_p ist Elternteil (parent) von e_c genau dann, wenn e_c Kind von e_p ist ($e_c \setminus e_p \iff e_p / e_c$).

Ein Element kann mehrere Eltern haben. Der durch die Kindbeziehungen zwischen Elementen induzierte Graph muss jedoch zyklensfrei und zusammenhängend sein.

Ein Element kann mit maximal einem Span ($e.span$) sowie mit einer Menge von Attributen assoziiert sein. Ein mit einem Span assoziiertes Element ist ein Annotationselement.

Zu einem Element kann der Textwert extrahiert werden. Der Textwert eines Annotationselements ist der Textwert des mit ihm assoziierten Spans.

Definition 4 (Attribut)

Ein Attribut besteht aus einem Namen und einem Wert und ist immer genau einem Element zugeordnet.

Definition 5 (Knoten)

Jedes Element, jeder Span und jedes Attribut ist ein Knoten des Korpusgraphen.

Die möglichen Namen und Werte der Elemente und Attribute sowie die konkreten Formen, welche Elemente welche Kinder und welche Attribute tragen können, werden von einem Korpuschema festgelegt, das im Rahmen des DDD-Projekts entwickelt werden wird.

Abbildung 2.2 auf der nächsten Seite zeigt diese einfachen Datentypen im Kontext des gesamten Korpusdatenmodells. In diesem Zusammenhang ist zu beachten, dass *alle* im Folgenden neu vorgestellten Objekte spezielle Elemente sind (und damit etwa Attribute tragen können).

Ein **Korpus** ist eine Menge von Dokumenten. Ein **Dokument** fasst eine Reihe zusammengehöriger Konzepte zusammen:

Der **Header** enthält zahlreiche bibliographische Informationen in Form einer Elementstruktur (die vom Projekt noch festzulegen ist und sich voraussichtlich an den entsprechenden Vorschlägen der Text Encoding Initiative TEI (TEI-P5) orientieren wird).

Texte repräsentieren den eigentlichen Text des Dokuments. Mehrere Texte können vorhanden sein, da es verschiedene Fassungen der Digitalisierung eines und desselben Textes geben kann – beispielsweise eine diplomatische Fassung, die eng an der Form der Vorlage orientiert ist, und eine normalisierte Fassung, die z. B. die Schreibweise von Wörtern vereinheitlicht (vgl. Abbildung 2.3 auf Seite 19). Eine Fassung ist jedoch nie auf mehrere Texte verteilt. Spans sind immer mit einem konkreten Text assoziiert und beschreiben einen Abschnitt daraus.

Mit einem Text kann eine Menge von **Annotationsebenen (Layer)** assoziiert sein. Jede Annotationsebene bildet eine hierarchische (optional: DAG-förmige, vgl. Dipper et al., 2004) Struktur von **Annotationselementen**. Jedes Annotationselement hat wie auch allgemeine Elemente einen **Typ** (z. B. »word«), eine ID sowie eine Menge von **Attributen**. Zusätzlich kann (und wird) jedem Annotationselement bis zu ein **Span** zugeordnet sein, der den annotierten Textabschnitt bezeichnet. Jedem Span kann man mithilfe dieser Daten einen Textwert zuordnen. Man beachte aber, dass

2 Datenmodell des Korpus

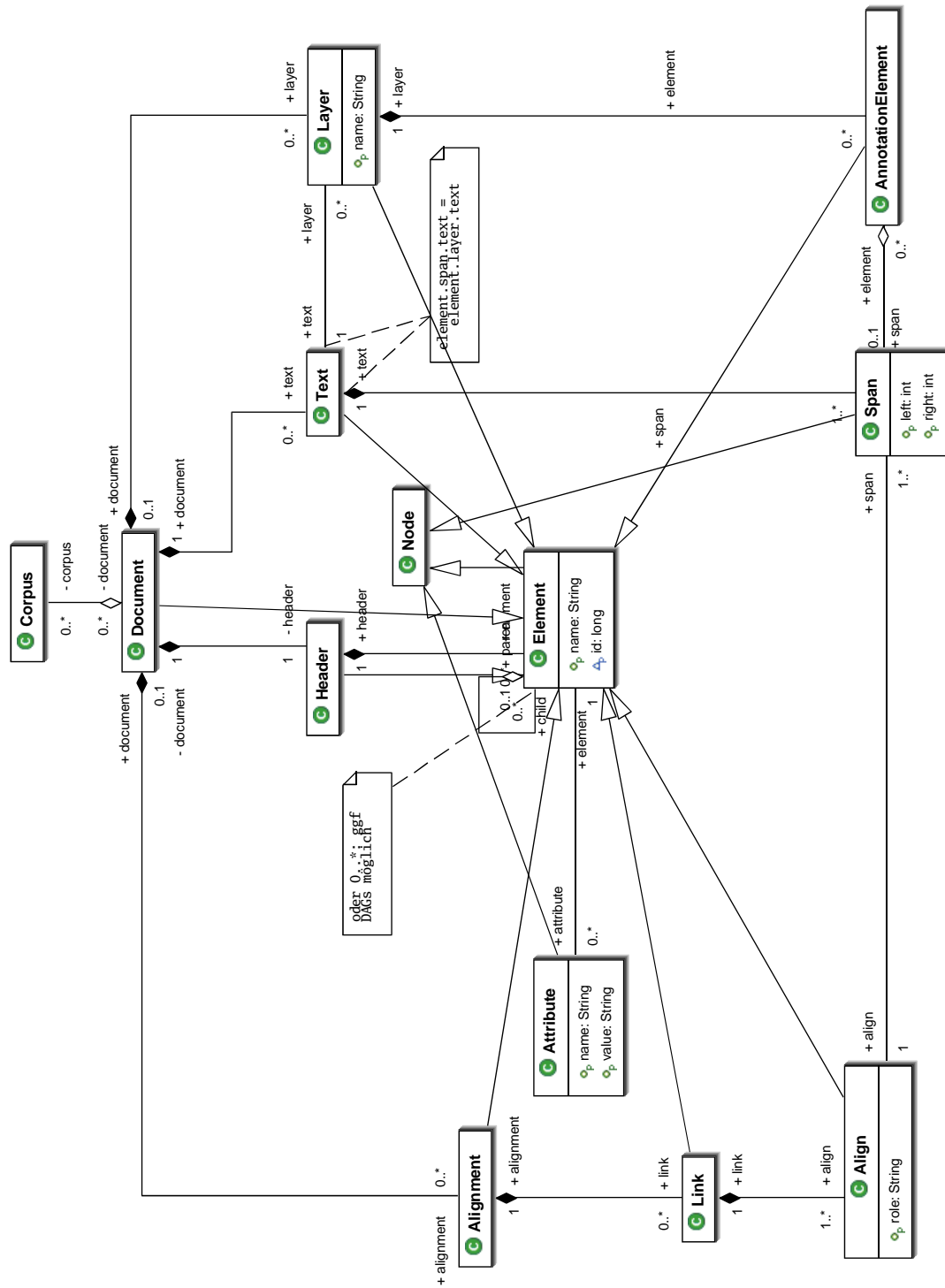


Abbildung 2.2: Datenmodell des Korpusgraphen (Entwurf)

. Eirif	
fazun	
idifi fazun	
hera duoder fuma	
hapt heptidun	Eiris sazun idisi sazun hera duoder suma
fumaherilezidun	hapt heptidun suma heri lezidun suma clu
fuma	bodun umbi cuoniouuidi insprinc hapt
clu	bandun inuar uigandun.
bodun umbicuonio uuidi	(b)
insprinc hapt	
bandun inuarugandun	
.H.	
(a)	

Abbildung 2.3: Zwei Umschriften desselben Texts (eines Abschnitts der Merseburger Zaubersprüche)

einem Span auch innerhalb einer Annotationsebene mehrere Elemente zugeordnet sein können!

Alignments können Spans auch aus unterschiedlichen Texten miteinander verknüpfen. Die Alignments liegen dabei in einer Form vor, in der jeweils zwei »Seiten« miteinander verknüpft werden (bei denen es sich auch um denselben Text handeln kann!). Die »Seiten« werden durch *Rollen* identifiziert. Detaillierter werden Alignments in den Abschnitten 3.4.4 auf Seite 42 und 7.4.9 auf Seite 102 beschrieben.

Neben diesen Inhalten, die im Korpus vorhanden sind und lediglich selektiert werden müssen, sind noch **generierte Inhalte** möglich, die etwa von Aggregationsfunktionen erzeugt werden können.

Das gesamte Korpus mit allen oben geschilderten Objekten wird als **Korpusgraph** aufgefasst.

2.3 Alignments

Alignments dienen dazu, verschiedene Textabschnitte (aus demselben oder aus unterschiedlichen Texten) miteinander in Beziehung zu setzen. Um Alignments (nach Faulstich, 2005) zu modellieren, wurden eine Reihe von Klassen eingeführt, die in Abbildung 2.4 dargestellt und im Folgenden anhand des Beispiels aus Abbildung 2.5 – der englischen und deutschen Fassung des Satzes »Er schloss die Tür ab« – erläutert werden sollen:

2 Datenmodell des Korpus

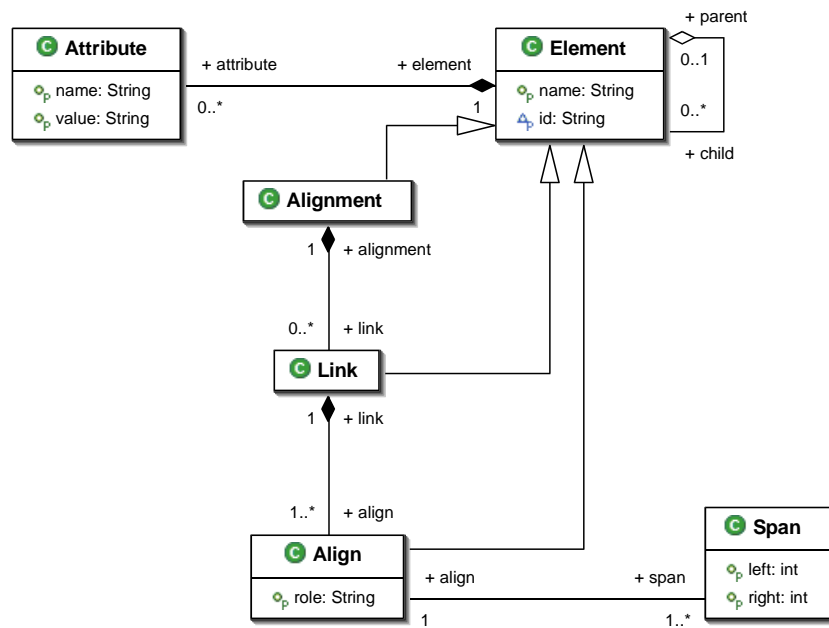


Abbildung 2.4: Datenmodell für Alignments (Ausschnitt)

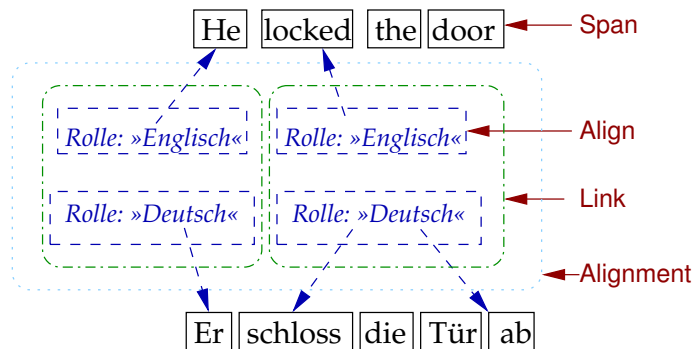


Abbildung 2.5: Elemente eines Alignments am Beispiel (Ausschnitt)

Die kleinste an einem Alignment beteiligte Einheit ist ein *Span* (vgl. Definition 1 auf Seite 15). Dies ist, wie im Rest des Korpusdatenmodells auch, ein zusammenhängender Ausschnitt eines Texts – im Beispiel etwa der Abschnitt des deutschen Texts, der das Wort »ab« enthält.

Definition 6 (Link)

Ein Link fasst eine Verbindung innerhalb des Alignments zusammen.

So ist beispielsweise die Verbindung zwischen dem englischen »He« und dem deutschen »Er« ein Link. Die Beteiligten an einem Link sind jedoch nicht einfach Spans, sondern *Align-Objekte*.

Definition 7 (Align-Objekt)

Ein Align-Objekt repräsentiert eine Seite eines Links innerhalb eines Alignments. Jedes Align-Objekt wird durch eine Rolle charakterisiert und kann mit mehreren Spans assoziiert sein.

Auf diese Weise lässt sich etwa »locked« mit den nicht zusammenhängenden Spans »schloss« und »ab« alignieren, und auch Auslassungen bzw. Einfügungen sind modellierbar (vgl. Faulstich, 2005).

Definition 8 (Alignment)

Ein Alignment schließlich ist eine logisch zusammengehörige Sammlung von Links.

Die an einem Alignment beteiligten Objekte – Align-Objekte, Links und Alignments – sind ihrerseits *Elemente* und können so Attribute tragen.

2.3.1 Greediness

Zur Navigation entlang Alignments sind auch alignierte Subspans zu beachten. Wie Faulstich (2005) gezeigt hat, sind dabei insbesondere Situationen problematisch, in denen 1. Einfügungen oder 2. Ersetzungen genau an den Rändern des Ausgangsspanns auftreten:

Sei s_0 ein Span in Text t . Es soll ein zugehöriger alignierter Text s'_0 im Text t' gefunden werden. Für die Alignierungsoperation sind nun auch Alignments mit Teilspans von s_0 zu berücksichtigen.

Problematisch sind nun die Fälle:

1. Ein Insert-Alignment (mit einem Quellspan der Länge 0) tritt so auf, dass sein Quellspan s_1 genau an einem Rand von s_0 liegt.

2 Datenmodell des Korpus

2. Ein Replacement-Alignment (mit einem Quellspan s_1 einer Länge größer 0) tritt so auf, dass s_1 einen Rand von s_0 enthält, jedoch nicht vollständig in s_0 enthalten ist.

In beiden Fällen muss vom Anwender unterschieden werden, ob an der betreffenden Stelle die größere oder die kleinere Variante des Ziels gewünscht ist, d. h. ob der Alignment-Operator am betreffenden Rand *greedy* oder *non-greedy* agieren soll.

Deshalb werden die Alignment-Operatoren mit einem Greediness-Flag für den entsprechenden Rand versehen.

2.3.2 Relationen zwischen Alignments

Da Alignments eine Doppelrolle spielen – einerseits bilden sie eine Verbindung zwischen Spans mit eigener Semantik, andererseits bestehen sie selbst aus Elementen, die Attribute tragen können – bedürfen sie einer besonderen Behandlung.

Um eine möglichst generische Anfrage zu ermöglichen, soll es deshalb möglich sein, zu einem Span die über eine bestimmte Rolle zugängliche Struktur aus Alignment-Elementen in Relation zu setzen, so dass dort weitere Beschränkungen anhand von Attributen stattfinden können.

Zusätzlich soll ein Shortcut zur direkten Anfrage von alignierten Textabschnitten über ein gegebenes Rollenpaar zur Verfügung stehen.

Schließlich müssen alle Navigationsmöglichkeiten die verschiedenen Greediness-Varianten zur Option stellen.

2.3.3 Alignments von Annotationselementen

Alignierungen sind zwischen beliebigen Spans möglich, werden in der Praxis jedoch oft zwischen ganz bestimmten Annotationselementen bzw. den damit assoziierten Spans erfolgen – etwa um in zwei Versionen eines Textes Wörter (auf der Grundlage einer jeweils passenden Wortannotation) miteinander zu alignieren. Entsprechend ist es in CDM2 (Faulstich, 2005) auch möglich, innerhalb der *Primary Units* einer Textebene Elemente mit alignierten Textspans zu haben – und die sollen sinnvollerweise ausgenutzt werden.

Es ist vielleicht sinnvoll, Alignments zwischen Annotationselementen (und daraus folgend den zugrundeliegenden Spans) und Alignments zwischen nicht mit Annotationselementen verbundenen Spans zu unterscheiden; allerdings führt dies zu einer weiteren Verkomplizierung der Sprache und des Modells. Wenn in der Praxis Anfragen wie etwa

```
... word [@pos='ADJ'] / aligned(de,lat)::word [@pos='ADJ']
```

– also mit einer Angabe des Zielelementtyps und mit nicht ineinander schachtelbaren Zielelementtypen – die Regel sind, sollten hier auch die Wandlung des Spans, der das Ergebnis der aligned-Achse ist, in das Element eindeutig sein.

Alternativ ist es auch möglich, das Datenmodell dahingehend zu ändern, dass nicht nur Spans, sondern auch Elemente miteinander aligniert werden können.

2 Datenmodell des Korpus

3 Benutzungsschnittstellen

In diesem Kapitel werden existierende Benutzungsschnittstellen (*User Interfaces, UI*) zu linguistischen Korpora vorgestellt und untersucht. Diese Untersuchung bildet die Basis für die in Abschnitt 4.2 vorgestellten Anforderungen der potentiellen *DDDquery*-Benutzungsschnittstellen an die vom Anfragesystem zu liefernden Daten.

3.1 Webschnittstellen

Kroymann et al. (2004) haben historische und diachrone Textkorpora sowohl des Deutschen als auch anderer Sprachen untersucht. Dieser Abschnitt stellt Erkenntnisse für die Anforderungen bezüglich der Such- und Anfrageoptionen sowie der Benutzungsschnittstellen der untersuchten Korpora zusammen. Korpora, die keine besonderen Suchfeatures etc. aufweisen, werden hier nicht betrachtet. Für die nicht UI-relevanten Features sei auf Kroymann et al. (2004) verwiesen.

3.1.1 Digitales Mittelhochdeutsches Textarchiv

Das Digitale Mittelhochdeutsche Textarchiv¹ wird in Kroymann et al. (2004, Abschnitt 2.4) beschrieben.

Eine Suche via Webformular ist in Abbildung 3.1 auf Seite 27 dargestellt.

Dabei kann nach Wortformen² (mit Wildcards) über einen oder mehrere Texte gesucht werden. Als Ergebnis wird eine Auswahlliste der gefundenen Wortformen gezeigt (a), nach Wahl einer Wortform bekommt man die Konkordanzliste in KWIC-Form³ (b). Aus dieser Liste kann man in die

1 <http://www.mhgta.uni-trier.de/Demo>

2 Die Suche nach Wortformen mit Sonderzeichen wie dem geschweiften *z* funktioniert jedoch anscheinend nicht, in der Wortformenliste wird offenbar eine normalisierte Form (hier *baz*) dargestellt.

3 KWIC steht für *Keyword in Context*, eine in der Sprachwissenschaft übliche Darstellung für Wörter in ihrer Umgebung. Dabei werden die relevanten Schlüsselwörter (z. B. die Ergebnisse einer Suche) jeweils zusammen mit etwa einer Zeile ihrer Umgebung

3 Benutzungsschnittstellen

Volltextdarstellung springen, in der die Zeile des Treffers (nicht jedoch der Treffer selbst) dann hervorgehoben wird (c). Als Besonderheit ist in der Volltextdarstellung – allerdings erst bei einem der Texte – jedes Wort mit dem entsprechenden Eintrag im Mittelhochdeutschen Handwörterbuch⁴ verlinkt.

Die Volltextdarstellung erfolgt seitenweise (auf die Original-Seiten bezogen), zusätzlich wird zur Navigation ein Inhaltsverzeichnis eingeblendet (links in Abbildung 3.1c).

Nach Kroymann et al. (2004) kann in der kommerziell vertriebenen CD-ROM-Version auch nach grammatischen Angaben, Belegstellen und Siglen gesucht werden.

3.1.2 Uni Bonn

Das Bonner Frühneuhochdeutsch-Korpus⁵ (Kroymann et al., 2004, 2.1) verfügt über keine Suchmöglichkeiten.

Auf der gleichen Seite gibt es ein Kant-Korpus mit Suchmöglichkeiten nach Wort in bestimmtem Kontext (über ein Formular), Anzeige KWIC (nur Suchwort, nicht Kontext hervorgehoben) mit Positionsangabe und Link auf den Volltext.

3.1.3 Thesaurus Indogermanischer Text- und Sprachmaterialien (TITUS)

Zu TITUS⁶ gibt es zwei Formen des Zugangs: Über ein Webinterface oder via WordCruncher (für letzteres siehe 3.2.3 auf Seite 39).

Über das Webformular kann man nach Wortformen (auch mit Wildcard) innerhalb einer Sprache oder einer Sprachvariante suchen. Auch hier gibt es wieder einen mehrstufigen Suchprozess: 1. Wahl der Sprache oder Sprachvariante, 2. Wortformeingabe – das Ergebnis ist dann eine Wortliste mit oder ohne Kontext oder eine Dropdownbox mit den Wörtern (*word wheel*), 3. von der Wortliste kommt man zur Volltextanzeige. In der Textanzeige (Volltext wie Kontext) ist jedes Wort klickbar und löst dann eine erneute Suche nach diesem Wort aus.

im Text dargestellt, üblicherweise so, dass die *Keywords* direkt untereinander und hervorgehoben dargestellt werden.

4 <http://germazope.uni-trier.de/Projects/WBB/woerterbuecher/lexer>

5 <http://ikp.uni-bonn.de/dt/forsch/fnhd>

6 <http://titus.uni-frankfurt.de/>

3.1 Webschnittstellen

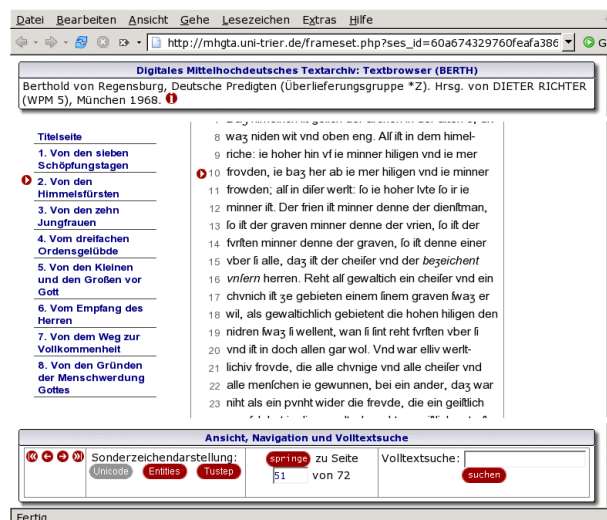
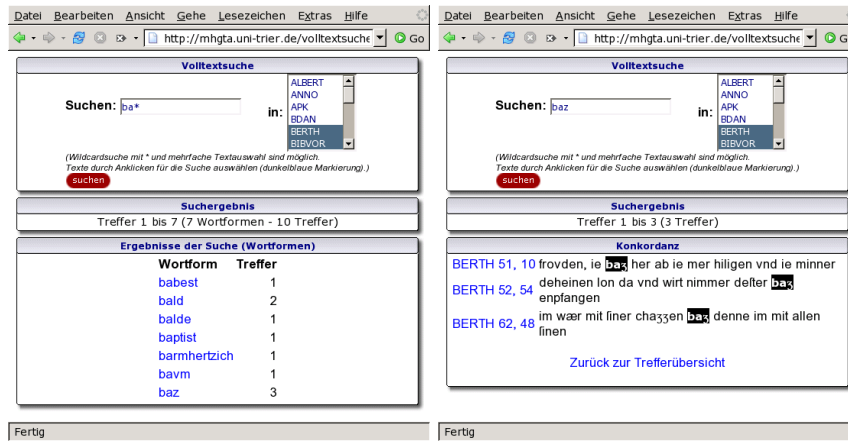


Abbildung 3.1: Volltextsuche im Digitalen Mittelhochdeutschen Textarchiv, vgl. Abschnitt 3.1.1 auf Seite 25.

3 Benutzungsschnittstellen

Für manche Texte ist die Anzeige der physischen und logischen Struktur realisiert, wie in Abbildung 3.2 am Beispiel der Merseburger Zaubersprüche dargestellt.

Line of ms.: 1	Verse: 1	Eiris sazun idisi sazun hera duoder	Verse: 2	suma
Line of ms.: 2		hapt heptidun suma heri lezidun	Verse: 3	suma clu
Line of ms.: 3		bodun umbi cuoniouuidi	Verse: 4	insprinc hapt
Line of ms.: 4		bandun inuar uigandun.		

Abbildung 3.2: Physische und logische Struktur in TITUS' Webschnittstelle.

3.1.4 Codices Electronici Ecclesiae Coloniensis (CEEC)

In den CEEC⁷ (vgl. Kroymann et al., 2004, 2.5) ist nur eine Suche über die (sehr ausführlichen) Katalogdaten möglich, die eigentlichen Texte werden als Scans zur Verfügung gestellt. Die Suche ist nach Schlüsselwort oder Schlüsselwortpräfix in diversen Katalogfeldern möglich, alternativ kann aus der Liste aller Wörter des gewählten Index gesucht werden. Als Ergebnis wird der Kontext des Treffers ohne Hervorhebung des Treffers selbst ausgegeben, man kann den Kontextbereich dabei erweitern (also mehr Kataloginhalt anzeigen). Es gibt Links zu den betreffenden Scans, soweit bereits erfasst.

3.1.5 Mittelhochdeutsche Begriffsdatenbank

Die Mittelhochdeutsche Begriffsdatenbank (MHDBDB⁸, vgl. Kroymann et al., 2004, 2.7) an der Universität Salzburg enthält recht umfangreiche (und dokumentierte⁹) Suchmöglichkeiten über ein Webinterface.

Anfragesprache

Die Anfragesprache bietet Möglichkeiten zur Suche nach Lemmata und darauf beruhenden Wortformen, exakten Varianten, Zeichensetzung, nach Wildcards (*, ?) an beliebiger Position im Suchstring, der Wortposition innerhalb der Textzeile, nach grammatischen Kategorien sowie nach Bedeutungskategorien (aus dem zugehörigen Thesaurus). Einfache Suchen können mit den booleschen Operatoren *und* und *oder* kombiniert werden. Über *Reihenabfragen* können Folgen von Suchbegriffen abgefragt werden.

⁷ <http://www.ceec.uni-koeln.de>

⁸ <http://mhdbdb.sbg.ac.at>

⁹ <http://mhdbdb.sbg.ac.at:8000/help/general.de.html>

Über *Kontextabfragen* kann nach gemeinsamen Vorkommen innerhalb eines (in Zeilen oder Worten konfigurierbaren) Kontextbereichs gesucht werden.

Die einfachsten Fälle der Sprache – nur ein Wort, ggf. mit Wildcards – suchen jeweils nach genauen Vorkommen des Suchworts sowie nach Wortformen mit demselben Lemma.

Webinterface

Der umfassendste Zugang zum Korpus der MHDBDB erfolgt über die Textanalyse-Maske, wie in Abbildung 3.3 auf der nächsten Seite dargestellt.

In einem Suchfeld können Anfragen in der oben beschriebenen Sprache eingegeben werden (a). Die übrigen Formularelemente dienen dazu, diese Anfrage weiter zu parametrisieren (für die Kontext- und Reihensuche), die zu durchsuchenden Texte einzugrenzen (auch nach Kategorien oder Autoren) und die Ergebnisdarstellung zu beeinflussen.

Die Ergebnisse der Suche werden in einer *Frequenztabelle* dargestellt. In deren Zeilen sind die Wortformen aufgeführt, in den Spalten die Texte. Für Lemmata, Textgruppen sowie jeweils für alle Texte und alle Wortformen sind Zusammenfassungsspalten und -zeilen vorhanden.

Ein Klick auf einen Frequenzwert führt in eine Maske zur Auswahl der Textbelege (b), aus der man die gewünschten Belegstellen aussuchen und dann mit mehr Kontext anzeigen lassen kann (c). Bei nur wenigen Treffern wird die Textbelegauswahl (b) übersprungen.

Ein Klick auf einen Lemmanamen (a) führt auf die zugehörige Wörterbuchseite (d), in der Informationen zu den Wortformen, Komposita, Häufigkeiten, Wortart sowie die Kategorisierung im Thesaurus (unten im Bild) dargestellt sind.

3.1.6 Lancaster Newsbook Corpus

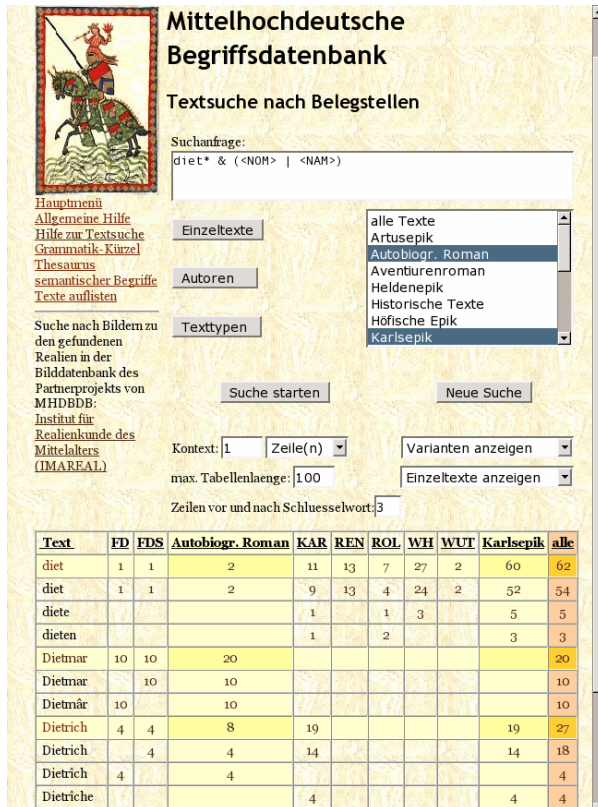
Das Lancaster Newsbook Corpus¹⁰ (Kroymann et al., 2004, 3.2.1) ist ein Korpus mit *Newsbook*-Texten: Newsbooks sind Zeitungsvorläufer, das Korpus verfolgt das Ziel, die Wiederverwendung von Text in verschiedenen Newsbook-Ausgaben quantitativ zu untersuchen.

Das Korpus ist auf CD-ROM erhältlich. Auf der Projektwebsite sind einige Screenshots¹¹ der Benutzungsschnittstelle des Projekts zu sehen. Für DDD ist vor allem die Anzeige der Alignments zwischen zwei Texten

¹⁰ <http://www.ling.lancs.ac.uk/newsbooks>

¹¹ <http://bowland-files.lancs.ac.uk/newsbooks/data.htm>

3 Benutzungsschnittstellen



Mittelhochdeutsche Begriffsdatenbank

Textsuche nach Belegstellen

Suchanfrage: diet* & (<NOM> | <NAM>)

Einzeltexte | Autoren | Texttypen

Suche starten | Neue Suche

Kontext: 1 | Zeile(n) | Varianten anzeigen | Einzeltexte anzeigen

max. Tabellenlänge: 100 | Zeilen vor und nach Schlüsselwort: 3

Text	FD	FDS	Autobiogr. Roman	KAR	REN	ROL	WH	WUT	Karlsepi	alle
diet	1	1	2	11	13	7	27	2	60	62
diet	1	1	2	9	13	4	24	2	52	54
diete				1		1	3		5	5
dieten				1		2			3	3
Dietmar	10	10	20							20
Dietmar		10	10							10
Dietmâr	10		10							10
Dietrich	4	4	8	19					19	27
Dietrich		4	4	14					14	18
Dietrich	4		4							4
Dietriche				4					4	4

(a) Suchergebnisse als Frequenztafel



Mittelhochdeutsche Begriffsdatenbank

Auswahl von Textbelegen

Alle anzeigen | Auswahl anzeigen | Auswahl löschen

☐ Karl der Grosse, Zeile 966 mit einer kreftigen diet,

☒ Karl der Grosse, Zeile 1111 daz er mit siner diete

☐ Karl der Grosse, Zeile 1512 nu schuof diu kristene diet

☒ Karl der Grosse, Zeile 2470 in landen unde in dieten,

☐ Karl der Grosse, Zeile 2482 unde in der verworhten diet

☐ Karl der Grosse, Zeile 3851 diz ist diu saeligste diet,

☐ Karl der Grosse, Zeile 7642 dô stach diu heidenischiu diet

☐ Karl der Grosse, Zeile 989 daz hiez er und er leit diet

(b) Auswahl der anzuzeigenden Textstellen



MHDBDB - Anzeige von Textbelegen

Karl der Grosse, Zeile 1108 - 1114

1108 hin wider welle riten;

1109 swenne er und die sîne

1110 hin kumen zuo dem Rine,

1111 daz er mit siner diete

1112 einen hof dar gebiete

1113 an sînen stuol ze Âche.

1114 ân alle widersprache

Karl der Grosse, Zeile 2467 - 2473

2467 er sprach: ob ir diz begât,

2468 daz ir hie gesprochen hât

(c) Anzeige der gewählten Belegstellen



Mittelhochdeutsche Begriffsdatenbank

diet

☒ Count Frequencies [<= | = >]

diet 496 NOM

diete 68 NOM

dieten 8 NOM

dieth 0 NOM

dieth 4 NOM

576

Compounds

hovediet

hovediet, hovediete

lantdiet

lantdiet, lantdieth

undiet

undiet

1 Gesellschaftl Leben/Allg. Erscheinungsformen 23110000

2 Mensch 20000000

Mensch/Körperliches Wesen 21000000

(d) Wörterbuchseite

Abbildung 3.3: Die Webschnittstelle der Mittelhochdeutschen Begriffsdatenbank, vgl. Abschnitt 3.1.5 auf Seite 28.

relevant. Sie erfolgt hier in Form einer dreispaltigen Tabelle: in der ersten Spalte stehen statistische Maße über die Ähnlichkeit der beiden Sätze einer Zeile, in den anderen beiden sind die jeweiligen Texte dargestellt, die Unterschiede darin sind farbig gekennzeichnet (vgl. Abbildung 3.4).

Similarity Score	Suggested derived Sentence	Suggested Source Sentence(s)
PSNG=0 WS=0		
PSD=0.954 PS=0.874 PSNG=1.0 WS=0.955	27) We are here very apt to believe a peace, because it is a thing exceedingly desired all over these Lands.	77) stilo novo. </h3> <p>We are here very apt to believe a peace, because it is a thing exceedingly desired all over these Lands.
PD=0.948 D=0.201 PVS=0.793 WS=0.896	28) And there is the more hope, because all things are very still and <reg orig="calme">calm</reg>, both as sea and land.	78) And there is the more hope, because all things are very still and 17) On a sudden, about Ten of the clock in the forenoon, this Iron Chest (which for many years hath been
PSD=1.0 PS=1.0	29) And we have added to all our conceits the like out of	79) And we have added to all our conceits the like out of

In this table, the numbers assigned to sentences denote locations of the sentences in the suggested derived and source texts. The n-grams shared by each pair of aligned sentences are highlighted with green, and shared single words (including those undergone inflectional and letter-case changes; function words are excluded) are highlighted with red colour.

Abbildung 3.4: Alignments im Lancaster Newsbook Corpus (vgl. Abschnitt 3.1.6 auf Seite 29). Quelle: Website dieses Korpus.

3.1.7 Corpus of Middle English Prose and Verse

Das *Corpus of Middle English Prose and Verse*¹² (vgl. Kroymann et al., 2004, 3.2.5) bietet eine einfache, boolesche oder Nähesuche. Die einfache Suche sucht nach Wortformen, die boolesche Suche läßt Wortformsuchen nach der Form

$$(\alpha \Theta_1 \beta) \Theta_2 \gamma \text{ within the same } \Sigma$$

mit Wortformen α, β, γ , den Booleschen Operatoren $\Theta_i \in \{\text{And, Or, Not}\}$ und einer Bereichsbeschränkung Σ mit den möglichen Werten *paragraph or line, paragraph, line* und *entire work* zu. Mit der Nähesuche kann man Anfragen der Form

$$(\alpha \Theta_1 (\text{within } i \text{ characters}) \beta) \Theta_2 (\text{within } i \text{ characters}) \gamma$$

mit Θ_i aus *Near, Not near, Followed By, Not followed by* formulieren.

¹² <http://hti.umich.edu/c/cme>

Alle Anfragen lassen sich auf einzelne Werke des Korpus beschränken, Wildcards sind nicht möglich. Die Ergebnisseite zeigt den Kontext und Links zum Abschnitt des Treffers, hebt jedoch den genauen Treffer nicht hervor.

3.1.8 Corpus del Español

Das Corpus del Español¹³ ist ein diachrones Corpus des Spanischen, das angibt, umfangreichere Suchfunktionen zu haben als die meisten anderen großen Korpora. Interessanterweise basiert das Korpus auf relationalen Datenbanken.

Zur Suche gibt es eine Anfragesprache. Im einfachsten Falle wird nach Wörtern gesucht, es gibt jedoch auch Möglichkeiten, um nach allen Formen eines Lemmas, nach Kollokationen (aus bis zu drei oder vier Wörtern je nach Zeitraum), nach bestimmten grammatischen Formen oder nach Wörtern mit Wildcards (_ und * an beliebiger Stelle) zu suchen. Darüberhinaus sind einfache boolesche Anfragen und das Erstellen eigener Wortlisten möglich.

Die Suchanfragen können über Zeiträume gefiltert werden, sodass man z. B. nach allen Verben, die nicht im 17. Jahrhundert, jedoch mehr als fünfmal im 18. Jahrhundert belegt sind, suchen kann.

Abbildung 3.5 auf der nächsten Seite zeigt einen typischen Suchlauf: Nachdem man im oberen Frame die Suchkriterien spezifiziert hat, erhält man eine Frequenztabelle (mittlerer Frame). Aus der kann man nach diversen Kriterien auswählen, für welche Treffer der Kontext im unteren Frame angezeigt wird. Die KWIC-Darstellung kann nach diversen Kriterien sortiert werden, in einem Popup-Fenster kann man auch mehr Kontext zu einem Treffer anzeigen lassen.

3.1.9 Annotated Corpora of Text (ACT)

Dieses Korpus tschechischer Handschriften¹⁴ bietet nach Kroymann et al. (2004, 3.6.1) zahlreiche Annotationsebenen und ein Web- sowie ein Java-tool, um auf diesen Ebenen zu suchen.

Über einen Katalog kann in den (wenigen) Metadaten der Texte gesucht werden. In den ausgewählten Texten kann dann mit einem Webformular nach zahlreichen Kriterien gesucht werden. Die Ergebnisse werden tabellarisch (Wörter mit deren Eigenschaften) dargestellt (vgl. Abbildung 3.6),

¹³ <http://www.corpusdelespagnol.org>

¹⁴ <http://prometheus.ms.mff.cuni.cz/act/>

3.1 Webschnittstellen

Mark Davies
BYU / ISU / NEH

CORPUS DEL ESPAÑOL

Usar español

Introduction / Tour

SEARCH: *.v_inf *
ACCENTS: á é í ó ú ü ñ

SORT BY: 1900s-Lit
LIMITS: s 1800s>5
INDEX: A
#: 50
GROUP BY: FORMS
SHOW: HITS
VERIFY: NO
RESET: SUBMIT

SEE CONTEXT	WORDS / PHRASES	1200s	1300s	1400s	1500s	1600s	1700s	1800s	Total	Oral	Lit	Text
1	ATARDECER				5			12	341	22	299	20
2	COMPARTIR	7	3	1	2	4		97	415	130	189	96
3	ESTALLAR			2	3	2		172	177	22	120	35
4	ASUMIR				2			12	399	163	91	145
5	REACCIONAR				1			20	177	51	85	41
6	FUNCIONAR							73	358	149	81	128
7	INGRESAR							89	218	76	55	87
8	INVITAR				1	1		33	142	75	45	22
9	ELIMINAR							19	417	105	45	267
10	TRANSCURRIR							26	67	9	44	14

<< INSTRUCTIONS

For expanded context, click on the number sign in the leftmost column. To re-sort, click on the desired column.

KEYWORDS IN CONTEXT

1/2	SIGLO	TEXT	RE-SORT BY: L2 L1	C	R1 R2
1	15	Predicación del Ev...	sea por hacer alarde de excesiva entereza, ya por	asumir	el cargo de reformadores y c
2	15	Tratado sobre los ...	podría muy bien suceder que alguien prefiriera	asumir	ese trabajo debido a la neces
3	18	Ensayo. Selección	insistemáticamente por su virtud intrínseca , hasta	asumir	de hecho la representación q
4	18	El médico rural	desde el primer momento, habíase visto forzada a	asumir	la alta dirección de los negoci
5	18	Jarrapellejos	mi prima data de tres meses, pronto me hallarían a	asumir	responsabilidades ; si es de
6	18	La fórmula de la A	tanto al olvido los llamados por su saber a	asumir	la representación de los inte
7	18	Mol	http://www.corpusdelespanol.org - Mozilla Firefox		sanitaria y gener
8	18	Una			gráfica. -A mi de
9	18	Car			losales de una n

NOTE: TITLE: Obra de Agricultura AUTHOR: Herrera, Gabriel Alonso de. (1470-1539) DATE: 1504 SEE ORIGINAL SOURCE
muy viejas hácense muy bravas, y porque las pueras paren dos veces al año, que cuatro meses andan preñadas y dos dan leche(1171). Es
bien sabérselos [compartir] los tiempos del emparejarse como entrambas vengán a ser buenas. Verdad es que todavía lleva grande ventaja la
cría que nasce por mayo, o

<http://www.cervantesvirtual.com/FichaObra.html?Ref=374>

Abbildung 3.5: Webschnittstelle des Corpus del Español.
Oben Eingabefelder zur Spezifikation der Suchkriterien, in der Mitte Frequenztabellen der Ergebnisse, unten Belegstellen in KWIC-Darstellung. Das Fenster im vordergrund zeigt einen Volltextausschnitt. Nähere Erläuterung in Abschnitt 3.1.8.

3 Benutzungsschnittstellen

The screenshot displays the ACT web search interface. On the left, the 'Search criteria' panel includes options for 'Rendered form (like)' (checked, 14%), 'Use equivalences (show all)' (unchecked, 14%), 'Lemma', 'Part of speech', 'Index', 'Position (lower bound)', 'Position (upper bound)', and 'Morphology (is a wildcard)' (unchecked). A 'Reset' button is at the bottom. Below this, a message states 'Search is restricted to the following documents: Bon'. On the right, the 'Order by' panel shows 'order' as 'ascending' and 'priority' as 'first'. The 'ACT - form detail' window is open, showing a table of properties for the first result: 'Rendered form' (ЛѢГЪ), 'Colocation left' (5), 'Colocation right' (5), 'Position in document' (2772), 'Page' (2), 'Position on page' (x), 'Row number' (147), 'Position in row' (13), 'Original form' (ЛѢГЪ), and 'Document' (Bolonsky zaitar). Below the search criteria is a table of results with columns: 'Rendered form', 'First occurrence', 'Lemma', 'Index', 'Part of speech', 'Morphology', '#', 'Detail', and 'Context'. The table lists four results from 'Bolonsky zaitar' (2772, 974, 2778, 208). At the bottom, the 'ACT - context' window is open, showing a KWIC-style context for the first result: 'ЛѢГЪ НАДЛѢКАНА ЯКО ВЕРЮЮ БО ТАКО И ЛѢГЪ' (left part) and 'ЯЕ НАДЛН ЯКО НЕПЛОДНН [ВЪ] ЛѢЖНН ДННДОЛЪ БО ЖНОЕД' (right part), with the form 'ЛѢГЪ' in the middle.

Abbildung 3.6: Websuche in ACT mit Ergebnistabelle sowie Kontext- und Detaildarstellung für den ersten Treffer.

zu jedem Treffer kann zudem je ein Fenster mit Detail- und Kontextinformationen (KWIC-Darstellung) eingeblendet werden.

3.1.10 IMS Corpus Workbench / CQP

Die Corpus Workbench¹⁵ des Instituts für Maschinelle Sprachverarbeitung (IMS) der Universität Stuttgart bietet Suchen auf Basis der Anfragesprache CQP. Zur Evaluation wurde die Online-CQP-Demo¹⁶ betrachtet.

Die Webschnittstelle verwendet ein dreiteiliges Interface: Im oberen Frame können Suchanfragen in CQP (oder einfache Stringsuchen, wenn man *Simple Mode* wählt) und Parameter derselben eingegeben werden, der mittlere Frame dient der Anzeige der Konkordanzen in einer KWIC-Darstellung, und im unteren Frame kann zu einer Belegstelle ausführlicherer Kontext angezeigt werden.

Diese Webschnittstelle bietet die Anzeige von Annotationen: Part-of-Speech-Tags können auf Wunsch in verschiedenen Darstellungen angezeigt werden, und auch die Phrasenstruktur ist in verschiedenen Varianten (geklammert, mit Labels, Farbkodierung oder mit dem Kopf der

¹⁵ <http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/>

¹⁶ <http://www.ims.uni-stuttgart.de/projekte/CQPDemos/cqpdemo.html>

3.1 Webschnittstellen

Home - CQP Mode - Simple Mode - Tools - Help Page Corpus: DICKENS

Query: "confuse, *" ([pos="IN"] | [pos="PP"]); sort = unsorted Reset Form

Run Query Distribution Frequencies

Display: tokens = word phrases = off context = sentence

Keyword: ☐ nearest ☐ [] within 5 words from match ☐ inclusive

Navigation: 1 - 20 Go [36 matches] tokens = word/POS phrases = colour + head context = sentence Apply

1. David Copperfield, Chapter 10
context Emly/NP was/VBD **confused/VBN** [by [by by/IN] [observe out/PP\$ all/RB observing/VBG [her her/PP]] ,] and/CC hung/VBD down/RP [face her/PP\$ head/NN , and/CC her/PP\$ face/NN] was/VBD covered/VBN [with with/IN [blush blushes/NNS]] ,SENT

2. David Copperfield, Chapter 12
context At/IN length/NN , **confused/VBN by/IN** fright/NN and/CC heat/NN , and/CC doubting/VBG whether/IN half/DT London/NP might/MD not/RB by/IN this/DT time/NN be/VB tuning/VBG out/RP for/IN my/PP\$ apprehension/NN , I/PP left/VBD the/DT young/] man/NN to/TO go/VB where/WRB he/PP would/MD with/IN my/PP\$ box/NN and/CC money/NN ; and/CC , panting/VBG and/CC crying/VBG , but/CC never/RB stopping/VBG , faced/VBN about/IN for/IN Greenwich/NP , which/WDT I/PP had/VBD understood/VBD was/VBD on/IN the/DT Dover/NP Road/NP ; taking/VBG very/RB little/RB more/RBR out/RB of/IN the/DT world/NN , towards/IN the/DT retreat/NN of/IN my/PP\$ aunt/NN , Miss/NP Betsey/NP , than/IN I/PP had/VBD brought/VBN into/IN it/PP , on/IN the/DT night/NN when/WRB my/PP\$ arrival/NN gave/VBD her/PP\$ so/RB much/] umbrage/NN ,SENT

3. David Copperfield, Chapter 19
context [I/PP] suppose/VBP [prospect the/DT opening/NN prospect/NN] **confused/VBD** [me me/PP] ,SENT

4. David Copperfield, Chapter 55
context [I/PP] was/VBD seriously/RB affected/VBN , without/IN knowing/VBG how/WRB much/] , [by by/IN [event late/] events/NNS] ; and/CC [exposure my/PP\$ long/] exposure/NN [to to/TO [wind the/DT fierce/] wind/NN] I had/VBD **confused/VBN** [me me/PP]

pretty , that I stopped in a sort of wonder ; and they all observed her at the same time , for as I stopped , they laughed and looked at her .

Emly is like me , ' said Peggotty , ' and would like to see him .

Emly was **confused by** our all observing her , and hung down her head , and her face was covered with blushes . Glancing up presently through her stray curls , and seeing that we were all looking at her still (I am sure I , for one , could have looked at her for hours) , she ran away , and kept away till it was nearly bedtime .

I lay down in the old little bed in the stem of the boat , and the wind came moaning on across the flat as it had done before . But I could not help fancying . now . that it moaned of those who were gone : and instead of thinking that the sea might rise in the night and float the boat away . I

Abbildung 3.7: CQP-Webdemo

3 Benutzungsschnittstellen

Phrase) einblendbar. Der Umfang des anzuzeigenden Kontexts ist ebenfalls einstellbar.

Neben der normalen Suche ist noch eine Verteilungs- und eine Frequenzsuche möglich. Die *Verteilungssuche* liefert eine Liste aller Texte des Korpus mit der jeweiligen Häufigkeit der Treffer in diesem Text, inclusive einem Diagramm, das die relative und absolute Häufigkeit repräsentiert (Abbildung 3.8a). Die Frequenzsuche liefert eine Liste der eindeutigen Treffer mit deren Anzahl (Abbildung 3.8b). Beide Suchen bieten jeweils Links, mit denen man wieder zur KWIC-Darstellung gelangt, jedoch beschränkt auf einen gewählten Text bzw. eindeutigen Treffer.

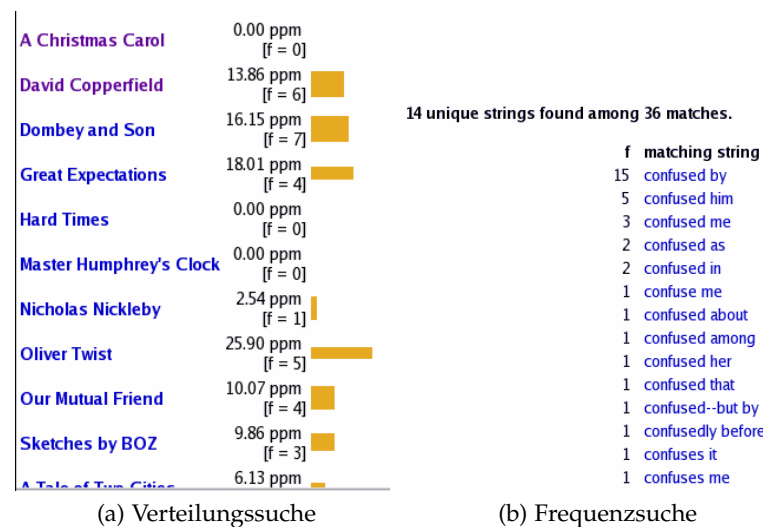


Abbildung 3.8: Verteilungs- und Frequenzsuche mit CQP

3.1.11 A Glossarial DataBase of Middle English

In der *Glossarial DataBase of Middle English* (vgl. Kroymann et al., 2004, 3.2.8)¹⁷ ist eine Stringsuche im Volltext, nur nach Lemmaformen oder nach POS-Tags möglich. Die Suche kann auf Formen nur eines POS-Tags beschränkt werden. Als Ergebnis wird eine Tabelle mit Quelle, Wort, POS-Tag, Sprache und Typ jedes Treffers geliefert, von der aus man zur Anzeige der Zeile des Treffers (ohne Kontext und Weiternavigationsmöglichkeit) gelangt.

¹⁷ <http://www.hti.umich.edu/g/gloss>

3.1.12 The Canterbury Tales Project

Das Canterbury Tales Project¹⁸ bietet Zugriff über eine Webschnittstelle mit der üblichen Suche nach Wortformen und KWIC mit Link auf den Volltext als Ergebnis. Die Volltexte werden jeweils gemeinsam mit gescannten Bildern der betreffenden Buchseiten präsentiert.

3.1.13 Corpus Scriptorum Latinorum (CSL)

Das Corpus Scriptorum Latinorum CSL¹⁹ bietet einen Zugang in der Art einer Websuche sowie Textverzeichnisse nach diversen Kriterien.

3.2 Nicht-Web-Tools

3.2.1 ICECUP III

ICECUP III ist das Anfragewerkzeug zum britischen Teil des (nicht historischen) *International Corpus of English*, ICE-GB²⁰. Es handelt sich um ein Windows-3.11-Programm, das zusammen mit einem kleinen Sampler des Korpus kostenlos zum Download zur Verfügung steht.

Das Programm zeigt Suchergebnisse in einer satzweisen Liste an (Abbildung 3.9 auf der nächsten Seite). Einige Annotationen werden dabei durch Farben bzw. Auszeichnungen wie Durch- und Unterstreichungen kodiert, es ist auch möglich, ähnlich wie bei der CWB (Abschnitt 3.1.10) die Klammerstruktur oder Annotationen für Wortarten etc. einzublenden.

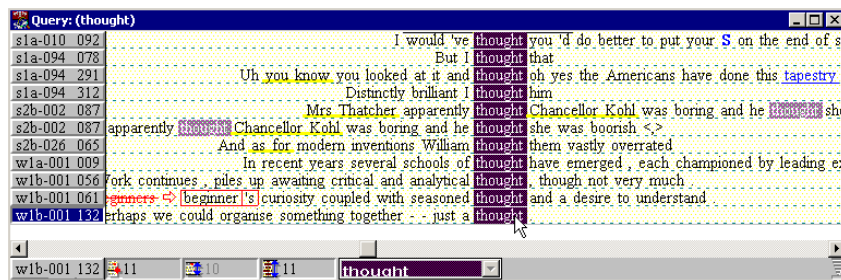


Abbildung 3.9: Anfrageergebnisse in ICECUP III

18 <http://www.cta.dmu.ac.uk/projects/ctp>

19 <http://www.forumromanum.org/literature/>

20 <http://www.ucl.ac.uk/english-usage/ice-gb/>

3 Benutzungsschnittstellen

Darüberhinaus bietet ICECUP III eine navigierbare Darstellung für den Syntaxbaum eines einzelnen Satzes (Abbildung 3.10) sowie einen *Corpus Browser*, mit dem das Korpus in einer hierarchischen Struktur dargestellt und durchblättert werden kann.

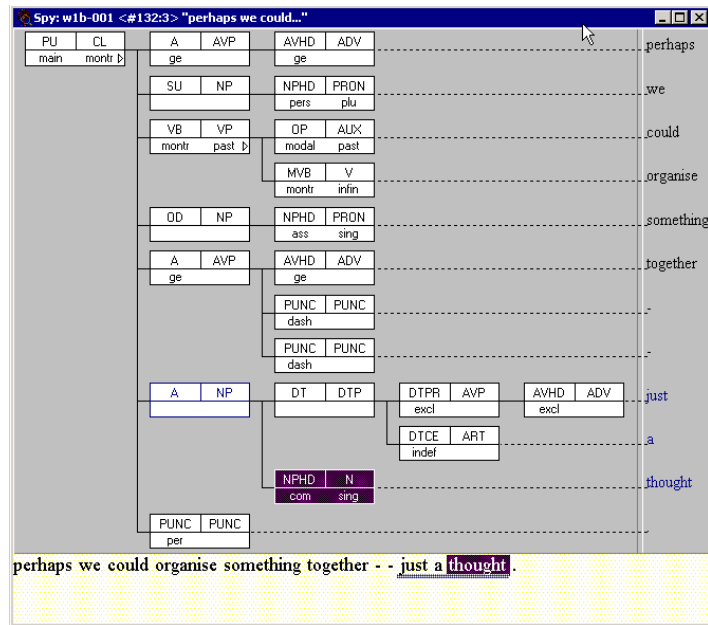


Abbildung 3.10: Syntaxbaum in ICECUP III

Alle Darstellungsoptionen lassen sich auf zahlreiche Weisen konfigurieren. Als Besonderheit können mithilfe dieses Programms Anfragen grafisch formuliert werden.

3.2.2 TIGERSearch und TIGERin

TIGERSearch (Lezius, 2002) ist ein Tool zum Durchsuchen von Baumbanken (Datenbanken mit Syntaxbäumen zu Sätzen, engl. *Treebanks*), dessen Datenformat als Besonderheit Graphstrukturen mithilfe von *sekundären Kanten* unterstützt. Diese sekundären Kanten können beliebige Knoten eines Baumes verbinden. Sie sind allerdings getrennt abzufragen, nicht mit den Standardoperatoren für Dominanz (Elternbeziehung).

Die Suchoberfläche ist in der Art einer integrierten Entwicklungsumgebung gestaltet, also darum bemüht, alle Informationen zum Korpus in einem Fenster zu vereinen. Dabei gibt es:

- einer Baumdarstellung für die Korpora,
- einen Informationsbereich über das selektierte Korpus,
- nach dem Laden eines Korpus einen Informationsbereich mit allerlei Metainformationen über das aktuelle Korpus (vgl. Abbildung 3.11),
- eine Lesezeichenfunktion für vergangene Suchanfragen,
- einen Texteditor für die Anfrage,
- einen grafischen Anfrageeditor (TIGERin, vgl. Voormann, 2002),
- ein Anzeigetool für Syntaxbäume.

Die Ergebnisse von Anfragen kann man in einer Graphanzeige-Komponente (die auch zum Explorieren des Korpus genutzt werden kann) in Syntaxgraphform betrachten (oder ins Grafikformat SVG exportieren). Zusätzlich ist eine Statistikansicht mit einer Häufigkeitsanalyse (samt Exportfunktion) vorhanden.

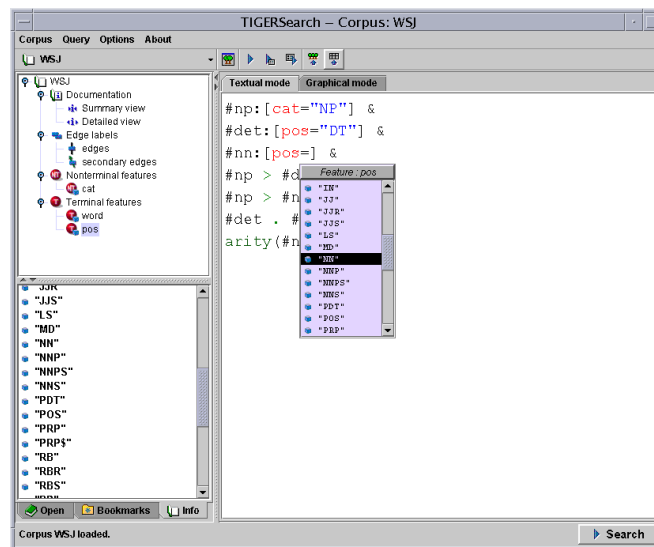


Abbildung 3.11: Korpusmetainformationen in TIGERSearch (aus Lezius, 2002)

3.2.3 WordCruncher

WordCruncher²¹ ist ein kommerzielles Programmpaket für Windows zur Textanalyse. Der kostenlos erhältliche WordCruncher Viewer wird auch von TITUS (Abschnitt 3.1.3 auf Seite 26) als zusätzliche Zugangsmöglichkeit angeboten.

²¹ <http://www.hamilton-locke.com/>

3 Benutzungsschnittstellen

Lat.:	rogavit	eum	a terra	reducere _[Infinitiv]	pussilum		
	bat	ihn	vom Land	zurück(zu)föhren	ein wenig		
Ahd.:	bát	hér	inan,	thaz	her íz fon erdu	arleitti	ein luzzil
	bat	er	ihn,	dass	er es vom Land	wegführe	ein wenig

Abbildung 3.12: Alignierungen und Interlinearübersetzung. Beispiele nach (Lüdeling et al., 2003).

Die Software kennt diverse Such- und Anzeigemöglichkeiten, wie sie auch von verschiedenen der Webtools angeboten wird; ein Fokus liegt auf einer besonders komfortablen Benutzungsoberfläche. Eine Besonderheit sind insbesondere statistische Analysen. Als Hilfsmittel zur Formulierung von Suchanfragen wird ein *Word Wheel* angeboten, eine Liste der möglichen Suchwörter zusammen mit deren Häufigkeit im aktuellen Text.

3.3 Sonstiges

3.3.1 Interlinearer Text

Eine Darstellungsform für alignierte Textebenen (und auch für Texte mit bestimmten Annotationen) ist *interlinearer Text*. Dabei werden verschiedene Varianten oder Annotationen eines Textes so untereinander geschrieben, dass die miteinander alignierten Objekte (i. A. Wörter) untereinander stehen. Üblich ist diese Darstellung etwa für Interlinearübersetzungen, in denen Texte in einer Sprache mit ihrer Übersetzung in Verbindung gebracht werden.

Bow et al. (2003) und Schmidt (2003) betrachten interlineare Texte genauer. Bow et al. gehen dabei auch auf Rendering-Probleme, eine Repräsentation in XML und Transformation mit XSLT ein.

In Lüdeling et al. (2003) wird dabei auch eine Darstellungsform (vgl. Abbildung 3.12 auf der nächsten Seite) vorgestellt, in der Wörter, die in anderer Reihenfolge als in ihrem alignierten Äquivalent vorkommen, mit diesem graphisch verbunden werden.

3.4 Auswertung existierender grafischer Benutzungsschnittstellen

Die untersuchten Korpora können Suchergebnisse in recht unterschiedlichem Umfang darstellen. Es gibt in der Art der Darstellung jedoch insbesondere bei den besser ausgestatteten Korpora deutliche Übereinstimmungen.

Dieser Abschnitt stellt die wesentlichen und auch für DDD interessanten GUI-Elemente vor, Abschnitt 4.2 ab Seite 46 erläutert die dafür notwendigen Anforderungen an die vom Anfragesystem zur Verfügung gestellten Daten.

3.4.1 Textbelege und Volltext

Viele der Korpora bieten Links auf den Volltext, oftmals ist darin dann das Suchwort hervorgehoben.

Es ist nicht immer sinnvoll, den gesamten Volltext (auf einer Seite) darzubieten. Das Digitale Mittelhochdeutsche Textarchiv etwa bietet hier eine an der physischen Struktur der Originalquelle orientierte Aufteilung in Seiten (vgl. Abbildung 3.1c auf Seite 27), in anderen Korpora wird in der Volltextdarstellung ein Abschnitt des Textes mit Navigationsmöglichkeiten zu anderen Bereichen angeboten.

3.4.2 Annotationen

Viele Korpora zeigen, auch wenn sie eine Suche über Annotationen ermöglichen, diese nicht an. Höchstens die physische Aufteilung in Zeilen, Seiten etc. wird dargeboten. Bei den historischen Webtools findet man lediglich in TITUS (Abschnitt 3.1.3 auf Seite 26) zwei Annotationsebenen gleichzeitig dargestellt, nämlich die physische und die logische Struktur (am Beispiel der Merseburger Zaubersprüche in Abbildung 3.2 auf Seite 28).

Die CQP-Webdemo bietet diverse Anzeigemöglichkeiten für Annotationen (vgl. Abbildung 3.7 auf Seite 35); dabei werden Annotationen auf Wortebene unterstützt sowie die Baumstruktur der Grammatik mit Klammern, Anmerkungen und Farben realisiert.

In ICECUP III (Abschnitt 3.2.1) werden etwa auch Syntaxbäume dargestellt.

WordCruncher (Abschnitt 3.2.3 ab Seite 39) nennt gewisse Annotationen »nichthierarchische Annotationen« – gemeint sind Annotationen wie

etwa der Sprecher in einem dramatischen Text, d. h. immer, wenn der Sprecher wechselt, findet sich an der entsprechenden Stelle eine Annotation.

ACT (vgl. Abschnitt 3.1.9 auf Seite 32) bietet eine Detailansicht, in der alle Annotationen zu einem Wort tabellarisch angezeigt werden können.

Lüdeling et al. (2004) zeigen eine Annotationsstruktur, in der die Annotationen tabellarisch in der Art interlinearen Texts mit den Inhalten (und ggf. weiteren alignierten Varianten des Texts) dargestellt werden.

Schmidt (2003) stellt interlinearen Text für Annotationen vor. Eine vergleichbare tabellarische Darstellung wird auch von Lüdeling et al. (2004, S. 17) präsentiert.

3.4.3 Keyword in Context (KWIC)

Mit einer *Keyword-in-Context*-Anzeige (KWIC) stellen die Oberflächen das Suchwort bzw. die Suchwörter in ihrer unmittelbaren Umgebung im Text dar. Dabei gibt es i. A. einen Link zum Volltext oder zumindest einem erweiterten Kontext.

Das Corpus del Español verwendet für die KWIC-Darstellung eine Tabelle (vgl. Abbildung 3.5 auf Seite 33), die auch nach Kriterien wie dem auf den Suchbegriff folgenden Wort sortiert werden kann.

3.4.4 Alignments

Alignments wurden unter den betrachteten Tools nur beim Lancaster Newsbook Corpus (Abschnitt 3.1.6, vgl. Abbildung 3.4 auf Seite 31) dargestellt: Dort werden je zwei zu vergleichende Texte satzweise gegenübergestellt, die Unterschiede werden farbig gekennzeichnet.

Bei Lüdeling et al. (2004, Tabelle 3, S. 16) wird eine andere, tabellarische Form der Alignierung dargestellt (Abbildung 3.13 auf der nächsten Seite).

S	w	e	r	l	e	n	r	e	c	h	t	k	û	n	n	e	n	w	i	l	·	đ	v	o	l	g	e
S	w	e	r	l	e	n	r	e	c	h	t	k	û	n	n	e	n	w	i	l	·	der	v	o	l	g	e

Abbildung 3.13: Alignierung von eng- und weitdiplomatischer Fassung (aus Lüdeling et al., 2004)

Im Zusammenhang mit Interlinearübersetzungen (vgl. Bow et al., 2003) sind wortweise tabellarische Alignierungen üblich (vgl. Abschnitt 3.3.1 auf Seite 40). Schmidt (2003) demonstriert diese Annotationsform auch für

andere Anwendungen wie Annotationen oder Alignments mit bildlichen oder symbolischen Darstellungen.

3.4.5 Frequenztabelle

Insbesondere größere Korpora bieten *Frequenztabellen* an, um die Ergebnisse einer Suchanfrage darzustellen. Die einfachste Form davon ist im Mittelhochdeutschen Textarchiv (vgl. Abschnitt 3.1.1 auf Seite 25) realisiert, hier werden einfach alle matchenden Wortformen mit der Anzahl der Treffer dargestellt (Abbildung 3.1a auf Seite 27).

Das Corpus del Español (Abschnitt 3.1.8) bietet eine feste Tabelle mit den Wörtern bzw. Phrasen in den Zeilen und den Jahrhunderten (bzw. feineren Gruppen für das 20. Jh.) in den Spalten (Abbildung 3.5 auf Seite 33). Interessant ist hier die farbige Markierung der Tabellenzellen nach Trefferanzahl.

Die MHDDB (Abschnitt 3.1.5 auf Seite 28) bietet die variabelste Frequenztabelle: Je nach Einstellungen kann hier (in der Begriffsdimension) nach Lemmata gruppiert werden, in der Korpusteildimension kann etwa nach Textsorten gruppiert werden. Für alle Gruppen sowie für die Gesamtheit sind Spalten realisiert (Abbildung 3.3a auf Seite 30).

Aus den Frequenztabellen gelangt man i. A. in eine KWIC-Darstellung (Abschnitt 3.4.3 auf der vorherigen Seite). Dabei wird, je nachdem, was in der Frequenztabelle angeklickt worden ist, der Suchbereich beschränkt; wahlweise auf Teilkorpora, einzelne Texte, einzelne Wortformen bzw. Lemmata oder Kombinationen daraus.

3.4.6 Verzeichnisse

Neben dem Zugang über eine Wortsuche gibt es oft noch die Möglichkeit, über Verzeichnisse z. B. der verfügbaren Texte zu navigieren. Teilweise werden auch in den Suchinterfaces Listen gewisser Inhalte (wie etwa der verfügbaren Texte oder Textkategorien) angeboten, aus denen die Nutzer dann eine Einschränkung ihrer Frage wählen können.

3.4.7 Metadaten

GUIs können auch für in eine Anfrage einzugebende Werte Auswahllisten oder ähnliche Mechanismen bieten. Dafür kann es erforderlich sein, Meta- bzw. Schemainformationen (etwa die Menge aller verfügbaren Part-of-Speech-Tags oder aller verfügbaren Texte) an die GUIs zu kommunizieren.

Während ersteres noch hinterfragbar ist (da üblicherweise feste Tagsets verwendet werden und die möglichen POS-Tags so als Schemabestandteil in die GUIs hartkodiert werden könnten), ist die Rückgabe von der Extension abhängiger Daten definitiv zu unterstützen.

3.4.8 Navigation

Einige der Tools bieten umfangreiche Möglichkeiten zur Weiternavigation. So führen Klicks auf Wörter in der Volltextanzeige mitunter zu Wörterbucheinträgen, zu Suchen mit diesen Wörtern oder zu weiteren Auswertungen bezüglich dieses Wortes.

4 Datenschema für Anfrageergebnisse

In diesem Kapitel soll das XML-basierte Datenformat für die Anfrageergebnisse näher vorgestellt werden. Wie im Einführungskapitel vorgestellt, werden die Ergebnisse einer Anfrage entsprechend vom Anfragesystem in diesem Format zurückgegeben. GUIs können die so präsentierten Ergebnisse dann visuell aufbereiten und dem Benutzer präsentieren.

Abschnitt 4.1 schildert Kriterien für den Entwurf des Datenmodells. In Abschnitt 4.2 auf der nächsten Seite werden dann die für die diversen im vorigen Kapitel zusammengestellten GUI-Darstellungen benötigten Daten sowie einige besondere Aspekte der Darstellungsform behandelt. Abschnitt 4.3 ab Seite 55 stellt das Schema vor.

4.1 Entwurfskriterien

In Kapitel 2 ab Seite 15 wird das Datenmodell des Korpus erläutert, das dem Anfragesystem zugrundeliegt. Das für die Ergebnisse verwendete Datenschema wird auf diesem Modell aufbauen.

Darüberhinaus werden die für typische GUI-Anwendungen benötigten Daten, wie sie in Abschnitt 4.2 auf der nächsten Seite vorgestellt werden, Berücksichtigung finden müssen. Auch die in Kapitel 6 vorgestellten Anforderungen an die Sprache und die Sprache selbst sind für das Datenformat relevant.

Insbesondere ist hier zu berücksichtigen, dass die Endanwender eher mit der Anfragesprache direkt in Verbindung kommen als mit dem hier entwickelten Datenformat. Wenn ein mögliches Feature des Ergebnisschemas deshalb besonderer Unterstützung durch die Anfragesprache bedarf, muss zwischen der Nützlichkeit dieses Features und der Verkomplizierung der Sprache abgewogen werden.

4.2 Daten für die GUI-Darstellung

Dieser Abschnitt stellt zusammen, welche Informationen für die in Abschnitt 3.4 vorgestellten GUI-Darstellungen benötigt werden (und also vom Datenformat repräsentiert werden sollen).

4.2.1 Textbelege und Volltext

Markierungen der Suchergebnisse sind bereits in Abschnitt 4.2.3 auf der nächsten Seite behandelt worden.

Sowohl um den Volltext anhand gewisser Annotationen (etwa zur physischen Struktur, vgl. etwa die Darstellung im Digitalen Mittelhochdeutschen Textarchiv in Abbildung 3.1c auf Seite 27) rendern zu können, als auch um zusätzliche Eigenschaften von Textabschnitten auszeichnen zu können (vgl. etwa die CQP-Webdemo, Abbildung 3.7 auf Seite 35), sollte es möglich sein, Annotationen auszuwählen, die mit dem Volltext gemeinsam zurückgegeben werden.

Insbesondere, wenn diese Annotationen die Struktur des Textes repräsentieren, sollten sie mit IDs oder einer ähnlichen Identifikationsmöglichkeit versehen werden, so dass die GUIs eine Möglichkeit anbieten können, ab dem betreffenden Textobjekt, Wort bzw. Token weiterzusuchen oder zu navigieren.

Unter Umständen ist es sinnvoll, nur einen Teil des Volltextes auszugeben; insbesondere bei besonders langen Texten.

4.2.2 Annotationen

Wie die in Abschnitt 3.4.2 auf Seite 41 zusammengestellten Beispiele existierender GUIs zeigen, sollten möglichst Annotationen des Textes aus mehreren Ebenen zur Verfügung stehen. Bei den Anfragen gibt es dabei oft eine primäre Annotationsebene, die auch das Arrangement der Inhalte in der Präsentation bestimmt, und weitere ergänzende Informationen.

Eine Ebene sollte sich als primäre Annotationsebene wählen lassen, die dann als hierarchische Ebene zurückgeliefert wird. Dies ist nutzbar z. B. für die Klammerdarstellung der Satzstruktur, oder für ein physisches Layout des anzuzeigenden Textes anhand dieser Annotationsebene.

Annotationen von weiteren Ebenen sollten sich integrieren lassen. Für die Realisierung von Darstellungen wie in Abbildung 3.2 auf Seite 28 ist die von Faulstich (2005) vorgestellte Variante für mehrere Annotationsebenen eine Möglichkeit.

Listing 4.1: Integration von Textabschnitten in Standoff-Annotationen

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xmlns:gXDF="http://www.deutschdiachrondigital.de/namespaces/DDD">

    <!-- Select leaf nodes with associated span -->
    <xsl:template match="*[@tref and @start and @end and not(child::*)]">
        <xsl:copy>
            <xsl:copy-of select="@*" />
            <xsl:value-of
                select="substring(//gXDF:text[@tid=current()/@tref],
                    current()/@start+1, (current()/@end - current()/@start))" />
        </xsl:copy>
    </xsl:template>

    <xsl:template match="*">        <!-- Simply copy everything else -->
        <xsl:copy>
            <xsl:copy-of select="@*" />
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Eine Alternative dazu ist, auch im Ergebnisformat Standoff-Annotation wie etwa im vorgesehenen internen Austauschformat des Projekts gXDF (vgl. Vitt, 2004) zu verwenden, d. h. alle Annotationen getrennt von den Texten zu speichern und die Verbindung über Spans (vgl. Abschnitt 2.2 auf Seite 15 herzustellen. Diese Variante bietet eine unproblematische Möglichkeit, beliebig viele unabhängige Annotationshierarchien mit demselben Text zu verbinden.

Zudem lässt sich auch aus dieser Darstellung leicht eine Variante mit integrierten Texten erzeugen, wie in Listing 4.1 gezeigt.

Für eine tabellarische Darstellung aller Annotationen will man ggf. eine Möglichkeit, alle Annotationen zu einem bestimmten Span abzufragen. Analog sollte es für eine Exportfunktionalität möglich sein, alle Informationen z. B. zu einem bestimmten Text zu selektieren.

4.2.3 Keyword in Context (KWIC)

Für die KWIC-Darstellung (vgl. etwa Abbildung 3.5) muss spezifiziert werden:

1. Was ist der Kontext?

2. Was ist der Treffer?

Auch in diesem Falle stellt sich wiederum die Frage, ob man die Hervorhebungen und Kontextwahl als Mittel in die Sprache einbaut oder ob man stattdessen die Hervorhebungen von der User-Interface-Software vornehmen lässt. Für letzteres spricht, dass die Anfragesprache einfach gehalten werden kann; dagegen, dass man oftmals gerade die Ergebnisse einer komplexen Suche markieren will und hier die entsprechenden Markierungsvorgänge doppelt leisten muss, bzw. dass man ggf. zur Markierung Daten benötigt, die zur Anzeige nicht erforderlich sind.¹

Da man Treffer auch in Volltextdarstellungen hervorheben wollen wird, ist eine generelle Möglichkeit des Hervorhebens oder Markierens hier sinnvoll.

Wenn eine solche Markierung umgesetzt wird, dann sind für die KWIC-Darstellung keine weiteren Vorkehrungen im Datenformat zu treffen: Sofern wie oben beschrieben der genaue Treffer mittels eines Markierungselementes (bei *Keyword* in Context ist es genau ein Element) ausgezeichnet ist und der Kontext den gesamten Treffer der Abfrage bildet, lassen sich die Tokens davor und danach mittels XPath- bzw. XSLT-Ausdrücken selektieren. Es ist jedoch ggf. empfehlenswert, einen *Gesamttreffer* mitsamt Kontext extra auszuzeichnen.

Etwa im *Corpus del Español* ist auch eine Sortierung nach bestimmten Kontextelementen vorgesehen (vgl. Abschnitt 3.1.8 auf Seite 32). Hier ist eine Designentscheidung zu treffen, ob man bereits in der Anfragesprache Sortiermöglichkeiten vorsieht und diese damit weiter verkompliziert oder die Sortierung den GUIs überlässt – im Sinne des in Abschnitt 4.1 vorgesehenen Kriteriums der Einfachheit der Sprache fällt diese Entscheidung zugunsten einer Sortierung innerhalb der GUIs aus, zumal für ein bloßes Rearrangement der Ergebnisse auch keine Datenübertragung zum RDBMS notwendig ist.

4.2.3.1 **Umsetzung der Markierungen**

Zur Markierung der Suchergebnisse in einem XML -basierten Ausgabeformat gibt es mehrere Möglichkeiten:

Markierung der matchenden XML-Elemente mit einem Attribut. Dabei wird ein Attribut definiert, das ein XML-Element als Suchergebnis

¹ Beispiel: »Markiere alle Wörter in Text *a*, die mit mehreren Wörtern in Text *b* aligniert sind.«

klassifiziert und das (gemäß Schema) in jedem ein Ergebnis enthalten könnenden Element vorkommen kann. Im folgenden Beispiel wird hierzu das Attribut `result` benutzt:

```
<sentence>
  <word>Dabei</word> <word>wird</word> <word>ein</word>
  <word result="true">Attribut</word> <word>definiert</word>
</sentence>
```

- Es kann jeweils nur auf Elementebene markiert werden, eine zusammenhängende Markierung über mehrere Elemente oder die Markierung nur eines Teils eines Elements ist nicht möglich
- + über den Attributwert bzw. das Attribut kann eine Klassifizierung vorgenommen werden

Markierung der Ergebnisinhalte mit einem Element Dabei wird ein neues XML-Element (hier `match`) definiert, das den zu markierenden Teil umschließt.

```
<sentence>
  <word>Dabei</word> <word>w<match no="1">ir</match>d</word>
  <match><word>ein</word> <word>Attribut</word></match>
  <word>definiert</word>
</sentence>
```

- + es können zusammenhängende Matches, die sich über mehrere Elemente erstrecken, oder die innerhalb eines Elements liegen, markiert werden
- die Markierung muss sich in den existierenden Elementbaum eingliedern
- + beliebige Attribute können zur Annotation verwendet werden

Markierung des Ergebnisbereichs mit Milestones Milestones² sind eine von der TEI vorgesehene Möglichkeit zum Markup nebenläufiger Hierarchien. Dabei werden leere Elemente zur Markierung sekundärer Hierarchien benutzt. Ein Match ließe sich etwa durch die Elemente `match-start` und `match-end` markieren:

² <http://www.tei-c.org/P5/Guidelines/NH.html#NHMI>

4 Datenschema für Anfrageergebnisse

```
<sentence>
  <!-- ... --> <word>ist</word> <matchstart/><word>schön</word>.
</sentence>
<sentence>
  <word>Deshalb</word><matchend/><word>sollte</word><!-- ... -->
</sentence>
```

- + es können zusammenhängende Matches, die sich über mehrere Elemente erstrecken, oder die innerhalb eines Elements liegen, markiert werden
- + die Markierung muss sich nicht in den existierenden Elementbaum eingliedern
- + beliebige Attribute können zur Annotation verwendet werden
- + Behandlung an Beginn und Ende eines Matches ist einfach
- Extraktion von Matches, die der Hierarchie widersprechen (wie im Beispiel oben) ist schwierig, da sie keinen zusammenhängenden Teil des Elementbaums bezeichnen
- Verschachtelte Matches bedürfen besonderer Behandlung

Fragmentierung Auch Fragmentierung³ gehört zu den von der TEI vorgeschlagenen Methoden zum Markup nebenläufiger Hierarchien. Dabei wird das betreffende Element aufgespalten, wenn es die Grenzen der primären Hierarchie des Ergebnisses überschreiten soll.

Zur Rekonstruktion der Fragmente sind verschiedene Methoden möglich: (a) *Virtual Joins* nach TEI⁴, bei denen in einem separaten Element alle zusammengehörenden Fragmente aufgeführt werden; (b) eine Verkettung mittels Attributen *next* und *prev* (ebenfalls nach TEI) oder (c) eine Markierung aller zusammengehörigen Elemente eines Treffers mit einem gleichwertigen Attribut. Im Beispiel wird Variante c demonstriert:

```
<sentence>
  <!-- ... --> <word>ist</word>
  <match no="1"><word>schön</word>.</match no="1">
</sentence>
<sentence>
  <match no="1"><word>Deshalb</word></match no="1">
```

³ <http://www.tei-c.org/P5/Guidelines/NH.html#NHFR>

⁴ <http://www.tei-c.org/P5/Guidelines/NH.html#NHVI>

```
<word>sollte</word>  
<!-- ... -->  
</sentence>
```

Joins über Spans: Standoff-Markierungen Die hervorzuhebenden Bereiche der Matches werden über eine separate Liste von Spans zurückgeliefert. Dieses Verfahren entspricht einer Standoff-Annotation, in der die Markierungen wie eine Annotationsebene behandelt werden.

- nur Matchen auf Text möglich, auf Elemente nur über Join mit Spans
- Extraktion der Matches erfordert komplexen Postprocessingschritt (Join).

Andererseits ist dies dasselbe Nachbearbeitungsverfahren, dass für die im Projekt verwendete Standoff-Annotation ohnehin verwendet wird (vgl. auch Listing 4.1 auf Seite 47).

Fazit Sofern die Anfrageergebnisse sich in die Struktur des Ergebnisses (und damit der primären Annotationsebene) eingliedern, erscheint die Markierung mittels eines speziellen Elements eine geeignete Methode, da damit Spans innerhalb eines (Kind-) Elements ebenso wie Folgen von Elementen markiert werden können und die Weiterverarbeitung im XML-Datenmodell erfolgen kann.

Falls die Markierungen *gelegentlich* gegen die (primäre) Hierarchie verstoßen, bietet eine Fragmentierung des Markierungselementes einen wenig invasiven Ansatz.

Andererseits entspricht eine Standoff-Annotation dem sonstigen Umgang mit Spans, und ist, wie in Listing 4.1 auf Seite 47 gezeigt, ebenfalls einfach auflösbar.

4.2.4 Alignments

Bow et al. (2003) stellen für interlinearen Text eine XML-Repräsentation vor, die sie mit XSL (über XSL Formatting Objects) in diverse Anzeigeformate, einschließlich HTML, gewandelt zu haben angeben. Diese Repräsentation berücksichtigt noch nicht die komplexeren Alignments (vgl. 3.4.4 auf Seite 42), und es muss auch noch geklärt werden, inwiefern sich dies auf *zeichenweise* Alignierungen übertragen lässt.

Ziele können sein:

- Gemeinsame Darstellung der (kleinsten) miteinander alignierten Einheiten.

Daraus kann man eine Art Interlineardarstellung generieren:

Et	non
inti	ni

- Darstellung der Edit-Operationen.

Diese kann man benutzen, um die tatsächlichen Veränderungen zwischen den Texten darzustellen.

- Markup der Texte mit Edit-Operationen.

Diese Darstellung kann visuell in der Art visueller Diff-Utilities (etwa Vim-Diff, Emacs' EDiff oder der visuelle Diff-Modus von Eclipse) präsentiert werden, man markiert also in einem der Texte Stellen, die im anderen fehlen oder geändert wurden farbig oder durch ähnliche Hervorhebungen. Vgl. hierzu auch die Darstellung des *Lancaster Newsbook Corpus* in Abbildung 3.4 auf Seite 31.

- Pseudo-Tabellarische Darstellung

Dies ist eine nahe an der Darstellung einer Tabelle in HTML orientierte Fassung, also

```
<table>
  <tr><td>Dies</td><td>ist</td><td>ein</td><td>Satz</td></tr>
  <tr><td>Dies</td><td>ist</td><td>/>      <td>Satz</td></tr>
</table>
```

Da im DDD-Korpus komplexe Alignments mit Ersetzungen, Vertauschungen, Einfügungen etc. vorgesehen sind und im von Faulstich (2005) vorgestellten und in Abschnitt 3.4.4 ab Seite 42 verfeinerten Datenmodell für Alignments auch beliebige Attribute an allen Objekten der Alignments vorgesehen sind, ist es sinnvoll, auch im Ergebnisformat keine Beschränkungen gegenüber diesem Entwurf vorzunehmen, um den UIs alle Möglichkeiten zur Visualisierung zu lassen. Es sollten also möglichst alle im Korpus zu den ausgewählten Alignments verfügbaren Informationen übernommen und die tatsächliche Formatierung den Benutzungsoberflächen überlassen werden.

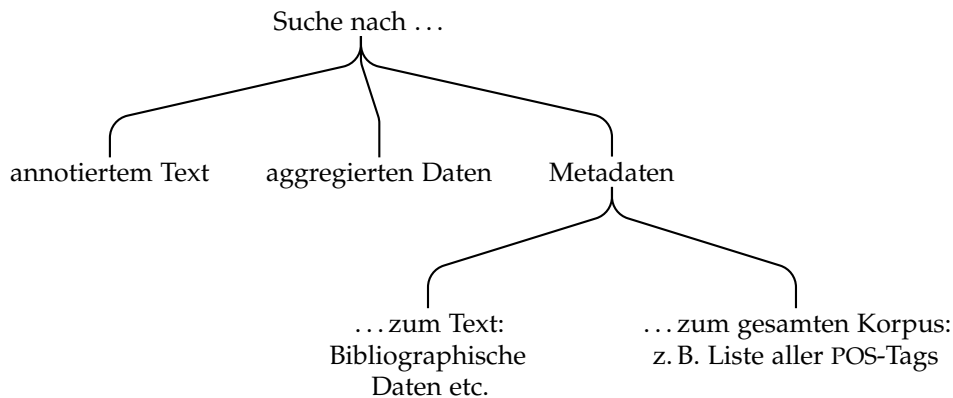


Abbildung 4.1: Suchtypen. *Aggregierte Daten* sind Daten, die beispielsweise durch Anwendung statistischer Operatoren (vgl. Faulstich et al., 2005) gewonnen werden. Diese Daten können unter anderem für eine komprimierte Darstellung von Suchergebnissen etwa als Frequenztabelle (vgl. Abschnitt 4.2.5) genutzt werden. *Metadaten* sind Daten, die in der Datenbank gespeichert werden, jedoch nicht in unmittelbarer Beziehung zu einem Text stehen – z. B. die Liste aller möglichen Part-of-Speech-Tags.

4.2.5 Frequenztabelle / Aggregierte Daten

Für die Realisierung von Frequenztabellen gibt es grundsätzlich zwei Ansätze:

1. Man definiert *Aggregationsoperatoren* in der Anfragesprache oder
2. man verlagert die Aggregation in die Nachbearbeitungsschritte.

Lösung 2 vereinfacht das Austauschdatenformat und die Anfragesprache, führt jedoch dazu, dass eine Menge unnötiger Daten transportiert werden müssen und dass die Aggregationsoperatoren unabhängig vom RDBMS zu implementieren sind (und somit insbesondere die Aggregationsfähigkeiten moderner RDBMS nicht ausgenutzt werden könnten).

Lösung 1 erfordert dagegen die Definition von Aggregationsoperatoren in der Anfragesprache (wie in Faulstich et al. (2005, Abschnitt 3.4.8) vorgeschlagen, aber noch nicht realisiert) *und* eine entsprechende Realisierung im Datenmodell: Dort müssen bestimmte generierte Daten möglich sein, die so nicht im Korpusdatenmodell vorkommen, oder man muss Teile des Ergebnisses durch aggregierte Daten ersetzen können.

Da die aggregierten Daten mit Texten in Zusammenhang stehen und genutzt werden können, um Teile der Suchergebnisse zu repräsentieren,

4 *Datenschema für Anfrageergebnisse*

ist eine Eingliederung in die Struktur zur Suchergebnisrepräsentation sinnvoll – etwa durch ein Element, das anstelle oder zusätzlich zu den zusammengefassten Inhalten eingefügt wird. Alternativ ist auch eine bloße Umsetzung über Attribute denkbar, aber hier sind die Möglichkeiten, Inhalte zu enthalten, recht eingeschränkt.

Eine konkrete Lösung für aggregierte Daten wird in Abschnitt 7.4.8 ab Seite 102 vorgestellt.

4.2.6 Verzeichnisse

Für die Anzeige von Verzeichnissen der Texte (vgl. Abschnitt 3.4.6 auf Seite 43) ist eine Abfrage der entsprechenden Metainformationen nötig. Suchen, insbesondere nach Strukturelementen, sollte nicht nur im Volltext, sondern auch in den Katalogdaten möglich sein.

Auch die Ergebnisse von Suchanfragen auf den Inhalten des Korpus sollten mit den zugehörigen bibliographischen Informationen verknüpft werden können, so dass GUIs beispielsweise zu jedem Treffer den Titel des entsprechenden Texts anzeigen können.

Schließlich ist es auch hier nötig, dass Ergebnisse in Form von Teilbäumen zurückgeliefert werden können; etwa um die Ergebnisse von Anfragen der Art »liefere alle bibliographischen Daten zu Gedichten des 17. Jahrhunderts« in strukturierter Form auswerten zu können (und sie z. B. nach Autoren gruppieren).

4.2.7 Metadaten

Metadaten, die explizit im Korpus realisiert sind, sollten prinzipiell auch abfragbar sein (beispielsweise die Menge aller Part-of-Speech-Tags), so dass UIs entsprechende Auswahllisten oder ähnliche Eingabehilfen generieren können. Gegebenenfalls muss man hier allerdings eine andere Struktur für das Rückgabeformat vorsehen als für die »normalen« Anfrageergebnisse.

Da eine solche Metadatenabfrage jedoch primär ein Bedürfnis der Softwareschnittstelle deckt, sollte sie nicht in die vom Nutzer zu lernende Anfragesprache aufgenommen werden, wenn sie diese weiter verkompliziert. Stattdessen können diese Daten der Software über andere Schnittstellen zur Verfügung gestellt werden.

4.2.8 Navigation und Query Refinement

Für die Realisierung von Navigationsmöglichkeiten von einem Anfrageergebnis zu einem neuen (vgl. Abschnitt 3.4.8 auf Seite 44) ist es sinnvoll, auf jedes einzelne Element des Ergebnisses Bezug nehmen zu können. Dazu sollten eindeutige IDs aus der Datenbank sowohl zur Selektion zugelassen als auch in den Ergebnissen repräsentiert werden.

4.3 Entwurf des Ergebnisschemas

Für die Anfragesprache wird in Abschnitt 7.2 die Entscheidung vorgestellt, sich auf eine bloße Selektion von Ergebnissen auf der Datenbank zu beschränken und auf Funktionen zur Restrukturierung der Daten zu verzichten, um die Sprache nicht unnötig zu verkomplizieren. Die Reorganisation der Daten wird dann nachbearbeitenden Tools (i. A. den GUIs) überlassen. Da die Ergebnisse im Industriestandard XML präsentiert werden, stehen hierfür mächtige Werkzeuge – etwa XSLT – zur Verfügung.

In diesem Sinne wird auch das Datenschema für die Anfrageergebnisse entwickelt. Das Schema baut auf dem bereits existierenden internen Austauschformat gXDF auf, das die Spezifikation von ODAGs mithilfe von Proxy-Elementen (vgl. Vitt, 2004) erlaubt.

4.3.1 gXDF

gXDF ist ein Format zur Standoff-Annotation, d. h. Texte und Annotationen sind voneinander getrennt dargestellt: Es gibt einen `texts`-Abschnitt, der die Texte enthält, und einen `elements`-Abschnitt für die Elemente. Die Texte sind mit einer ID versehen und mit einem Intervall annotiert. Bei vollständiger Repräsentation eines Texts t in einem Element `text` ist dieses Intervall $[0, |t| - 1]$ (wobei $|t|$ die Länge des Textes repräsentiert), wird nur ein Ausschnitt des Textes dargeboten, ist das Textelement mit dem entsprechenden Teilintervall annotiert.

Innerhalb des `elements`-Teils können beliebige Elemente stehen, die in einem separaten Schema für das Korpus zu spezifizieren sind.

Diese Elemente können mit *Spans* assoziiert werden, indem ihnen (aus dem gXDF-Namensraum) die Attribute `tid` mit einer Referenz auf den Text sowie die Intervallattribute `start` und `end` zugeordnet werden können.

Auch ein `span`-Element für alleinstehende Spans ist in gXDF vorgesehen.

Erweiterungen zur Unterstützung des Datenmodells

Um die in der Anfragesprache relevanten, aber noch nicht im ursprünglichen gXDF vorgesehenen Elemente des Datenmodells zu unterstützen, werden zusätzlich eingeführt:

- ein Element `layer` mit dem String-Attribut `name`, das Nachfahre von `elements` sein kann und jeweils eine Annotationsebene umschließt.
- Elemente `alignment`, `link` und `align` (letzteres mit `spans` als zulässigen Kindern), um Alignments gemäß Abschnitt 3.4.4 ab Seite 42 zu modellieren

4.3.2 Erweiterungen und Anpassungen für *DDDquery*

Die Erweiterungen für *DDDquery* entstammen einem neuen Namensraum, der im Folgenden mit `dq:` abgekürzt wird.

Im Kern ist ein *DDDquery*-Anfrageergebnis ein *Teilgraph* des Korpusgraphen. Die Auswahl der Knoten des Korpusgraphen, die in die Ergebnisknotenmenge übernommen werden, ist in Abschnitt 7.2 auf Seite 81 beschrieben. Die Kanten des Ergebnisgraphen sind die Kanten des durch die Ergebnisknotenmenge induzierten Teilgraphen des Korpusgraphen.

4.3.2.1 Selektierte Elemente

Der Elementteil des Anfrageergebnisses besteht aus den entsprechend der Beschreibung in Abschnitt 7.2 ausgewählten Elementen. Die Verschachtelung der Elemente entspricht dem Korpusgraphen, es ist jedoch zu beachten, dass dabei gewisse Schachtelungsconstraints des Korpusschemas verletzt werden können.

Angenommen beispielsweise das Korpusschema spezifiziere, dass ein `page`-Element immer eine Reihe von `line`-Elementen enthalte, die wiederum `word`-Elemente enthalten können, so dass sich folgendes Beispiel⁵ ergibt:

```
<page no="1">
  <line no="1">
    <word>Eiris</word>
    <word>sazun</word>
  <!-- -->
```

5 Diese XML-Ausschnitte sind aus Gründen der Lesbarkeit *nicht* in gXDF, sondern deutlich vereinfacht


```
</line>  
</page>
```

Dann wird das Ergebnis der Anfrage `//page#/line/word#/"Eiris"` (das # markiert einen Knoten zur Ausgabe) zu

```
<page no="1">  
  <word>Eiris</word>  
  <word>sazun</word>  
  <!-- -->  
</page>
```

– also ohne die line-Elemente!

Elementknoten, die ausgegeben werden, werden immer zusammen mit all ihren Attributen und ggf. mit ihrem assoziierten Span ins Ergebnis übernommen.

4.3.2.2 Benannte Markierungen

Wie in Abschnitt 7.4.6 ab Seite 99 gezeigt, können Knotenmuster in der Anfrage mit einem *benannten Marker* versehen werden. Elementknoten, die mit einem solchen Marker versehen sind, werden mit einem zusätzlichen Attribut `dq:marker` versehen, dessen Textwert der Name des Markers ist.

Auf diese Weise können auch Benutzungsschnittstellen, die einen Anfrageausdruck des Benutzers oder der Benutzerin umschreiben, um z. B. Kontext von eigentlichem Suchmuster zu unterscheiden, einen entsprechend benannten Marker hinzufügen.

4.3.2.3 Aggregierte Inhalte

In Anfragen mit Aggregationsfunktionen wird an der Stelle der Aggregationsfunktion ein Element `dq:aggregate` eingefügt, das die Attribute `function` mit dem Namen der Funktion und `aggregate` mit dem aggregierten Pfadausdruck in *DDDquery*-Syntax trägt. Das Ergebnis der Aggregationsfunktion ist der Textinhalt des `dq:aggregate`-Elements.

Ein Beispiel ist in Abschnitt 7.4.8 ab Seite 102 dargestellt.

4.3.2.4 Texte, Spans und Alignments

Die Textinhalte, die zur Ausgabe vorgesehen sind, werden wie in gXDF in einem separaten `texts`-Element zurückgegeben. Dabei steht jeder Text in einem `text`-Element. Es werden nicht notwendigerweise die vollständigen

Texte zurückgeben, sondern nur das *maximal auszugebende Intervall* eines Textes.

Definition 9 (maximal auszugebendes Intervall)

Sei t ein Text. Sei $S_{\#}$ die Menge aller zur Ausgabe markierter Spans, $E_{\#}$ die Menge aller zur Ausgabe markierter Elemente und $S_{E_{\#}}$ die Menge aller Spans, die mit einem Element aus $E_{\#}$ assoziiert sind.

Sei $\hat{S} := S_{\#} \cup S_{E_{\#}}$. Dann ist das maximal auszugebende Intervall für einen Text t definiert als

$$\begin{aligned} I^+(t) := (l^-, r^+) &\iff (\exists r : \exists (t, l^-, r) \in \hat{S}) \\ &\quad \wedge (\exists l : \exists (t, l, r^+) \in \hat{S}) \\ &\quad \wedge (\forall l', r' : l' < l^- \vee r' > r^+ \rightarrow \neg \exists (t, r', l') \in \hat{S}) \end{aligned}$$

d. h. das maximale Intervall von t enthält alle Spans von t , die in die Ergebnismenge selektiert wurden.

Spans, die zur Ausgabe markiert worden sind, werden explizit als span-Objekte in der Ausgabe aufgeführt.

Alignments werden, wenn sie zur Ausgabe markiert sind, wie Annotationselemente als Standoff-Annotation dem Ergebnis hinzugefügt.

4.3.2.5 Weiternavigation mit IDs

Jedes Element wird mit einer eindeutigen ID als Attribut versehen, nach der in der Anfragesprache (über die normale Attributsyntax) auch selektiert werden kann. Spans werden eindeutig über das Tripel aus Textreferenz, Start- und Endpunkt identifiziert. Auf diese Weise können UIs Navigationselemente anbieten, die auf ein bestimmtes Element des Ergebnisses bezug nehmen und von dort weiternavigieren.

4.3.3 Beispiel

Im Beispiel soll die Anfrage `//token/"Eiris"` ausgeführt werden, es soll also nach Token-Elementen gesucht werden, die mit einem Span assoziiert sind, der den Inhalt »Eiris« hat. Zusätzlich soll Kontext ausgegeben werden: Und zwar alle Annotationen der Annotationsebene mit dem o. g. Treffer, die sich innerhalb desselben verse befinden, sowie zur Einordnung die weiteren Vorfahren aus der zugehörigen Annotationsebene.

Daraus ergibt sich die Beispielanfrage:

```
#+(match) //token/"Eiris" #-(match) \\ verse # $v  
    \\ gXDF:layer #      &      $v/*#
```

Das Ergebnis ist in Listing 4.2 auf der nächsten Seite zu sehen.

Zur Syntax sei grundsätzlich auf Kapitel 7 verwiesen, hier nur die verwendeten Elemente:

`//token/"Eiris"` ist die eigentliche Anfrage nach einem Token-Element mit einem Span, der genau »Eiris« matcht. Dieser Teil ist in Marker `#+(match) ... #-(match)` eingeschlossen, um die Ergebnisse der beiden Schritte `token` und `"Eiris"` mit einer *match* benannte Markierung zu versehen.

Mittels `\\verse # $v` werden dann Vorfahren `verse` selektiert, zur Ausgabe markiert und an die Variable `$v` gebunden. `\\gXDF:layer #` selektiert weiter bis zum nächstäußeren `layer`-Element.

Dazu wird der zweite Ausdruck `$v/*#` verwendet, um alle Nachfahren des an `$v` gebundenen `verse`-Knotens ebenfalls zur Ausgabe zu markieren.

Im Ergebnis der Anfrage sind drei Teile zu erkennen:

Der `gXDF:elements`-Teil enthält die Elementstruktur. Man beachte, dass das zu markierende Token-Element (das erste) ein Attribut `dq:marker` mit dem String aus der Anfrage trägt.

Der `dq:spans`-Teil enthält dann die zu markierenden Spans des Ergebnisses.

Der Textabschnitt ist nach den oben beschriebenen Regeln zum maximal auszugebenden Intervall dann im dritten Teil, `gXDF:texts`, zu finden.

4 Datenschema für Anfrageergebnisse

Listing 4.2: Ergebnis einer Anfrage

```
<?xml version="1.0" encoding="iso-8859-1"?>
<dq:results
  xmlns:dq="http://www.deutschdiachrondigital.de/namespace/dddq-results"
  xmlns:gXDF="http://www.deutschdiachrondigital.de/namespace/gXDF2"
  xmlns="http://www.deutschdiachrondigital.de/namespace/corpus">

  <!-- Die selektierten Elemente in der Originalstruktur -->
  <gXDF:elements>
    <gXDF:layer name="logical" tref="d1t2" nid="layerd1t2logical1">
      <logical nid="d1t2logical1" tref="d1t2" source="d1t1" start="0" end=
        "451">
        <part nid="t2P1" tref="d1t2" no="1" start="0" end="142">
          <verse nid="t2P1v1" tref="d1t2" no="1" start="0" end="35">
            <token dq:marker="match" nid="t2t1" tref="d1t2" start="0" end=
              "5"/>
            <token nid="t2t2" tref="d1t2" start="6" end="11"/>
            <token nid="t2t3" tref="d1t2" start="12" end="17"/>
            <token nid="t2t4" tref="d1t2" start="18" end="23"/>
            <token nid="t2t5" tref="d1t2" start="24" end="28"/>
            <token nid="t2t6" tref="d1t2" start="29" end="35"/>
          </verse>
          <!-- die weiteren Verse und Parts wurden nicht zur Ausgabe
            markiert -->
        </part>
      </logical>
    </gXDF:layer>
  </gXDF:elements>

  <!-- Eingefügt, da der Spanknoten "Eiris" markiert worden ist -->
  <dq:spans>
    <gXDF:span dq:marker="match" tref="d1t2" start="0" end="5" />
  </dq:spans>

  <!-- Der maximal auszugebende Span ist (d1t1, 0, 451) vom logical-
    Element -->
  <gXDF:texts>
    <gXDF:text tid="d1t2" start="0" end="451">Eiris sazun idisi sazun hera
      duoder suma hapt heptidun suma heri lezidun suma clu bodun umbi
      cuoniouuidi insprinc hapt bandun inuar uigandun. Phol ende uuodan
      uuorun zi holza du uuart demo balderes uolon sin uuoz birenkit thu
      biguol en sinhtgunt. sunna era suister thu biguol en friia uolla
      era suister thu biguol en uuodan so he uuola conda sose benrenki
      sose bluotrenki. sose lidi renki ben zi bena bluot zi bluoda lid
      zi geliden sose gelimida sin.</gXDF:text>
  </gXDF:texts>

</dq:results>
```

5 Linguistische Anfragesprachen

In diesem Kapitel werden wichtige bereits existierende linguistische Anfragesprachen kurz vorgestellt. Dies geschieht besonders vor dem Hintergrund des in Kapitel 2 vorgestellten Datenmodells und der Erkenntnisse aus Kapitel 4.3 für die erwünschten Anfrageergebnisse.

Die Erkenntnisse aus diesem Kapitel fließen dann in die Anforderungsliste für *DDDquery* in Kapitel 6 auf Seite 71 sowie in den Sprachentwurf ein.

5.1 Emu

Cassidy und Harrington (2001) stellen Emu vor, ein System zur Anfrage und Annotation von Sprachdaten mit Fokus auf gesprochenen Texten.

Eine Emu-Datenbank besteht aus mehreren *Utterances* – üblicherweise Wörtern oder Sätzen –, von denen jede mit mehreren Annotationsebenen verbunden sein kann. Annotationsebenen bestehen entweder aus *Ereignissen*, die mit einem Zeitpunkt assoziiert sind, oder aus mit einem *Zeitraum* assoziierten *Segmenten*. Lücken und Überlappungen sind möglich.

Es sind drei Arten von Relationen vorgesehen:

- Die *Sequenzrelation* gibt eine Halbordnung über die Objekte eines Levels an, die explizit oder implizit durch die mit einem Objekt assoziierten Zeitinformationen angegeben werden kann;
- die *Dominanzrelation* assoziiert Tokens mit ihren Bestandteilen im selben oder in anderen Annotationsebenen;
- eine allgemeine gerichtete und intransitive *Assoziationsrelation* steht für anwendungsabhängige Assoziationen zur Verfügung.

Dominanz impliziert bei Emu temporale Inklusion, weshalb Ereignisse niemals Eltern in einer Dominanzbeziehung sein können.

Einfache Anfragen beziehen sich jeweils auf ein Token (d. h. Segment oder Ereignis) und begrenzen dessen Labels oder Position relativ zu seinen Geschwisterknoten. Mit zusammengesetzten Anfragen kann man

Beschränkungen auf Dominanz- oder Sequenz- / Präzedenzbeziehungen spezifizieren, der dabei auszugebende Knoten wird entweder mittels eines Hash-Symbols (#) gekennzeichnet oder durch seine Position.

Beispielanfragen sind bei Cassidy und Harrington (2001, S. 71) zu finden sowie bei Lai und Bird (2004).

5.2 XPath-Derivate

5.2.1 LPath

LPath wird von Bird et al. (2005) als Erweiterung von XPath (Clark und DeRose, 1999) um drei typische für linguistische Fragestellungen relevante Features vorgeschlagen:

- (1) *unmittelbare Präzedenz*, da für Präzedenz in XPath nur transitive Hüllen der Präzedenzbeziehung vorgesehen sind;
- (2) *Subtree Scoping*, um eine Teilanfrage auf einen zuvor spezifizierten Teilbaum zu beschränken; und
- (3) *Edge alignment* zur Festlegung der Position eines Knotens im Dokumentbaum (und nicht in der Ergebnissequenz).

Neben diesen zusätzlichen Fähigkeiten der Sprache stellt LPath eine Reihe von Kurzschreibweisen für Navigationsschritte vor, die in XPath teilweise nicht vorgesehen sind (vgl. Bird et al., 2005, Tabelle 1). Die Auswertung erfolgt mithilfe eines Intervallansatzes (vgl. etwa Grust et al., 2004).

5.2.2 Generalized XPath (Cassidy)

Cassidy (2003) stellt einen Ansatz vor, um XPath für allgemeine gerichtete und beschriftete Graphen zu generalisieren. Dabei strebt er vor allem die Aufhebung dreier Beschränkungen von XPath an:

1. Das Ergebnis einer XPath-Anfrage ist eine Knotenmenge, wünschenswert für manche Anwendungen wäre der gesamte traversierte Pfad (also eine *Pfadmenge* als Ergebnistyp). Als Anwendungsbeispiel nennt Cassidy Suchen wie »finde Sequenzen von Phonemen mit Eigenschaft x «.
2. mehr, und vor allem *benutzerdefinierbare Achsen* als beliebige Abbildung von Knoten zu Knotenmengen. Eine Achse ist laut Cassidy

(2003) durch zwei Eigenschaften definiert: Durch den Kantentyp, dem zu folgen ist, und durch die Frage, ob Transitivität gegeben ist oder nicht. Auch bei den existierenden Achsen gibt es nicht in allen Fällen eine intransitive Form (*preceding*, *following*; vgl. Bird et al., 2005), darüberhinaus kann sich Cassidy eine Generalisierung der booleschen Transitivität hin zu einer Schrittzahl vorstellen.

3. Bedingungen auf Pfaden (statt nur auf Knoten).

Ausgehend von einem einfachen Blick auf XPath (bzw. zumindest dessen Kern), der einen XPath-Ausdruck als Folge von Anwendungen von Funktionen der Form *Nodeset* \rightarrow *Nodeset* sieht, schlägt Cassidy Ausdrücke der graphbasierten Sprache als Folge von Funktionsanwendungen auf Pfadmengen vor. In jedem Schritt wird dabei angegeben

- die Achse,
- die Anzahl der Schritte,
- eine Bedingung und
- den Ort im Pfad, an dem die Bedingung gelten soll.

Auf eine mögliche an XPath orientierte Syntax wird nur kurz eingegangen. Konkrete Überlegungen zu einer effizienten Implementierung existieren in (Cassidy, 2003) noch nicht, nur der Hinweis, dass eine so allgemeine Implementierung möglicherweise nicht effizient zu realisieren sei.

5.2.3 EXPath

Von Iacob (2005); Iacob und Dekhtyar (2005) wird die Anfragesprache EXPath vorgestellt, die XPath um Anfragemöglichkeiten für nebenläufige Hierarchien erweitert.

Der Sprache liegt das GODDAG-Datenmodell (Sperberg-McQueen und Huitfeldt, 2004) zugrunde (die Abkürzung GODDAG steht für *Generalized Ordered-Descendant Directed Acyclic Graph*), ein Datenmodell zur Repräsentation nebenläufiger Annotationshierarchien auf demselben Text, bei dem mehrere Annotationshierarchien an der Wurzel sowie auf der Blatt- bzw. Inhaltsebene miteinander verbunden sind (Abb. 5.1 auf der nächsten Seite). Dabei kann nach Iacob und Dekhtyar (2005) die Semantik aller Beziehungen zwischen zwei Knoten aus unterschiedlichen Hierarchien durch Beziehungen zwischen Blattknoten ausgedrückt werden.

Location Steps in EXPath liefern nicht *nodesets*, sondern *nodeset collections* zurück, die jeweils ein *nodeset* pro Hierarchie enthalten.

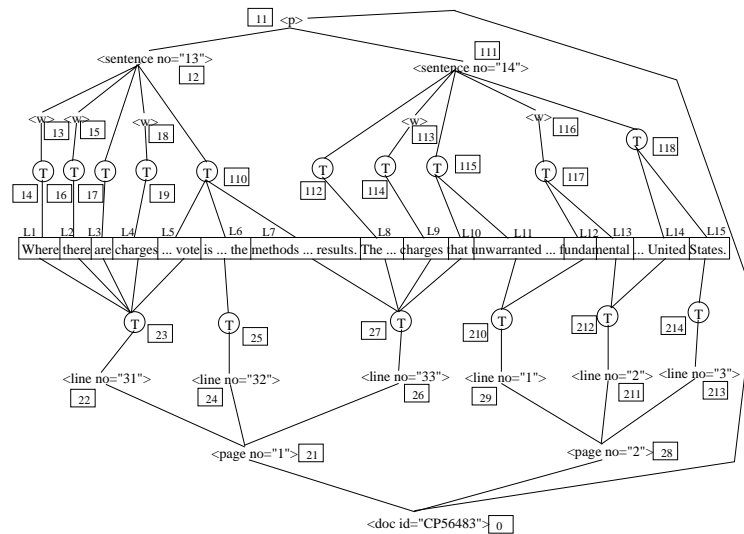


Abbildung 5.1: Beispiel für einen GODDAG. Das Bild zeigt zwei Annotations-hierarchien, die sich jeweils auf denselben Text (Mitte) beziehen. Nicht graphisch dargestellt ist die gemeinsame Wurzel beider Hierarchien. (aus Jacob und Dekhtyar, 2005)

Zum Umgang mit GODDAGs definieren die Autoren elf zusätzliche Achsen:

Die irreflexiven EXPath-Achsen `xancestor` und `xdescendant` sowie deren reflexive Gegenstücke `xancestor-or-self`, `xdescendant-or-self` beziehen die Vorfahren-Beziehung (ähnlich Emu, vgl. Abschnitt 5.1) auf Inklusion auf der Inhaltsebene, d. h. ein Knoten p ist Vorfahr eines Knotens q genau dann, wenn alle von p dominierten Inhalte auch von q dominiert werden. Damit sind diese Beziehungen auch auf Knoten unterschiedlicher Hierarchien anwendbar. Analog werden die Achsen `xfollowing` und `xpreceding` gebildet: Hier muss der gesamte Textinhalt des nachfolgenden Knotens vollständig hinter dem Textinhalt des aktuellen Knotens liegen.

Darüberhinaus werden Achsen für überlappenden Text definiert: `preceding-overlapping` beinhaltet alle Knoten, die mit dem Kontextknoten überlappen und vor diesem beginnen, `following-overlapping` alle Knoten, die mit dem Kontextknoten überlappen und nach diesem enden, `overlapping` ist die Vereinigung jener beiden Achsen.

Schließlich existieren noch »Vereinigungsachsen« xancestor-or-overlapping und xdescendant-or-overlapping.

Desweiteren definieren die Autoren eine Reihe zusätzlicher bzw. an

das neue Datenmodell angepasster Funktionen (vgl. Iacob, 2005).

Eine Beispielimplementierung (die ausschließlich im Arbeitsspeicher arbeitet) existiert.

5.3 TGrep2

TGrep2 (Rohde, 2005) ist ein Werkzeug für die Kommandozeile zum Durchsuchen von Baumbanken, die etwa im Format der Penn Treebank vorliegen.

Patterns in TGrep2 bestehen dabei im Wesentlichen aus Knoten und Relationen zwischen Knoten. Knoten besitzen Namen (wie etwa NP für Nominalphrasen), über die in Anfragen reguläre Ausdrücke möglich sind. Zahlreiche Relationen sind definiert: Etwa für Eltern- (<) und Kindrelation (>, man beachte die gegenüber etwa TIGERSearch genau umgekehrte Verwendung, vgl. Abschnitt 5.5), *n*te und einzige Kinder, Dominanz (Vorfahrenbeziehung, <<), Präzedenz (..) und unmittelbare Präzedenz (.), Geschwisterbeziehungen und zahlreiche Kombinationen daraus.

Darüberhinaus können die zahlreichen Relationssymbole noch modifiziert werden: Durch Voranstellen eines Ausrufezeichens kann man Relationen negieren, durch Nachstellen eines Gleichheitszeichens wird die Relation reflexiv und ein vorangestelltes Fragezeichen macht eine Relation optional.

Über Relationen sind boolesche Ausdrücke möglich. Darüberhinaus können Teilausdrücke mit Labels versehen und später in der Anfrage wiederbenutzt werden, und auch die Definition von Makros ist möglich.

Für gewöhnlich wird der erste Knoten eines Patterns ausgegeben, es ist jedoch auch möglich, andere Knoten zur Ausgabe zu markieren (').

5.4 CQP

CQP (Ewert, 2005) ist das Anfragesystem der IMS Corpus Workbench der Universität Stuttgart.

Die einfachsten CQP-Anfragen sind Wortanfragen, die wahlweise reguläre Ausdrücke enthalten können. Komplexere Anfragen sind Attribut-Wert-Paare, die auch mit booleschen Operatoren miteinander verknüpft werden können: Etwa sucht [(lemma="under.+") & (pos="V.*")]; nach Verben mit dem Präfix »under«. Alles in denselben eckigen Klammern stehende bezieht sich dabei wie etwa auch in TIGER (Abschnitt 5.5) auf denselben Knoten.

Regular Expressions sind dabei auch auf Knotenebene möglich, wie hier anhand () * gezeigt::

```
[pos = "IN"]  
[pos = "DT"] ?  
(  
  [pos = "RB"]  
  [pos = "JJ.*"]  
) *  
[pos = "N.*"] ;
```

Eine besondere Markierung eines Elements des Suchergebnisses ist möglich (@). Knotenlabels (Bezeichner, die an Knotenausdrücke gebunden werden können) ermöglichen komplexere Ausdrücke wie beispielsweise `a:[] "and" b:[] :: a.word = b.word`; für Ausdrücke wie *more and more*.

Das strukturelle Markup der Korpora erfolgt mittels XML. Für strukturelle Anfragen können XML-Tags direkt in die Anfragen eingebaut werden, darüberhinaus existieren eingebaute Funktionen und Makros zur Abfrage von Strukturinformationen: `[lemma="interest"] within np`; etwa sucht nur innerhalb von np-Elementen.

Komplexere Queries können zu Makros zusammengefasst und wiederverwendet werden.

Neben dem oben beschriebenen Anfrageteil der Sprache bietet CQP dabei auch diverse Kommandos zum Laden und Verwalten von Korpora, für Frequenzanalysen, Operationen auf Anfrageergebnissen sowie zur Wahl von Optionen, etwa zur Ergebnisdarstellung. Das Webfrontend der IMS Corpus Workbench wird in bezug auf seine GUI-Funktionen in Abschnitt 3.1.10 auf Seite 34 behandelt.

5.5 TIGERSearch

TIGERSearch (Lezius, 2002; König et al., 2003) ist ein umfangreiches Werkzeug zur Suche in Baumbanken. Ein Baum in TIGERSearch entspricht i. A. einem Satz, die Terminalknoten Wörtern, wobei Kanten Labels tragen und Knoten mit Feature-Wert-Paaren versehen sein können.

Im Gegensatz zu den meisten Syntaxbaum-Tools werden dabei auch kreuzende sowie *sekundäre* Kanten unterstützt. Letztere sind zusätzliche Kanten, die beliebige Knoten verbinden können und somit zur Modellierung sprachlicher Phänomene, die sich nicht in Bäumen ausdrücken lassen, herangezogen werden können (vgl. Abbildung 6.1 auf Seite 72).

Auf sekundären Kanten kann jedoch nicht mit denselben Dominanzoperatoren, wie sie für primäre Kanten zur Verfügung stehen, angefragt werden.

Die Anfragesprache dient zugleich als Korpusdefinitionssprache. Ein Knoten wird dabei als Paar aus Knotenbezeichner und einer Menge von Feature-Wert-Paaren gesehen. Der Ausdruck

```
#n1:[word="das" & pos="ART"]
```

bezeichnet dabei in der Interpretation als Knotenbeschreibungssprache einen Knoten n_1 mit zwei Features, die den Knoten mit dem Wort »das« und dem Part-of-Speech-Tag »ART« annotieren. In der Anfrageinterpretation bezeichnet derselbe Ausdruck ein *Knotenconstraint*: Es trifft auf einen Knoten zu, der ein Feature *word* mit dem Wert »das« und ein Feature *pos* mit dem Wert »ART« besitzt, und bindet diesen an die Variable (#) n_1 .

Zwischen Knoten bzw. Knotenconstraints können Kantenrelationen stehen. Von den Grundrelationen, Dominanz (>) und Präzedenz (.), existieren dabei zahlreiche Varianten, die etwa Transitivität, Transitivität mit gewissen zulässigen Abständen, Negation, bei Dominanz auch Beschriftungen und linke / rechte Ecken angeben. Auch eine Geschwisterrelation (\$) existiert. Anfragen funktionieren dabei im Grunde so, dass man eine Menge von Subgraphen definiert, die in den Graphen des Korpus gesucht werden (dies ist auf einer Teilmenge der Anfragesprache auch graphisch möglich, vgl. Voormann 2002).

Über Konstanten sind reguläre Ausdrücke möglich. Sowohl Bedingungen innerhalb von Knotenconstraints als auch ganze Graphausdrücke können mit Booleschem Operatoren verknüpft werden, Negationen dürfen allerdings keine Variablen überdecken. Schließlich gibt es noch einige Prädikate für bestimmte (Teil-)Grapheneigenschaften wie etwa den Ausgangsgrad.

5.6 CorpusSearch

CorpusSearch (Randall, 2005) ist ein weiteres kommandozeilenbasiertes Tool zum Durchsuchen von Baumbanken, die zur Penn-Treebank kompatibel sind.

Zur Bedienung schreibt man eine Anfragedatei und erhält eine Ausgabedatei mit Treffern und Metainformationen dazu. Diese Ausgabedateien enthalten Metainformationen, die Originaltexte der Treffer, die Treffer im Penn-Format (d. h. Syntaxbäume in einer Klammerstrukturnotation

– dadurch können erneute Anfragen auf den Ausgabedateien gestartet werden) sowie eine Zusammenfassung der Ergebnisse.

Die Anfragesprache sieht eine Reihe von *Suchfunktionen* vor, die grundlegende Relationen wie Dominanz repräsentieren. Argumente zu Suchfunktionen können einfache Wildcards (* bzw. # für Ziffern) beinhalten, negiert werden (!) und Listen von Alternativen (|) beinhalten. Beispielsweise sucht (*VB*|*HV* iPrecedes NP-SBJ*) Knoten eines Typs, der VB oder HV enthält und einem Knoten unmittelbar vorausgeht, dessen Typ mit NP-SBJ beginnt.

Suchfunktionen können mit AND und OR verknüpft sowie negiert werden.

5.7 Xaira

Xaira (Burnard et al., 2005a,b) ist die Weiterentwicklung eines Suchwerkzeugs für das British National Corpus hin zu einem allgemeinen XML-Suchwerkzeug. Dafür ist eine Anfragesprache (CQL 2) in Entwicklung.

In CQL 2 werden die Suchanfragen selbst in XML formuliert (vgl. die Beispielanfragen in Abbildung 5.2 auf der nächsten Seite).

Das Xaira-Datenmodell sieht ein Korpus als eine Hierarchie aus *corpus strings*: einem *base string* und daran angehängten *attached strings*. Letztere können entweder inkludierte Strings sein oder aber Attribute (letztere sind immer Blätter im Attachmentbaum). Suchergebnisse erstrecken sich nie über die Grenzen von Korpusstrings hinweg.

Grundsätzlich kann dabei nach *Formen*, *Lemmata* und benannten Attributen (*addkeys*) gesucht werden. Als boolesche Konstrukte zur Ergebnisverarbeitung stehen Vereinigung und Schnitt zur Verfügung.

Eine Anfrage kann *Sequenzen* spezifizieren. Eine Sequenz kann dabei drei Elementtypen beinhalten: *Queries*, die nach o.g. Formen suchen können und eine untere und obere Grenze für Wiederholungen haben, *Lücken* (*gaps*) und *Negationen*.

Elementanfragen fragen nach XML-Elementen in den Korpusstrings, die Sequenzen (wie oben beschrieben) enthalten.

Darüberhinaus ist es möglich, nach XPath-Mustern zu suchen.

5.8 Weitere Arbeiten

Taylor (2003) untersucht die Verwendung von XSLT als Anfragesprache.

head-Elemente, die »fish« enthalten und nicht in div-Elementen mit dem Attribut »foo« enthalten sind

```
<pattern match="div[@name="foo"]" outside="yes">
  <element name="head">
    <seq><gap/><query><form>fish</form></query><gap/></seq>
  </element>
</pattern>
```

„fish“ in u-Elementen, deren Attribut who dem id-Attribut eines person-Elements mit einem sex-Attributwert von »M« entspricht

```
<element name="u">
  <attribute name="sex" keymap="who"><form>M</form></attribute>
  <seq>
    <gap/>
    <query><form>fish</form></query>
    <gap/>
  </seq>
</element>
```

Sätze, die nur „neither“, jedoch nicht „nor“ enthalten

```
<element name="s">
  <and>
    <seq><gap/><query><form>neither</form></query><gap/></seq>
    <seq><neg><form>nor</form></neg></seq>
  </and>
</element>
```

Abbildung 5.2: Beispielanfragen für Xaira CQL 2 (aus Burnard et al., 2005a)

Die Grenzen von XPath werden dabei durch die Verwendung von XSLT-Kommandos und diversen Template-Konstruktionen überwunden.

Da die Syntax von XSLT recht schreibintensiv und fern vom mentalen Modell der linguistischen Zielgruppe der Anwendungsdomäne ist, hält sie Taylor selbst für ungeeignet zur direkten Benutzung durch die Linguisten und schlägt vor, XSLT lediglich als Backend für Sprachen höherer Ebenen zu verwenden. In ihrer Arbeit illustriert sie Übersetzungen von Emu (vgl. Abschnitt 5.1) und TGrep2 (vgl. Abschnitt 5.3) nach XSLT.

Marx (2004) stellt **CXPath** (Conditional XPath) vor, eine Variante von XPath, die äquivalent zur Prädikatenlogik erster Stufe ist.

Auch die im ersten Teil unter dem Gesichtspunkt der Benutzungsschnittstellen vorgestellten Korpustools bieten Anfragesprachen. Üblicherweise ist dabei die Grundeinheit ein Wort, sodass eine Anfrage der Form *irgendwas* nach Vorkommen des Wortes *irgendwas* sucht. Darüberhinaus ist üblicherweise * als Wildcard und die Kombination von Wörtern möglich.

5.9 Zusammenfassung

Die meisten oben genannten Sprachen beruhen auf einem Konzept von Knoten, die durch Muster spezifiziert werden und Attribute tragen können, und Relationen (ob diese nun Relationen, Suchfunktionen oder Achsen heißen) dazwischen. Üblicherweise gibt es dabei eine Beschränkung auf Bäume, oder eine Unterstützung von DAGs ist wie bei TIGER vorhanden, aber nicht in die sonstigen Anfragekonstrukte integriert.

Die Anfragetools bieten üblicherweise eine einfache Möglichkeit zur Volltextsuche. Insbesondere bei den Sprachen der in Kapitel 3 vorgestellten Webtools ist die Grundform der Anfrage zumeist ein Muster, dass dann auf den Volltext auf ein Wort oder eine Wortfolge gematcht wird.

Die Selektion zur Ausgabe und Kontextmarkierung ist in den existierenden Sprachen relativ unvollkommen. Einige Sprachen bieten die Möglichkeit, einen zur Ausgabe vorgesehenen Knoten in der Anfrage zu Markieren, die Ausgabe selbst besteht üblicherweise entweder aus dem bloßen Ergebnistupel oder in einem fest verdrahteten Kontext (z. B. der Satz bei Baumbank-Anfragewerkzeugen). Dies ist für *DDDquery* nicht ausreichend.

Die meisten Sprachen orientieren sich darüberhinaus an Wörtern, auch dies ist für *DDDquery* nicht genügend (vgl. Lüdeling et al., 2004).

6 Anforderungen an die Anfragesprache

In diesem Abschnitt sollen die Anforderungen zusammengestellt werden, die die zu entwerfende Anfragesprache erfüllen soll. Diese Anforderungen beziehen sich auch auf vorangegangene Veröffentlichungen des DDD-Projekts, so etwa von Faulstich et al. (2005); Dipper et al. (2004); Lüdeling et al. (2004).

Zunächst wird jedoch eine von Lai und Bird durchgeführte Anforderungsanalyse speziell für Baumbanken (Datenbanken für Syntaxbäume) betrachtet:

6.1 Anforderungsanalyse für Baumbanken von Lai und Bird (2004)

Lai und Bird (2004) untersuchen im Rahmen einer Anforderungsanalyse existierende Anfragesprachen für Baumbanken, um daraus Anforderungen an künftige Anfragesprachen in diesem Bereich abzuleiten. Dabei stellen sie sieben Beispielqueries vor (Lai und Bird, 2004, Abb. 1).

Die Autoren betrachten dabei einen Syntaxbaum als geordneten Baum, der jeweils einen Satz präsentiert. Die Terminale bzw. Blattknoten des Baumes sind dabei der eigentliche Text des Satzes, während die Nichtterminale / inneren Knoten syntaktische Kategorien modellieren.

Die hierarchische Organisation der Bäume repräsentiert die syntaktische Struktur bzw. Konstituentenstruktur¹ der zugeordneten Sätze. Dabei gibt es einige über einfache geordnete Bäume hinausgehende Anforderungen:

- verschiedene, nebenläufige (*concurrent*) Hierarchien,²

¹ Eine Konstituentenstruktur ist das Ergebnis einer *Konstituentenanalyse*, bei der nach Glück (2000) ein Gesamtausdruck wie etwa ein Satz in eine Teile zerlegt wird, die wiederum rekursiv weiter zerlegt werden.

² zumeist realisiert als mehrere Bäume, die sich auf Wortebene überschneiden oder gemeinsame Blätter haben

6 Anforderungen an die Anfragesprache

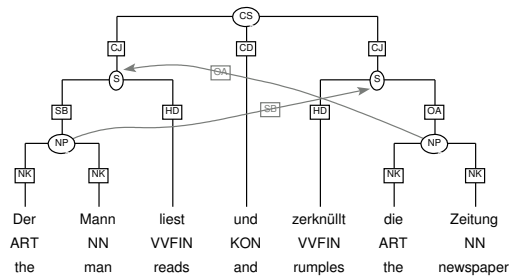


Abbildung 6.1: Sekundäre Kanten für nicht baumförmig darstellbare Sprachkonstrukte (aus Lezius 2002)

- kreuzende Kanten für diskontinuierliche Konstituenten
- sekundäre Kanten oder ähnliche Konstrukte für über Baumförmigkeit hinausgehende Syntax (vgl. Abbildung 6.1)

Die Spezifikation des zu matchenden Teilbaums soll dabei sowohl über die Existenz von Knoten als auch über Beziehungen zwischen Knoten möglich sein, an Beziehungen ist die vertikale Navigation im Raum über Eltern-Kind-Beziehungen bzw. *Dominanz* ebenso wie die horizontale Navigation (*Präzedenz*) notwendig. Beide Beziehungen sollten sowohl in nichttransitiver Form anfragbar sein als auch deren transitive Hülle (also z. B. »a ist Kind von b«, aber auch »a ist Nachfahre von b«), und auch eine Rückwärtsnavigation (z. B. von einem Wort zu dessen Eltern) sowie Negation und Abschlüsse auf möglichst komplexen Strukturen sollten möglich sein.

Navigation sei möglichst auch jenseits der hierarchischen Struktur nötig, so solle man etwa konkrete Wortfolgen ohne Rücksicht auf deren Einbindung in die syntaktische Struktur anfragen können.

Bezüglich der Suchergebnisse sei eine Unterscheidungsmöglichkeit zwischen Knoten, die Teil des Anfrageergebnisses bilden sollen, und Knoten, die lediglich Randbedingungen zur Einschränkung des Suchergebnisses bilden, nötig.

Schließlich schildern die Autoren noch Anforderungen bezüglich Updates, die in dieser Arbeit jedoch nicht behandelt werden sollen.

6.2 Datenmodell

Kernanforderung ist die Unterstützung des angestrebten Datenmodells des DDD-Korpus (vgl. auch Dipper et al., 2004). Dieses Datenmodell

wird im Detail in Kapitel 2 vorgestellt, hier noch einmal die wesentlichen Bestandteile:

- Repräsentationen einer Reihe von Texten, die in unterschiedlichen Fassungen vorliegen können;
- systematische Annotationen, die sich jeweils auf bestimmte Abschnitte (*Spans*) eines Textes beziehen und die in Form eines Baumes oder gerichteten Graphen angeordnet und in verschiedene, miteinander inkompatible Annotationsebenen gruppiert sind;
- Alignierungen zwischen Texten, d. h. verschiedene Textabschnitte aus unterschiedlichen Texten oder auch demselben Text sind miteinander in Beziehung gebracht worden;
- Bibliographische und sonstige Metainformationen, die zur Katalogisierung der einzelnen Texte und ihrer Annotationen dienen und ähnlich den Annotationen in einer hierarchischen Struktur organisiert sind.

6.3 Volltextsuche und Pattern Matching

In einem Textkorpus spielt selbstverständlich die Volltextsuche eine wichtige Rolle. Neben der Suche nach exakt spezifizierten Strings sind auch Pattern-Matching-Verfahren, insbesondere die Suche mit regulären Ausdrücken, wichtig.

Dabei ist zu beachten, dass Wörter als kleinste Einheit der Suchen nicht ausreichend sind. Insbesondere liegen zu den historischen Texten oftmals unterschiedliche Definitionen des Wortbegriffs vor, so können bei manchen Texten etwa physische (durch Leer- oder Satzzeichen getrennte) von logischen Wörtern unterschieden werden.

6.4 Relationen zwischen Spans

Relationen, die zwischen Textabschnitten innerhalb eines Textes bestehen können, müssen auch in Anfragen spezifiziert werden können. Konkrete Relationen sind von Faulstich et al. (2005) bereits spezifiziert worden: Ein Span kann von einem anderen Span *Präfix* oder *Suffix* sein, ihn *enthalten* (oder darin enthalten sein), mit ihm *überlappen*, im Text *vorangehen* oder *folgen* – beides auch unmittelbar angrenzend. Dies ähnelt temporalen Relationen zwischen Zeitintervallen aus temporalen Datenbanken.

6.5 Relationen zwischen Elementen

Zwischen den hierarchisch bzw. graphförmig angeordneten Elementen sollen ebenfalls die möglichen Relationen angefragt werden können. Dies betrifft die Grundrelation »ist Kind von«, aber auch deren transitive (Nachfahrenrelation) und reflexiv-transitive Variante sowie die jeweiligen Umkehrungen davon (Eltern- bzw. Vorfahrenrelationen). Daneben sind ggf. auch komplexere Relationen wie etwa Geschwisterbeziehungen denkbar.

Darüberhinaus müssen Relationen angeboten werden, die Span- und Elementbeziehungen verbinden, etwa das aus XPath bekannte *following-sibling*.

6.6 Spezielle Anforderungen an linguistische Anfragesprachen

In ihrer Untersuchung über existierende Sprachen für Baumbanken beschreiben Lai und Bird (2004) eine Reihe von Anforderungen an Anfragesprachen für Baumbanken (siehe dazu auch Abschnitt 6.1 auf Seite 71). In Bezug auf die Anfragen sind hier insbesondere nichttransitive Varianten der horizontalen Beziehungen (d. h. unmittelbar aufeinanderfolgende Elemente) und die Beschränkung von Teilanfragen auf Teilbäume (*subtree scoping*) relevant (vgl. auch LPath, Abschnitt 5.2.1 auf Seite 62).

6.7 Elemente, Spans etc. in Verbindung

Schließlich wird es nötig sein, die verschiedenen Elemente des Datenmodells in Verbindung setzen zu können. Man muss also den zu einem Annotationselement gehörigen Span oder die zu einem Span gehörenden Annotationselemente erreichen können, anhand einer Annotation in der einen Ebene eine Annotation desselben Spans in einer anderen Ebene finden, zu Spans passende Alignments finden und zu Elementen die zugehörigen Attribute ansprechen können.

6.8 Sequenzoperatoren

Für die Spezifikation komplexerer Graphstrukturen sind Operatoren auf Sequenzen von Relationen, wie sie in Abschnitt 6.5 beschrieben wurden,

wünschenswert. So sollten Sequenzen von Relationen konkateniert, alterniert und konjugiert werden können, auch die Kleenesche Hülle (also die beliebig häufige Wiederholung der Sequenz) ist wünschenswert. Es sollten also reguläre Ausdrücke über Graphstrukturen spezifiziert werden können.

6.9 Aggregierte Inhalte

Es muss möglich sein, Inhalte des Korpus mithilfe spezieller Funktionen zu aggregieren und die aggregierten Ergebnisse in das Anfrageergebnis zu übernehmen. Auf diese Weise können Frontends beispielsweise Frequenztabellen erzeugen (vgl. Abschnitt 3.4.5 auf Seite 43).

6.10 Anfrageergebnis

In Abschnitt 4.2 auf Seite 46 wurden die Daten zusammengetragen, die für die relevanten Darstellungsfähigkeiten der GUIs benötigt werden. Für einige dort genannte Fähigkeiten ist es nötig, entsprechende Auswahlmechanismen in die Sprache einzubauen: Insbesondere muss in einem Suchmuster spezifiziert werden können, was eigentlich zurückzuliefernde Daten sind und welche Teile der Anfrage lediglich Randbedingungen spezifizieren.

Darüberhinaus ist es etwa für die Spezifikation von Kontextinformationen versus dem eigentlich hervorzuhebenden Treffer (etwa für KWIC-Darstellungen, vgl. Abschnitt 4.2.3 auf Seite 47) sinnvoll, zwischen Kontextinformationen und eigentlichem Treffer unterscheiden zu können.

Schließlich sollten weitere Informationen (z. B. aus bestimmten Annotationsebenen) hinzuselektiert werden können, die zur Aufbereitung oder Annotation der Ergebnisdarstellung sinnvoll sind.

6.11 Eigenschaften der Sprachsyntax

Neben den oben aufgeführten Anforderungen daran, was die Sprache *können* muss, gibt es auch noch Anforderungen, wie die Sprache *sein* sollte.

So ist es notwendig, dass die Sprache möglichst einfach erlernbar ist. Dafür sollte sie nicht allzuvielen spezielle Sprachkonstrukte bieten und

sich nach Möglichkeit an vorhandenen Sprachen orientieren, um ggf. vorhandenes Vorwissen der Benutzer auszunutzen.

Neben der Erlernbarkeit ist auch die interaktive Benutzbarkeit im Alltag zu beachten: So sollten insbesondere typische, häufig benötigte Konstrukte nicht allzuviel syntaktischen Overhead erfordern, komplexere Anfragen dagegen sollten glieder- und kommentierbar sein.

Da sich diese Anforderungen teilweise widersprechen, ist ein geeigneter Kompromiss zwischen den verschiedenen Idealen zu finden.

6.12 Übersicht

Die folgende Aufstellung faßt die o. g. Anforderungen noch einmal stichpunktartig zusammen.

- Suche nach Spans
- Corpusauswahl
 - nach bibliographischen Informationen (Katalog)
 - nach automatisch generierten Metadaten
 - Aggregierte Daten
- Suche nach einzelnen Spans durch Spezifikation von
 - Substrings
 - ggf. Fuzzy-Stringsuche (Oracle-Feature lt. Faulstich et al. 2005)
 - regulären Ausdrücken
- Einzelne Elemente
 - durch Typ / Tagname
 - Attributwerte
 - Position relativ zu anderen Elementen
- Relationen zwischen einzelnen Elementen
 - Dominanz, intransitiv, transitiv
 - Präzedenz, intransitiv, transitiv
 - »Eckdominanz«
- Alignments
 - Projektion von Spans über alignierte Textlayer
- Span-Prädikate

- prefix, suffix
- contains, overlaps
- directlyPrecedes, precedes, startsBefore
- concat, intersection
- Relationen
 - Relationen Elemente \leftrightarrow Elemente
 - Relationen Spans \leftrightarrow Elemente
 - Relationen Spans \leftrightarrow Spans
 - Projektionen durch Alignments
 - Relationen Elemente \rightarrow generierter Inhalt (z. B. Aggregationen)
- Sequenzoperatoren
 - Reguläre Ausdrücke über beliebige Prädikate
 - Für Sequenzen à la *Article Adjective⁺ Noun*
 - Konkatenation
 - Alternative
 - Kleenesche Hülle (Sternoperator)
- Boolesche Operatoren (auf Prädikaten)

6 Anforderungen an die Anfragesprache

7 Anfragesprache

Dieses Kapitel stellt die entworfene Anfragesprache vor.

In Abschnitt 7.1 werden dabei zunächst die wesentlichen Designentscheidungen vorgestellt. Die folgenden Abschnitte enthalten dann eine genaue Beschreibung der Elemente der Sprache.

7.1 Grundlagen

In Kapitel 6 auf Seite 71 wurden die Anforderungen an die Anfragesprache beschrieben. Obschon die existierenden Sprachen (vgl. Kapitel 5) diese Anforderungen nicht zu erfüllen vermögen (sie beruhen insbesondere auf anderen Datenmodellen), ist geplant, sich an diesen Sprachen zu orientieren, um den Lernaufwand möglichst zu verringern.

7.1.1 Orientierung an und Abgrenzung von XPath

Dabei ist eine Entscheidung zugunsten XPath (Berglund et al., 2005) als Grundlage für eine eigene Sprachdefinition gefallen. Für XPath sprechen verschiedene Argumente:

- Das XPath zugrundeliegende Datenmodell ist dem von DDD nicht unähnlich. Auch hier gibt es benannte, hierarchisch organisierte Elemente mit Attributen und Textinhalten, und die Navigation durch Strukturen, die zentrale Aufgabe der DDD-Anfragesprache ist, ist zentrales Konzept von XPath.
- XPath ist Teil der Sprachfamilie um XML. Die meisten aktuellen Entwicklungen im Umfeld linguistischer Korpora nutzen XML oder orientieren sich daran (vgl. etwa Bird et al., 2005; Jacob und Dekhtyar, 2005; Ide et al., 2000; Burnard et al., 2005b; TEI-P5), und auch für DDD ist eine umfassende Unterstützung für XML etwa als Austausch- und Exportformat vorgesehen (Dipper et al., 2004).
- Aufgrund der hohen Verbreitung von XPath können bei vielen Nutzern XPath-Kenntnisse angenommen werden; der Lernaufwand

7 Anfragesprache

dieser Benutzer ist beim Aufbau auf eine existierende Sprache geringer.

Die wichtigsten Erweiterungen gegenüber XPath sind:

- die Einführung eines neuen Knotentyps *Span* zur Modellierung der Spans des Datenmodells,
- neue Knotentests des Typs *Span* zur Realisierung der Volltextsuche (es gibt zu diesem Thema in XPath derzeit lediglich ein Anforderungsdokument, Buxton und Rys, 2003),
- neue Achsen sowohl zum Umgang mit den Besonderheiten des eigenen Datenmodells als auch zur Befriedigung besonderer Anforderungen aus dem linguistischen Umfeld (etwa *immediately-following*),
- ein neues Ausgabemodell (vgl. Abschnitt 7.2) zur Umsetzung der komplexeren Anforderungen an die Ergebnisrepräsentation,
- joinfähige Variablen und reguläre Pfadausdrücke zur Formulierung komplexerer Graphstrukturen und zur Beschränkung von Teilausdrücken auf Teilgraphen.

Dagegen werden einige Sprachelemente aus XPath 2 nicht übernommen. Dies betrifft insbesondere Funktionalität, die sich auf die Konstruktion von Knotensequenzen und Inhalten bezieht, da *DDDquery* wie in Abschnitt 7.2 geschildert möglichst auf die Selektion von Daten beschränkt sein soll.

7.1.2 Ausführliche und abgekürzte Syntax

Ähnlich XPath (Clark und DeRose, 1999) soll die Sprache zunächst eine Syntax mit ausführlichen, aber einfachen und semantisch präzisen Schreibweisen bekommen. Mit dieser Syntax sollen in *DDDquery* möglichen Konstrukte ausgedrückt werden können.

Für häufig benötigte Teilanfragen werden dann Kurzschreibweisen eingeführt. Auf diese Weise drücken etwa die Konstrukte *sentence/word* und *element(sentence)/child::element(word)* (in XPath wie in *DDDquery*) dasselbe aus.

Dabei sind – analog zu LPath (Bird et al., 2005) – durchaus mehr Abkürzungen als bei XPath so geplant, dass die fertige Syntax auch nicht

allzuweit von den an Knoten und Relationen orientierten Sprachen wie etwa TIGERSearch (Lezius, 2002) entfernt ist.

Im Verarbeitungsmodell werden zunächst die Abkürzungen zur ausführlichen, normalisierten Syntax expandiert, die dann die Grundlage für die weitere Verarbeitung wie etwa die Umsetzung in eine Anfrage an ein RDBMS bildet.

7.2 Anfrageergebnisse

Wie in Abschnitt 4.2 auf Seite 46 geschildert, ist es nicht ausreichend, eine Menge von Tupeln zurückzuliefern. Andererseits sind die Möglichkeiten, komplex strukturierte Dokumente aufzubauen, wie sie etwa XSLT oder XQuery bieten, so umfangreich, dass sie die Anfragesprache unnötig verkomplizieren würden.

DDDquery hat deshalb als Mittelweg zum Ziel, Teilgraphen des Korpusgraphen auszuwählen, die eine feste Struktur haben (und deren serialisierte Form in weiteren Verarbeitungsschritten etwa mit XSLT anwendungsspezifisch umgeformt werden kann).

Die Konstruktion des Ergebnisgraphen erfolgt dabei wie folgt:

1. Mithilfe der in Abschnitt 7.4 näher beschriebenen Syntax werden *Pfade* im Korpusgraphen beschrieben.
2. Die in Abschnitt 7.4.8 auf Seite 102 beschriebenen Generatoren können neue temporäre Knoten im Korpusgraphen erzeugen – durch Aggregation von vorhandenen Daten. Diese neuen Knoten werden jedoch nicht dauerhaft materialisiert, sondern nur für die Bearbeitung der konkreten Anfrage betrachtet.
3. Mit den in Abschnitt 7.4.6 auf Seite 99 beschriebenen *Markern* werden bestimmte Knoten auf den gewählten Pfaden zur Ausgabe markiert.
4. Der durch die markierten Knoten induzierte Teilgraph wird serialisiert und als Ergebnis der Anfrage zurückgegeben.

7.3 Bibliographische Daten, Headerdaten

Um den Sprachentwurf nicht weiter zu verkomplizieren, wurde entschieden, die im Datenmodell beschriebenen Headerdaten ebenso wie

Annotationen zu behandeln. Das bedeutet, dass für ein Element e , dass einen Textinhalt c tragen kann, der Inhalt c in einem Textobjekt gespeichert und über einen Span mit dem Element e verbunden wird. Dies trifft auch zu, wenn e kein klassisches Annotationselement (also Teil einer Annotationsebene) ist.

7.4 Sprachelemente

In diesem Abschnitt sollen die Bestandteile der Anfragesprache ausführlich vorgestellt werden. Begonnen wird nicht mit dem Startsymbol der Grammatik, einer kompletten Anfrage, sondern mit dem grundlegendsten Konstrukt: Den Pfadmustern.

7.4.1 Pfadmuster

Pfadmuster bilden das zentrale Element der *DDDquery*-Sprachdefinition.

Ein Pfadmuster beschreibt eine Menge von Pfaden im Korpusgraphen. Der Ausgangspunkt eines Pfades wird dabei durch seinen Kontext bestimmt: Absolute Pfade beginnen am Wurzelknoten des Korpus, relative Pfade, wie sie etwa in Prädikaten (vgl. Abschnitt ??) stehen können, am jeweiligen Kontextknoten, Pfade, die mit einer Variablenreferenz (Abschnitt ??) beginnen, am an diese Variable gebundenen Knoten.

7.4.1.1 Semantik

Ein Pfad $p = (v_0, v_1, \dots, v_n)$ ist eine eine Sequenz von Knoten (Nodes) des Korpusgraphen (vgl. Datenmodell, Kapitel 2), in der die aufeinanderfolgenden Knoten v_i, v_{i+1} jeweils in einer bestimmten Relation r_i stehen.

Ein Pfadmuster $\pi = (\sigma_0, \sigma_1, \dots, \sigma_n)$ besteht aus einer Sequenz von Schritten. Jeder dieser Schritte σ_i (siehe für Details dazu Abschnitt 7.4.2) wiederum besteht aus einer Achse α_i und einer Reihe von Filterbedingungen ϕ_i . Die Achse beschreibt die Richtung eines Schritts, die Filterbedingungen beschränken die Menge der Knoten weiter.

Ein Pfadmuster π trifft auf einen Pfad p genau dann zu, wenn für jeden Knoten v_i des Pfades die Filterbedingungen ϕ_i zutreffen, und wenn jede Relation r_i des Pfades durch die Achse α_i des Pfadmusters charakterisiert wird.

Ein Pfad ist dabei tatsächlich als Navigationsbewegung durch das Korpus zu verstehen, d. h. er geht von einem Startpunkt – dem Kontext-

knoten – aus und führt zu einem Zielpunkt: Dem *Zielknoten*. Wenn das Ergebnis eines Pfadausdrucks auf einen Wert reduziert werden soll (der z. B. für einen Vergleichsausdruck zur Verfügung steht), dann wird hier tatsächlich der Wert des Zielknotens bewertet. In einem hypothetischen Ausdruck wie

```
sentence[word/attribute::syllables > 4]
```

in dem der Teilpfad `word/attribute::syllables` mit einer Zahl verglichen werden soll, wird hier also der Wert des Zielknotens (nämlich des Attributes `syllables` des `word`-Elements) zum Vergleich herangezogen.

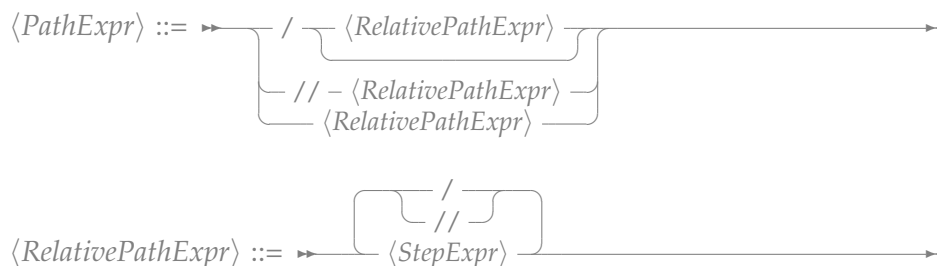
7.4.1.2 Auswertung: Kontext und Focus

Die Auswertung eines Ausdrucks ist abhängig von dessen *Kontext*. Dabei wird wie in XPath zwischen *statischem* und *dynamischen* Kontext unterschieden: Der statische Kontext umfasst alle Informationen, die zum Zeitpunkt der statischen Analyse des Ausdrucks vorliegen, der dynamische Kontext zusätzlich Informationen, die bei der Auswertung des Ausdrucks vorliegen – etwa die Extension der Korpusdatenbank. Von besonderer Bedeutung ist dabei der *Fokus* des Ausdrucks: Dies ist das aktuell verarbeitete Objekt (i. d. R. der *Kontextknoten*, also der aktuell verarbeitete Knoten des Graphen).

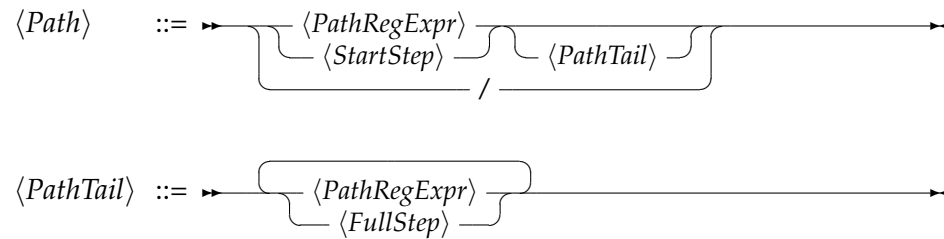
Bestimmte Sprachkonstrukte – namentlich Schritte (vgl. Abschnitt 7.4.2 auf der nächsten Seite) und Prädikate (vgl. Abschnitt 7.4.10 auf Seite 103) – erzeugen einen neuen Fokus, um einen Teilausdruck auszuwerten. In diesem Fall bezeichnet man den Fokus des betreffenden Konstrukts als *äußeren Fokus*, den neu erzeugten Fokus des Teilausdrucks als *inneren Fokus*. Details sind in den diese Konstrukte betreffenden Abschnitten beschrieben.

7.4.1.3 Syntax

Während die Syntax in XPath relativ einfach wie folgt definiert ist:



sieht die Definition in *DDDquery* etwas komplizierter aus:



Die Unterschiede sind im wesentlichen zur Umsetzung der regulären Pfadausdrücke und der Achsensyntax mit Relationssymbolen (Folgeabschnitt) notwendig.

Normalisierte Syntax und Syntaxbaum

In der normalisierten Syntax werden die $\langle \text{PathTail} \rangle$ -Elemente gestrichen und mit ihrer Expansion ersetzt, da sie lediglich der einfacheren Formulierung der Grammatik dienen.

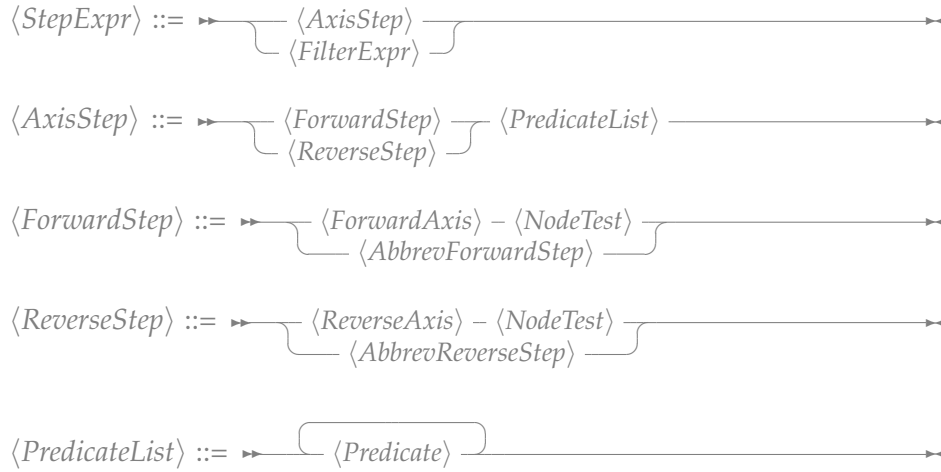
7.4.2 Schritte

Wie oben beschrieben, besteht ein Pfad aus Schritten (und Regulären Pfadausdrücken, auf die weiter unten eingegangen wird). Sieht man von den abkürzenden Schreibweisen ab, so besteht jeder Schritt aus

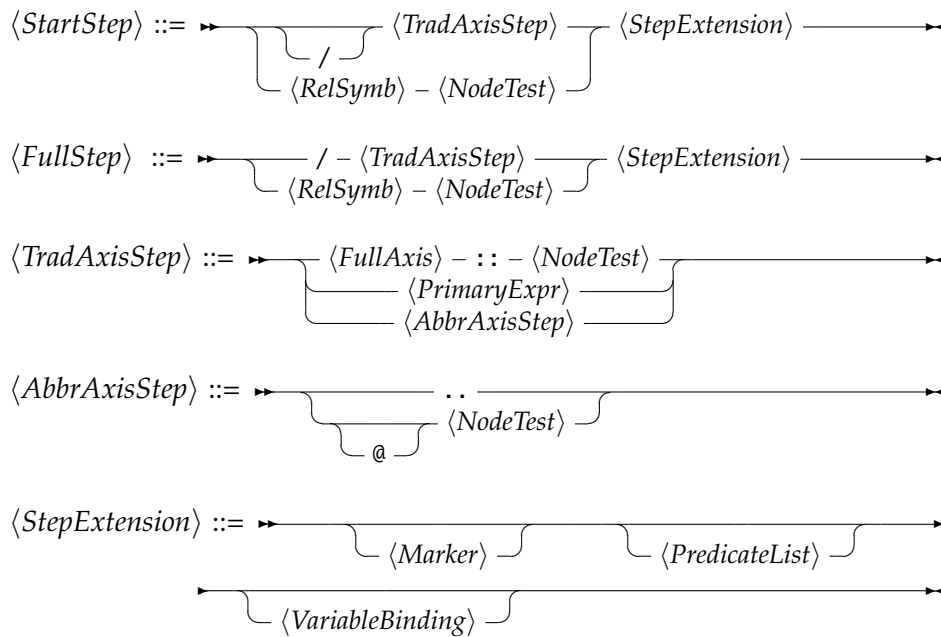
- einer *Achse*, die die Richtung spezifiziert, in die der Schritt navigiert (vgl. Abschnitt 7.4.4 auf Seite 90);
- einem *Knotentest*, der die Menge der selektierten Knoten weiter beschränkt (vgl. Abschnitt 7.4.5 auf Seite 96);
- optionalen erweiternden Komponenten:
 - einem optionalen *Marker*, der die Ausgabe des Knotens beeinflusst (vgl. Abschnitt 7.4.6 auf Seite 99);
 - optionalen *Prädikaten*, mit denen beliebige zusätzliche Bedingungen an die matchenden Knoten gestellt werden können (vgl. Abschnitt 7.4.10 auf Seite 103);
 - einer optionalen Variablenbindung (vgl. Abschnitt 7.4.7 auf Seite 101).

7.4.2.1 Syntax

In XPath:



In DDDquery sieht die Syntax für Schritte wie folgt aus:



Ein $\langle \text{StartStep} \rangle$ ist der erste Schritt eines Pfades, alle weiteren Schritte sind vom Typ $\langle \text{FullStep} \rangle$. Dabei werden zwei Grundformen unterschieden: $\langle \text{TradAxisStep} \rangle$ entspricht der bereits aus XPath bekannten Form eines

Achsenschritts aus vollem Achsenname ($\langle FullAxis \rangle$) und $\langle NodeTest \rangle$ oder einer abgekürzten Form ($\langle AbbrAxisStep \rangle$). Diese traditionelle Form des Schritts wird von seinem Vorgänger durch einen Schrägstrich / getrennt.

Eine mit *DDDquery* neue, verkürzte Schreibweise bietet die Trennung mehrerer Schritte durch Relationssymbole $\langle RelSymb \rangle$ an, die Achse und Schritt-Trennung zugleich repräsentieren und in Abschnitt 7.4.4 ab Seite 90 näher vorgestellt werden. Damit sind die folgenden Schreibweisen äquivalent:

"Eiris" --> "sazun" "Eiris" / following::"sazun"

Normalisierte Syntax und Syntaxbaum

In der normalisierten Syntax fehlen sowohl die $\langle RelSymb \rangle$ -Form als auch die Form $\langle AbbrAxisStep \rangle$. Sie werden durch die vollständige Repräsentation aus $\langle FullAxis \rangle$ und $\langle NodeTest \rangle$ ersetzt.

7.4.3 Typen entlang eines Pfades

Knoten besitzen einen Typ, der i. A. *Element*, *Span* oder *Attribut* ist.

Jede Achse besitzt einen *Kontexttyp* und einen *Zieltyp*, sie navigiert von Knoten des Kontexttyps zu Knoten des Zieltyps. Beispielsweise ist der Kontext- und der Zieltyp der Achse *child* *Element*, die Achse navigiert also von Elementknoten zu Elementknoten. Knotentests besitzen (abgesehen vom allgemeinen Test *node()*) ebenfalls einen Typ.

Damit ein Pfad Sinn ergibt, müssen die jeweils aufeinandertreffenden Typen miteinander kompatibel sein: Einer Achse mit dem Zieltyp *Span* muss also ein Knotentest vom Typ *Span* folgen, und nach einem Schritt mit dem Zieltyp *Element* muss ein Schritt mit (einer Achse mit) dem Kontexttyp *Element* kommen.

Konflikte können beispielsweise entstehen:

- zwischen Achse und Knotentest, z. B.

$$\underbrace{\text{sibling} :: \text{exact-match}(\text{"lenrecht"})}_{\substack{\text{Zieltyp:} \\ \text{Element}}} \quad \underbrace{\text{!}}_{\text{Typ: Span}} \quad (7.1)$$

- zwischen zwei Schritten, z. B.

$$\underbrace{\text{child} :: \text{word}}_{\text{Zieltyp: Element}} \quad \underbrace{\text{!}}_{\text{Kontexttyp: Span}} \quad \underbrace{\text{following} :: \text{exact-match}(\text{"lenrecht"})}_{\text{Kontexttyp: Span}} \quad (7.2)$$

Um die Widersprüche aufzulösen, müssen geeignete *Konvertierungsschritte* eingefügt werden. Dies ist notwendig, da auch im zugrundeliegenden Korpusgraphen ein Schritt notwendig ist, um z. B. von Annotationselement zu Span zu gelangen.

Definition 10 (Span-Relationen)

Seien s und s' Spans. Dann seien folgende Relationen definiert:

$$s \simeq s' \equiv s.tid = s'.tid \wedge s.left = s'.left \wedge s.right = s'.right$$

s entspricht s'

$$s \trianglelefteq s' \equiv s.tid = s'.tid \wedge s.left \geq s'.left \wedge s.right \leq s'.right$$

s ist in s' enthalten

$$s \triangleleft s' \equiv s \trianglelefteq s' \wedge \neg(s \simeq s')$$

s ist in s' echt enthalten

$$s \trianglerighteq s' \equiv s' \trianglelefteq s$$

s enthält s'

$$s \triangleright s' \equiv s' \triangleleft s$$

s enthält s' echt

7.4.3.1 Abbildungen von Annotationselementen zu Spans

Definition 11

Wie oben definiert, kann jedes Annotationselement mit einem Span assoziiert sein. Dabei bezeichne für ein Annotationselement a :

$$span(a) \equiv \begin{cases} \text{falls ein mit } a \text{ assoziierter Span } s \text{ existiert:} & s, \\ \text{sonst:} & \perp \end{cases}$$

Seien \circ eine Span-Relation wie in Definition 10, a, a' mit je einem Span assoziierte Annotationselemente und s ein beliebiger Span, dann gelte

$$a \circ s \equiv span(a) \circ s$$

7.4.3.2 Abbildungen von einem Span zu zugehörigen Annotationsobjekten

Auf einem gegebenen Span s und einer gegebenen Annotationsebene \mathcal{A} lassen sich verschiedene Operationen definieren.

- Ein *genau passendes* Annotationselement, d. h. Text, Anfang und Ende des Elements stimmen mit dem Span überein.

$$A_{\simeq}(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \simeq s\} \quad (7.3)$$

7 Anfragesprache

- Enthaltene Annotationselemente.

$$A_{\triangleleft}(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \triangleleft s\} \quad (7.4)$$

$$A_{\trianglelefteq}(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \trianglelefteq s\} \quad (7.5)$$

- Umschließende Annotationselemente.

$$A_{\triangleright}(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \triangleright s\} \quad (7.6)$$

$$A_{\trianglerighteq}(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \trianglerighteq s\} \quad (7.7)$$

- Die in Richtung »enthalten« bzw. »umschließen« jeweils nächsten Elemente.

$$A_{\trianglelefteq}^!(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \trianglelefteq s \wedge \neg \exists a' : a \triangleleft a' \trianglelefteq s\} \quad (7.8)$$

$$A_{\trianglerighteq}^!(\mathcal{A}, s) \equiv \{a \in \mathcal{A} \mid a \trianglerighteq s \wedge \neg \exists a' : a \triangleright a' \trianglerighteq s\} \quad (7.9)$$

Dabei ist zu beachten, dass die in den Gleichungen (7.8) und (7.9) definierten Operatoren jeweils durchaus eine Menge zurückliefern können. Man beachte dazu das folgende Beispiel:

```
<line id="l1" tid="t1" start="0" end="2">
  <word id="l1w1" tid="t1" start="0" end="2">Bla</word>
</line>
```

Hier ist z.,B. $A_{\trianglelefteq}^!(\mathcal{A}, (t1, 1, 2)) = \{l1, l1w1\}$, da beide Annotationselemente mit demselben Span assoziiert sind.

Wenn man voraussetzt, dass innerhalb einer Annotationsebene ausschließlich baumförmige Annotationen vorkommen, kann man unter Berücksichtigung der Hierarchiebeziehung das einen Span am engsten vollständig umschließende Element wie folgt definieren:

$$a_{\trianglerighteq}^{''}(\mathcal{A}, s) \equiv a \in A_{\trianglelefteq}^!(\mathcal{A}, s) \text{ mit } \neg \exists a' \in A_{\trianglelefteq}^!(\mathcal{A}, s) : a // a' \quad (7.10)$$

Dabei bezeichne $//$ die Vorfahrenrelation.

Für die maximalen in einem Span enthaltenen Elemente findet man jedoch keine eindeutige Definition. Dazu betrachte man etwa:

```
<line id="l1" tid="t1" start="0" end="13">
  <word id="l1w1" tid="t1" start="0" end="2">von</word>
  <word id="l1w2" tid="t1" start="4" end="9">rechen</word>
  <word id="l1w3" tid="t1" start="10" end="16">handen</word>
</line>
```


Man kann hier zwar wiederum Nachfahren ausschließen wie in :

$$A_{\leq}^{!!}(\mathcal{A}, s) \equiv \left\{ a \in A_{\leq}^!(\mathcal{A}, s) \mid \neg \exists a' \in A_{\leq}^!(\mathcal{A}, s) : a // a' \right\} \quad (7.11)$$

Jedoch ist mit dem Beispiel oben etwa

$$A_{\leq}^{!!}(\mathcal{A}, \langle \mathbf{t}1, 4, 16 \rangle) = \{11\mathbf{w}2, 11\mathbf{w}3\},$$

ohne dass eine weitere Reduktionsmöglichkeit besteht.

Schließlich gibt es gleich zwei Definitionsmöglichkeiten für äquivalente Spans:

$$A_{\simeq}^{!!}(\mathcal{A}, s) \equiv \left\{ a \in A_{\simeq}(\mathcal{A}, s) \mid \neg \exists a' \in A_{\simeq}(\mathcal{A}, s) : a // a' \right\} \quad (7.12)$$

$$A_{\simeq}^{ii}(\mathcal{A}, s) \equiv \left\{ a \in A_{\simeq}(\mathcal{A}, s) \mid \neg \exists a' \in A_{\simeq}(\mathcal{A}, s) : a' // a \right\} \quad (7.13)$$

Die Variante aus Gleichung (7.12) wählt jeweils die »höchsten«, die aus Gleichung (7.13) die »niedrigsten« Knoten aus.

Es existiert also keine so eindeutige Abbildung $Span \times Annotationsebene \mapsto Element$ wie für den umgekehrten Fall, solange man von beliebigen Spans ausgeht.

Für den Weg von Element zu Span, wie in den Beispielen oben nötig wäre, ist die Lösung eindeutig, da mit jedem Element maximal ein Span assoziiert sein kann (vgl. Abbildung 2.2 auf Seite 18), und so wird einfach eine Achse $element-span : \langle Element \rangle \rightarrow \langle Span \rangle$ definiert.

Für den umgekehrten Weg jedoch existieren verschiedene Möglichkeiten (vgl. Abschnitt 7.4.3.2 auf Seite 87), die in Tabelle 7.1 auf der nächsten Seite dargestellt sind.

7.4.3.3 Automatisch eingefügte Konvertierungsschritte

Um einerseits die Anwender nicht mit rein syntaktisch begründeten Fehlermeldungen zu belasten und zudem Schreibarbeit zu sparen, ist es sinnvoll, für Fälle wie in (7.1) und (7.2) automatisch »sinnvolle« Konvertierungsschritte einzufügen. Dabei wird für den Fall $\langle Element \rangle \rightarrow \langle Span \rangle$ die einzig mögliche Achse $element-span$ und für $\langle Span \rangle \rightarrow \langle Element \rangle$ die Achse $matching-element$ gewählt.

Achse	Relation ^a	Beschreibung
matching-element	A_{\simeq}	Span entspricht Element-Span
containing-element	A_{\supseteq}	Element-Span enthält Span
	$A_{\supseteq}^!$	dito, aber am engsten
contained-element	$A_{\subseteq}^!$	Span enthält Element-Span
	$A_{\subseteq}^!$	dito, aber am engsten

Tabelle 7.1: Achsen für den Weg von Span nach Element

^a siehe Abschnitt 7.4.3.2 auf Seite 87

Aus Beispiel (7.1) wird so etwa

$$\underbrace{\text{sibling}::\text{element}() / \text{element-span}::\text{exact-match("lenrecht")}}_{\substack{\text{Zieltyp:} \\ \text{Element}}} \quad \text{Typ: Span} \quad (7.14)$$

und aus (7.2)

$$\underbrace{\text{child}::\text{word}}_{\text{Zieltyp: Element}} / \text{element-span}::\text{span}() / \underbrace{\text{following}::\text{"lenrecht"}}_{\text{Kontexttyp: Span}} \quad (7.15)$$

7.4.4 Achsen

Achsen geben die Richtung an, in der durch den Korpusgraphen navigiert wird.

Eine Achse ist derjenige Teil eines Schrittes, der für die Erzeugung eines neuen, inneren Fokus F_i zuständig ist. Wie in Abschnitt 7.4.1.2 auf Seite 83 beschrieben, wird ein Schritt für jeden Knoten v_o im äußeren Fokus F_o des Schrittes ausgeführt. Jede Achse repräsentiert eine bestimmte Achsenrelation R . Zur Erzeugung der Knotensequenz im inneren Fokus des Schrittes werden nun alle Knoten des Graphen gewählt, die mit v_o in der Achsenrelation R stehen.

Als Beispiel betrachten wir die Achse `child` im Teilpfad

$$\text{//verse/} \quad \begin{array}{c} F_o \\ \downarrow \\ \text{child}::\underbrace{\text{element(word)}}_{\sigma} \end{array} \quad \begin{array}{c} F_i \\ \downarrow \end{array}$$

Im äußeren Fokus des Schrittes σ befinden sich hier die Menge V aller verse-Elemente. Für jedes verse-Element $v \in V$ wird nun die Menge $\text{child}(v)$ aller Knoten berechnet, die zu v in der Kindrelation stehen. Die Vereinigung dieser Mengen ist die Kontextsequenz im inneren Fokus der child-Achse, die dann mit dem Knotentest `element(word)` weiter gefiltert wird.

$$F_i = \{u \mid \exists v \in V : u \in \text{child}(v)\}$$

7.4.4.1 Typen einer Achse

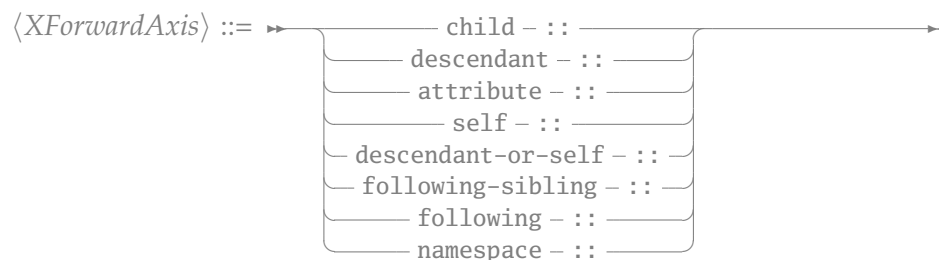
Jede Achse hat einen Ziel- und einen Kontexttyp. Der Zieltyp entspricht dem *principal node kind* in XPath, es ist der Typ, den die Knoten auf der Achse (d. h. in deren inneren Fokus) haben. Der Kontexttyp ist der Typ, der von den Kontextknoten im äußeren Fokus erwartet wird, um die Achse anwenden zu können.

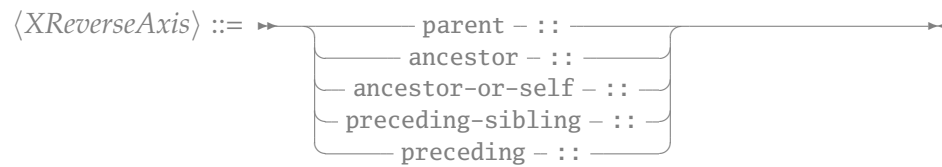
Als Ziel- und Kontexttypen der Achsen sind die im Datenmodell vorgesehenen Typen möglich. Eine Achse a heißt *finale Achse*, wenn keine Achse a' existiert, deren Kontexttyp dem Zieltyp von a entspricht. Die Attributachse ist eine finale Achse.

Im Gegensatz zu XPath können Achsen parametrisiert werden – das ist z. B. für die Alignment-Achsen nötig, die Parameter für die Rollen und die Greediness akzeptieren (vgl. Abschnitt 3.4.4 auf Seite 42). Darüberhinaus sind abkürzende Schreibweisen für Achsen bzw. für die Kombination von Achse und dem Schritt-Trennzeichen / möglich.

7.4.4.2 Achsen im Vergleich zu XPath

XPath kennt die folgenden Achsen:





Mit Ausnahme der Namespace-Achse, die in *DDDquery* nicht umgesetzt und in XPath 2.0 ohnehin als veraltet gilt, sind diese Achsen auch in *DDDquery* vorhanden.

Wie der nachfolgenden Aufstellung zu entnehmen, können einige der zusätzlichen *DDDquery*-Achsen parametrisiert werden. Die Parameter werden in diesem Falle in Klammern und durch Kommata getrennt hinter der Achse (jedoch vor den ::) notiert.

Nicht von XPath übernommen ist die Unterscheidung von vorwärts- und rückwärtsgerichteten Achsen in der Grammatik; diese Unterscheidung ist ohnehin nicht problemlos auf alle Achsen (layer) anwendbar. Stattdessen ist die Richtung eine bloße Eigenschaft einer Achse, ebenso wie Kontext- und Zieltyp.

Darüberhinaus gibt es neben den XPath-üblichen abgekürzten Achsen-schritten für einige häufig benötigte Achsen wie in LPath (Bird et al., 2005) eine zusätzliche Kurzschreibweise in der Form relationaler Symbole, die zugleich als Trenner zwischen zwei Schritten gilt und hier den / ersetzt.

7.4.4.3 Verfügbare Achsen

Im folgenden werden die möglichen Achsen zur Navigation durch den Kontextgraphen aufgeführt. In der Semantikdefinition ist die Schreibweise *a* achse *b* zu lesen als »*a* ist Knoten im äußeren Fokus der Achse und *b* ist Knoten im inneren Fokus der Achse«, im Übrigen werden die im Datenmodell in Abbildung 2.2 auf Seite 18 beschriebenen Attribute, Assoziationen und Methoden verwendet.

Die Achsenbeschreibungen folgen der Form:

Achsenname(<i>Argumente</i>) : <Kontexttyp> → <Zieltyp>	Kurzform
Verbale Beschreibung	
Formale Beschreibung	

Achsen zur Navigation in Elementhierarchien

Vertikale Navigation

$\text{child} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ a / b

Zielknoten sind Kindknoten des Kontextknotens
primitive Relation child im Datenmodell

$\text{parent} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a \setminus b$

Zielknoten sind Elternknoten des Kontextknotens
 $a \text{ parent } b \iff b \text{ child } a$

$\text{descendant} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a // b$

Zielknoten sind Nachfahren des Kontextknotens
 $a \text{ descendant } b \iff a \text{ child } b \vee \exists c_1, \dots, c_n : a \text{ child } c_1 \wedge c_1 \text{ child } c_2 \wedge \dots \wedge c_n \text{ child } b$

$\text{ancestor} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a \setminus \setminus b$

Zielknoten sind Vorfahren des Kontextknotens
 $a \text{ ancestor } b \iff b \text{ descendant } a$

Horizontale Navigation

$\text{sibling} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a \wedge b$

Zielknoten sind Geschwister des Kontextknotens
 $a \text{ sibling } b \iff \exists p : p \text{ child } a \wedge p \text{ child } b$

$\text{following} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \rightarrow t$

Zielspan folgt nach Kontextspan
 $s \text{ following } t \iff t.\text{left} > s.\text{right} \wedge s.\text{text} = t.\text{text}$

$\text{following} : \langle \text{AnnElement} \rangle \rightarrow \langle \text{AnnElement} \rangle$ $a \rightarrow b$

Mit Zielelement assoziierter Span folgt auf mit Kontextelement assoziiertem Span
 $a \text{ following } b \iff a.\text{span} \text{ following } b.\text{span}$

$\text{preceding} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \leftarrow t$

$s \text{ preceding } t \iff t.\text{right} < s.\text{left} \wedge s.\text{text} = t.\text{text}$

$\text{preceding} : \langle \text{AnnElement} \rangle \rightarrow \langle \text{AnnElement} \rangle$ $a \leftarrow b$

$a \text{ preceding } b \iff a.\text{span} \text{ preceding } b.\text{span}$

$\text{following-sibling} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a \rightarrow \wedge b$

$a \text{ following-sibling } b \iff a \text{ following } b \wedge a \text{ sibling } b$

$\text{immediately-following-sibling} : \langle \text{AnnElement} \rangle \rightarrow \langle \text{AnnElement} \rangle$ $a \rightarrow \wedge b$

$a \text{ immediately-following-sibling } b \iff a \text{ immediately-following } b \wedge a \text{ sibling } b$

$\text{preceding-sibling} : \langle \text{Element} \rangle \rightarrow \langle \text{Element} \rangle$ $a \leftarrow^{\wedge} b$
 $a \text{ preceding-sibling } b \iff a \text{ preceding } b \wedge a \text{ sibling } b$

$\text{immediately-preceding-sibling} : \langle \text{AnnElement} \rangle \rightarrow \langle \text{AnnElement} \rangle$ $a \leftarrow^{\wedge} b$
 $a \text{ immediately-preceding-sibling } b \iff$
 $a \text{ immediately-preceding } b \wedge a \text{ sibling } b$

Span-Relationen

$\text{immediately-following} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \rightarrow t$
 $s \text{ immediately-following } t \iff s.\text{right} = t.\text{left} \wedge s.\text{text} = t.\text{text}$

$\text{immediately-preceding} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \leftarrow t$
 $s \text{ immediately-preceding } t \iff s.\text{right} = t.\text{left} \wedge s.\text{text} = t.\text{text}$

$\text{contained} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \rightsquigarrow t$
 Zielspan ist in Kontextspan enthalten
 $s \text{ contained } t \iff s.\text{left} \leq t.\text{left} \wedge s.\text{right} \geq t.\text{right} \wedge s.\text{text} = t.\text{text}$

$\text{containing} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$ $s \rightsquigarrow^< t$
 Zielspan enthält Kontextspan
 $s \text{ containing } t \iff t \text{ contained } s$

$\text{prefix} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$
 Zielspan ist ein Präfix von Kontextspan
 $s \text{ prefix } t \iff t.\text{left} = s.\text{left} \wedge t.\text{right} \leq s.\text{right} \wedge s.\text{text} = t.\text{text}$

$\text{suffix} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$
 Zielspan ist ein Suffix von Kontextspan
 $s \text{ suffix } t \iff t.\text{right} = s.\text{right} \wedge t.\text{left} \geq s.\text{left} \wedge s.\text{text} = t.\text{text}$

$\text{overlapping-following} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$
 Zielspan überlappt mit Kontextspan und geht nach Ende des Kontextspans weiter
 $s \text{ overlapping-following } t \iff s.\text{left} < t.\text{left} \wedge s.\text{right} < t.\text{right}$

$\text{overlapping-preceding} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$
 $s \text{ overlapping-preceding } t \iff t \text{ overlapping-following } s$

$\text{whole-text} : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle$
 $s \text{ whole-text } t \iff t.\text{left} = s.\text{text.left} \wedge t.\text{right} = s.\text{text.right} \wedge s.\text{text} = t.\text{text}$

Selektiert den Span, der sich über den gesamten zum Kontextspan gehörenden Text erstreckt.

Navigation von Elementen zu anderen Knotentypen

$\text{element-span} : \langle \text{Element} \rangle \rightarrow \langle \text{Span} \rangle$ ²

Zielspan ist mit Kontextknoten assoziierter Span
primitive Assoziation im Datenmodell

$\text{attribute} : \langle \text{Element} \rangle \rightarrow \langle \text{Attribute} \rangle$ ^{@¹}

Zielknoten ist ein Attribut namens $\langle \text{Name} \rangle$ des Kontextelements
Relation im Datenmodell

$\text{layer}(\langle \text{Name} \rangle) : \langle \text{Span} \rangle \rightarrow \langle \text{Element} \rangle$

Zielknoten ist Annotation des Spans in Annotationsebene $\langle \text{Name} \rangle$
 $s \text{ layer}(n)e \iff e \text{ element-span } s \wedge e.\text{layer.name} = n$

$\text{alignment}(\langle \text{Rolle} \rangle, \langle \text{Rolle} \rangle[, \langle \text{Greed} \rangle[, \langle \text{Greed} \rangle]]) : \langle \text{Span} \rangle \rightarrow \langle \text{Align} \rangle$

Zielknoten ist align-Element mit Rolle $\langle \text{Rolle} \rangle$, das mit dem Kontext-
span assoziiert ist (vgl. Abschnitt 3.4.4)
 $s \text{ alignment}(r)a \iff s \in a.\text{span} \wedge a.\text{role} = r$

$\text{aligned}(\langle \text{Rolle} \rangle, \langle \text{Rolle} \rangle[, \langle \text{Greed} \rangle[, \langle \text{Greed} \rangle]]) : \langle \text{Span} \rangle \rightarrow \langle \text{Span} \rangle \quad s \sim (r_s, r_t[, g[, g_r]])$

Zielspan ist über die beiden Rollen mit Kontextspan aligniert, dabei
werden die Greediness-Angaben g und g_r für den linken bzw. rechten
Rand des Quellspans beachtet. ist g_r nicht angegeben, so sei $g_r := g$,
ist g nicht angegeben, so sei $g_r := g := \text{non-greedy}$.

7.4.4.4 Die layer-Achse: Assoziationen zwischen Annotationsebenen

Assoziationen zwischen unterschiedlichen Annotationselementen in unterschiedlichen Annotationsebenen werden über gemeinsam genutzte Spans definiert. Dies kommt der Bedeutung der Annotationsebenen (nämlich unterschiedliche Annotationen desselben Texts zu sein) am nächsten.

Definition 12

Seien $\mathcal{A}_1, \mathcal{A}_2$ Annotationsebenen, $a_1 \in \mathcal{A}_1, a_2 \in \mathcal{A}_2$ Annotationselemente und $s_1 = \text{span}(a_1), s_2 = \text{span}(a_2)$ Spans, dann gelte

$$a_1 \approx_{\mathcal{A}_2} a_2 \iff s_1 \simeq s_2 \quad (7.16)$$

d. h. a_2 sei ein a_1 entsprechendes Annotationselement in Annotationsebene \mathcal{A}_2 .

¹ ggf. automatisch, vgl. Abschnitt 7.4.3.3 auf Seite 89

² bzw. /@. Dies ist eine abkürzende Schreibweise gemäß XML-Syntax, vgl. Abschnitt 7.4.2 auf Seite 84

7 Anfragesprache

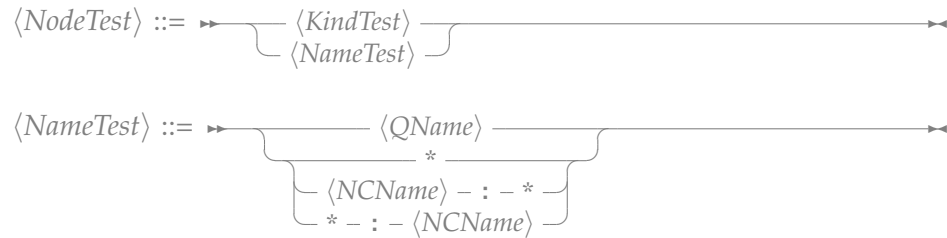
Die entsprechende *DDDquery* -Schreibweise ist dann $a_1/\text{layer}(\mathcal{A}_2) :: \text{element}()$, wobei für \mathcal{A}_2 der Name der entsprechenden Annotationsebene einzusetzen ist.

Die in den Abschnitten 7.4.3.1 und 7.4.3.2 aufgeführten Anmerkungen für die Abbildungen von Annotationselementen zu äquivalenten Spans und umgekehrt treffen auch hier zu.

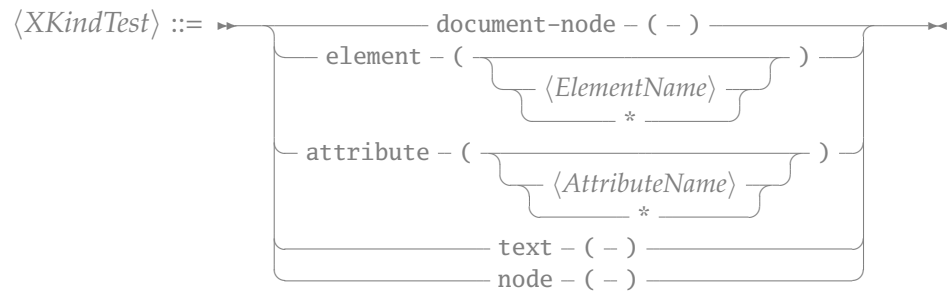
7.4.5 Knotentests

Knotentests sind wie in XPath grundlegende Bedingungen, die von allen in einem Schritt selektierten Knoten erfüllt werden müssen. Die Unterscheidung zwischen Knotentests und Prädikaten wird aus Kompatibilitätsgründen von XPath übernommen.

XPath sieht Tests auf Namen und Knotentyp vor:



In XPath sind, wenn man diverse Tests auf Typen aus Schemata und von unserem Datenmodell nicht unterstützte Typen wie Processing Instructions oder Kommentare weglässt, die folgenden Knotentests auf Typen möglich:



Hier wird in *DDDquery* zusätzlich ein generischer Test $\text{span} - (-)$ für Spans eingeführt. Zu beachten ist dabei, dass dieser Test nur anderweitig (also durch eine Volltextsuche, an ein Element oder ein Alignment) gebundene Spans liefert.

7.4.5.1 Volltextsuche

Darüberhinaus werden die Knotentests für die Realisierung der Volltextsuche benutzt, indem *Volltextmuster* ($\langle \text{TextPattern} \rangle$) zulässige Knotentests sind. Ein Span erfüllt einen solchen Knotentest also genau dann, wenn das Muster vollständig auf diesen Test zutrifft.

Gerade für die Muster der Volltextsuche werden abkürzende Schreibweisen vorgesehen, so dass man etwa "sazun" für `exact-match("sazun")` oder `r"id[ei].*"` für `re-match("id[ei].*")` schreiben kann.

Die Grammatik für Knotentests in *DDDquery* ist in Abbildung 7.1 dargestellt.

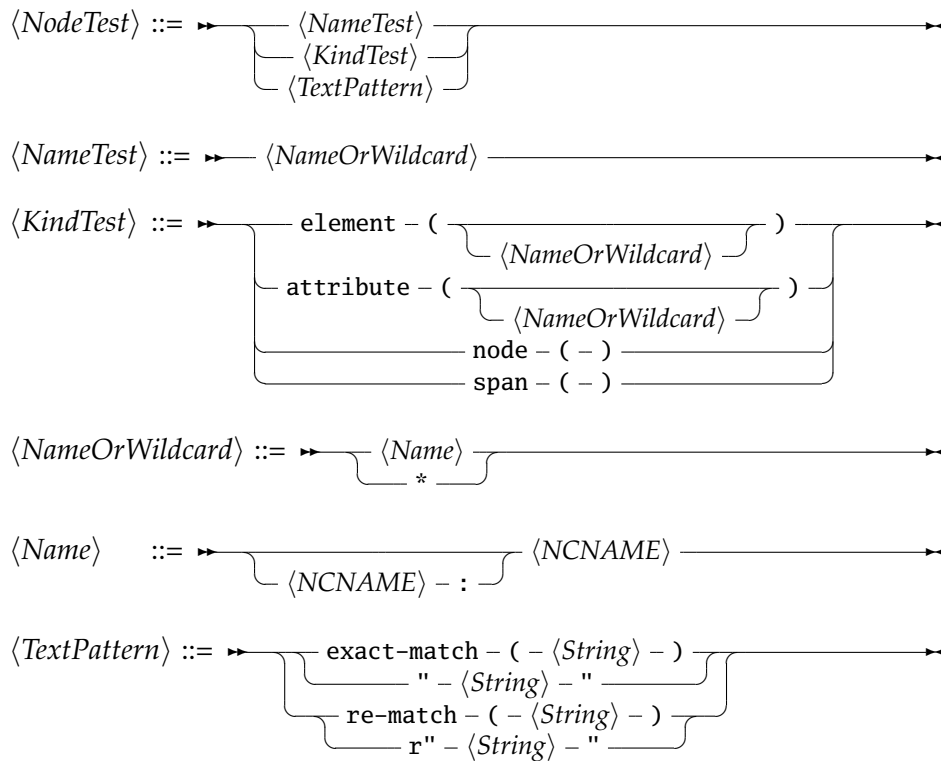


Abbildung 7.1: Grammatik für Knotentests

Natürlich kann ein solcher Test mit den Span-Achsen verbunden werden:

```
//word/element-span::exact-match("Eiris")
```

7 Anfragesprache

trifft so beispielsweise nur auf solche Spans zu, die exakt »Eiris« lauten und mit einem word-Element assoziiert sind;

```
//verse/contained::exact-match("Eiris")
```

dagegen auf alle Spans, die »Eiris« lauten und in einem verse-Element enthalten sind, und

```
//word/element-span::span()/suffix::r"r[ie]s"
```

auf alle Spans, die »ris« oder »res« lauten und Suffix eines word-Elements sind.

Weitere Knotentests für Textmuster sind denkbar.

Im diesem Zusammenhang beachte man bitte auch die Typkonvertierung für nicht zusammenpassende Typen in Pfadausdrücken, die in Abschnitt 7.4.3.3 vorgestellt wurde.

Die in Kapitel 3 vorgestellten Tools bieten üblicherweise (ausschließlich) eine Volltextsuche nach Wörtern an, und auch die Volltextindexe von Oracle Text basieren auf Wörtern (wobei die genaue Interpretation von »Wort« durchaus benutzerdefinierbar ist). Andererseits werden Wortsuchen nicht für alle Aufgaben des Projekts genügen. Eine Suche wahlweise nach Wörtern oder nach beliebigen Textausschnitten erscheint deshalb sinnvoll.

Normalisierte Syntax und Syntaxbaum

In der normalisierten Syntax gibt es nur die Langformen, nicht die Kurzschreibweisen.

7.4.5.2 Verfügbare Knotentests

Die Knotentests arbeiten immer auf einer Menge von Knoten (die in der Semantikbeschreibung als Index zum Knotentest angegeben ist), die von der Achse des zugehörigen Schrittes ausgewählt wurde, und beschränken diese weiter, d. h. wählen eine Teilmenge daraus aus.

`element($\langle Name \rangle$) : $\langle Element \rangle$ $\langle Name \rangle$`

Selektiert Elemente des angegebenen Namens.

$element_E(n) = \{e \in E \mid e.name = n\}$

`attribute($\langle Name \rangle$) : $\langle Attribute \rangle$ $\langle Name \rangle$`

Selektiert Attribute des angegebenen Namens.

$attribute_A(n) = \{a \in A \mid a.name = n\}$

`* : $\langle Node \rangle$ *`

Wählt alle Knoten (des Zieltyps der Achse)

$*_N = N$

<code>exact-match($\langle String \rangle$) : $\langle Span \rangle$</code>	<code>"$\langle String \rangle$"</code>
Wählt alle Spans, deren Textwert exakt $\langle String \rangle$ entspricht $exact-match_S(p) = \{s \in S \mid s.string() = p\}$	
<code>re-match($\langle String \rangle$) : $\langle Span \rangle$</code>	<code>r"$\langle String \rangle$"</code>
Wählt alle Spans, deren Textwert vollständig auf den regulären Ausdruck in $\langle String \rangle$ matcht.	
<code>wild-match($\langle String \rangle$) : $\langle Span \rangle$</code>	<code>m"$\langle String \rangle$"</code>
Wählt alle Spans, deren Textwert auf das Muster in $\langle String \rangle$ passt; wobei * für beliebig viele beliebige Zeichen und _ für ein beliebiges Zeichen steht	

7.4.6 Kontextauswahl und Treffermarkierungen (Marker)

Wie in Abschnitt 4.2 geschildert, ist es i. A. nicht ausreichend, nur die durch den Pfad markierten Zielknoten als Ergebnis zurückzuliefern. Stattdessen sind sinnvoll

1. Strukturinformationen, die das Ergebnis einzuordnen helfen,
2. Textinhalte, die einen Kontext des Ergebnisses darstellen.

Für 1 ist als einfache Lösung möglich, alle Vorfahren hinzuzuselektieren; damit erreicht man jedoch eine komplexe Schreibweise. Eine weitere Beschränkung kann dann durch die Nachbearbeitungen durch die GUI-spezifischen XSLT-Skripte erfolgen.

Um jedoch die Größe des Ergebnisses zu beschränken, ist es sinnvoller, einzelne Knoten in einem Pfadmuster zur Ausgabe zu markieren und zu annotieren. Die Syntax ist in Tabelle 7.2 auf der nächsten Seite zu finden.

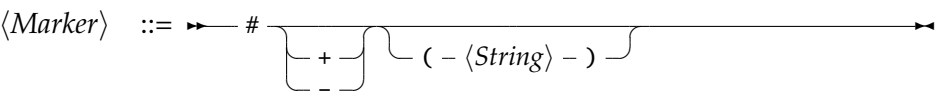
Dabei wurden benannte Marker der Form $\#(- \langle Name \rangle -) \rightarrow$

Zu beachten ist dabei, dass nur Knoten auf Pfaden selektiert wurden, deren Pfadmuster vollständig angewandt wurde. Man betrachte beispielsweise die Anfrage:

```
//word[@pos='ADV']#/"geht"
```

Angenommen, es existieren zwar word-Elemente, auf die das Prädikat `@pos="ADV"` zutrifft, jedoch keine solchen, deren Inhalt "geht" lautet, so wird hier trotz des Markers beim word nichts in das Ergebnis übernommen, da der gesamte Pfad keinen Treffer erzielt.

Mehr zur Semantik der Anfrageergebnisse ist in Abschnitt 7.2 auf Seite 81 zu finden. Einem Beispiel, dass den Einsatz von Markern in



Marker	Beschreibung
#	einfacher Marker: Knoten wird ausgegeben
#(<Name>)	gibt Knoten aus und markiert ihn mit Wert $\langle \text{Name} \rangle$
#+	alle Knoten bis zum nächsten #- werden ausgegeben (optional mit $\langle \text{String} \rangle$)
#-	beendet Ausgabe der Knoten

Tabelle 7.2: Syntax zur Ausgabe und Markierung von Knoten

Ausdruck	Beschreibung
#+//word//*	selektiert alle word-Elemente mit allen Vor- und Nachfahren
//word["Eiris"]# \\ * #+(context) \\ page	Alle word-Elemente, deren Span auf »Eiris« lautet. Außerdem alle Vorfahren bis einschließ-lich page, diese aber mit »context« markiert.

Tabelle 7.3: Beispiele für Ausdrücke mit Markern

Verbindung mit den Anfrageergebnissen demonstriert, ist Abschnitt 4.3.3 ab Seite 58 gewidmet.

Marker innerhalb von Prädikaten sind nicht vorgesehen.

Normalisierte Syntax und Syntaxbaum

In der normalisierten Syntax fehlen die Konstrukte mit #+ und #-, stattdessen ist jeder Knoten mit dem entsprechenden Marker annotiert.

7.4.7 Variablen / Joins

XPath sieht Variablenreferenzen in seiner Syntax vor:

$\langle VarRef \rangle ::= \$ \langle VarName \rangle$

$\langle VarName \rangle ::= \langle QName \rangle$

Variablen können in XPath entweder durch *for*-Ausdrücke oder Quantoren gebunden werden – beides ist in *DDDquery* allerdings nicht vorgesehen – oder Variablen der *Host Language* (etwa XSLT) entsprechen, die Konstrukte zur Variablenbindung liegen hier ausserhalb von XPath (etwa durch das `xsl:variable`-Element in XSLT (Kay et al., 2003)).

In *DDDquery* sollen Variablen jedoch auch zum Ausdruck komplexerer Graphstrukturen genutzt werden. Deshalb gibt es die folgenden Ergänzungen gegenüber XPath:

- Eine Variablenreferenz kann einem Schritt, der nicht in einem Prädikat steht, hinzugefügt werden.

In der Syntax:

$\langle FullStep \rangle ::= \begin{array}{c} \text{---} / - \langle TradAxisStep \rangle \text{---} \langle StepExtension \rangle \text{---} \\ \quad \quad \quad \underbrace{\quad \quad \quad}_{\langle RelSymb \rangle - \langle NodeTest \rangle} \end{array}$

$\langle StepExtension \rangle ::= \begin{array}{c} \text{---} \underbrace{\quad \quad \quad}_{\langle Marker \rangle} \underbrace{\quad \quad \quad}_{\langle PredicateList \rangle} \text{---} \\ \quad \quad \quad \underbrace{\quad \quad \quad}_{\langle VariableBinding \rangle} \end{array}$

$\langle VariableBinding \rangle ::= \text{---} \$ - \langle Name \rangle \text{---}$

Für jeden Treffer im Suchergebnis müssen dabei alle Schritte, die an dieselbe Variable gebunden werden, auf denselben Knoten matchen.

7.4.8 (Aggregations-)funktionen

Neben dem reinen Abfragen von Subgraphen ist auch das Generieren bzw. Aggregieren von Inhalten notwendig. Dies betrifft beispielsweise Aggregationsfunktionen zur Darstellung von Frequenztabellen, wie sie von zahlreichen Korpustools erzeugt werden können (vgl. Abschnitt 3.4.5 auf Seite 43).

Zur Darstellung empfiehlt sich der Einbau von Funktionen in die Sprache. Zum einem ist die Funktionssemantik naheliegend – eine Funktion liefert das Ergebnis der Bearbeitung einer Menge von Argumenten, was der Semantik von Aggregationen entspricht, darüberhinaus wird eine Funktionssyntax auch in anderen Sprachen wie XPath verwendet, so dass durch eine Übernahme dieser Syntax vorhandenes Wissen der Benutzer weiterverwendet werden kann.

Als Verallgemeinerung dazu sind in der Anfragesprache *Generatoren* Sprachelemente, die neue Inhalte generieren – im Allgemeinen auf der Basis in der Datenbank vorhandener Informationen.

Die Syntax ist gegenüber XPath nicht verändert (abgesehen von den Abweichungen bei Ausdrücken, vgl. Abschnitt A.2):

$$\langle \text{FunctionCall} \rangle ::= \text{Name} - (\langle \text{Expression} \rangle)$$

So könnte etwa folgender Pseudo-Ausdruck

```
document[@typ="H"]/text[@fassung="n"]/count(./word[@lemma="renn"])
```

eine Liste aller Dokumente des Typs *H* mitsamt der Anzahl der Vorkommen von Wörtern mit dem Lemma *renn* in Texten der Fassung *n* liefern. Das Ergebnis könnte etwa so aussehen wie in Listing 7.1 auf der nächsten Seite.

Funktionsaufrufe sind als Schritte möglich. Ein »Weiternavigieren« nach Generatoren ist jedoch nicht möglich.

7.4.9 Alignments

Zur Navigation zwischen alignierten Textabschnitten (zu Alignments im Datenmodell vgl. Abschnitt 3.4.4 auf Seite 42) ist zu beachten, dass alle Bestandteile eines Alignments (mit Ausnahme der Spans) Attribute tragen können. Andererseits wird eine häufige Anwendung die einfache Navigation entlang gewisser Rollen sein.

Die Alignmentbestandteile (mit Ausnahme der Spans) werden deshalb als Elemente betrachtet, sodass die für Elemente zur Verfügung stehenden

Listing 7.1: Aggregierte Inhalte in einem Anfrageergebnis. In diesem Beispiel sind Elemente des Namensraums dq Teil der vom Query-Prozessor generierten Syntax, während Elemente des Namensraums ddd aus den Korpusdaten kopiert worden sind.

```
<dq:result
  query='document[@typ="H"]/text[@fassung="n"]/count(//word[@lemma="renn"])'>
  <ddd:document id="d1" typ="H">
    <ddd:text id="d1t1" fassung="n">
      <dq:aggregate
        function="count"
        argument="descendant-or-self::word[@lemma='renn']">
        42
      </dq:aggregate>
    </ddd:text>
  </ddd:document>
  <ddd:document id="d2" typ="H">
    <!-- ... -->
  </ddd:document>
</dq:result>
```

Achsen für die Navigation innerhalb von Alignments verwendet werden können. Für die Navigation von Spans zu den zugehörigen Alignmentselementen und umgekehrt werden parametrisierbare Achsen zur Verfügung gestellt (Abschnitt 7.4.4 ab Seite 90).

Zur Semantik der Alignmentnavigation und Greediness siehe auch Abschnitt 3.4.4 ab Seite 42.

7.4.10 Boolesche Ausdrücke und Prädikate

$$\langle \text{PredicateList} \rangle ::= \rightarrow \overbrace{\langle \text{Predicate} \rangle} \rightarrow$$

$$\langle \text{Predicate} \rangle ::= \rightarrow [- \langle \text{Expression} \rangle -] \rightarrow$$

Prädikate bilden einen weiteren, den Knotentests nachgelagerten Filter für Knoten. Dabei wird der Ausdruck $\langle \text{Expr} \rangle$ im Kontext jedes Elements v_i der zu filternden Knotensequenz V ausgewertet. Das Ergebnis jeder dieser Auswertungen wird als boolescher Wert $b(v_i)$ interpretiert. Ist $b(v_i)$ falsch, so wird der zugehörige Knoten v_i verworfen, andernfalls in die Ergebnissequenz des Prädikats übernommen.

Wie in anderen linguistischen Anfragesprachen wie etwa CQP soll es möglich sein, Teilanfragen mit booleschen Ausdrücken zu realisieren. Boolesche Ausdrücke sind dabei innerhalb von Prädikaten möglich, für die oberste Anfrageebene steht mit inneren regulären Pfadausdrücken ein ähnliches Konstrukt zur Verfügung.

Boolesche Ausdrücke basieren (ebenso wie Vergleichsausdrücke) darauf, dass Pfadausdrücke in einem *Wertzusammenhang* ausgewertet werden können.

Dabei gilt:

- Wenn ein Pfadausdruck ausschließlich aus einer $\langle \text{PrimaryExpression} \rangle$ besteht, dann ist der Wert des Pfadausdrucks der Wert eben dieser $\langle \text{PrimaryExpression} \rangle$. Der boolesche Wert einer $\langle \text{PrimaryExpression} \rangle$ E ist wahr gdw. der Wert von E nicht false ist.
- Andernfalls:
 1. Der *boolesche Wert* eines Pfadausdrucks ist genau dann wahr, wenn der Pfadausdruck auf (mindestens) einen Pfad des Graphen zutrifft.
 2. Ein *Textwert* eines Pfadausdrucks ist genau dann wahr, wenn der Pfadausdruck auf genau einen Pfad des Korpusgraphen zutrifft und der Zielknoten des Pfades einen Textwert hat.
 - Ist der Zielknoten ein Attributknoten, so ist der Textwert der Wert des Attributs.
 - Ist der Zielknoten ein Span, so ist der Textwert der Wert des Spans.
 - Ist der Zielknoten ein Annotationselement, so ist der Textwert der Textwert des mit dem Element assoziierten Spans.

Ein Prädikat p wählt einen Knoten v dann aus, wenn die Auswertung des Ausdrucks, der das Argument zu p bildet, im Kontext von v eine nichtleere Menge zum Ergebnis hat. Im Ausdruck

`word[@pos='ADJ' | @pos='ADV']`

wird durch das Prädikat also nur diejenige Menge von word-Elementen gewählt, die ein Attribut *pos* mit dem Wert *ADJ* oder ein Attribut *pos* mit dem Wert *ADV* haben.

7.4.11 Ausdrücke, Vergleiche und Arithmetik

Da in *DDDquery* keine Konstruktion von Knotensequenzen vorgesehen ist, werden dies betreffende Teile der XPath-Ausdrücke nicht unterstützt. So fallen etwa Quantoren und Mengenarithmetik (Vereinigung, Schnitt) weg.

Die Ausdrücke kommen hier im Kern zur Formulierung von Bedingungen in Prädikaten zum Einsatz. Ihre Syntax ist im Anhang in Abschnitt A.2 dargestellt.

Auswertung

Pfadausdrücke, die in einem Vergleichs- oder arithmetischen Ausdruck vorkommen, werden zunächst wie in XPath im *Wertkontext* ausgewertet, d.h. wenn diese Pfadausdrücke auf *genau einen* Knoten des Korpusgraphen zutreffen, dann wird der Textwert dieses Knotens (bei Spans der Textinhalt, bei Annotationselementen der Textinhalt des assoziierten Spans, bei Attributen der Wert) verwendet.

7.4.12 Anfragen

Dieser Abschnitt definiert jene Sprachkonstrukte, die tatsächlich als Anfrage angegeben werden können.

$\langle \text{Query} \rangle ::= \text{---} \langle \text{InnerPathExpr} \rangle \text{---}$

Dabei sind nicht nur Pfade, sondern $\langle \text{InnerPathExpr} \rangle$ s (Definition im folgenden Abschnitt) zulässig, um komplexere Konstrukte mithilfe von Joinvariablen zu ermöglichen:

```
//sentence#$s//word/"Eiris"$w & $s/element-span::span()/prefix:$w
```

wählt beispielsweise alle Sätze, die mit dem word »Eiris« beginnen,

```
word#/"idisi" | verse#/suffix::"un"
```

wählt alle Wörter »idisi« und alle Verse, die auf »un« enden.

7.4.13 Reguläre Pfadausdrücke

Zu den Anforderungen an eine linguistische Anfragesprache gehört auch die Fähigkeit, reguläre Ausdrücke nicht nur auf Text, sondern auch auf komplexere Objekte (also Knoten im Kontextgraph) anzuwenden (etwa Faulstich et al., 2005; Cassidy, 2002). In diesem Sinn ist es also sinnvoll, über Teilpfade reguläre Ausdrücke zu definieren. Dies beinhaltet den Anspruch auf die Operationen

7 Anfragesprache

- Konkatination,
- Alternative und
- Kleenesche Hülle.

Konkatination ist im bisherigen Sprachentwurf bereits der Pfadsemantik immanent. Alternativen lassen sich einbauen und als Oder-Ausdruck aufzählen:

$$\pi_1 (\pi_{2a} \vee \pi_{2b}) \pi_3 \iff \pi_1 \pi_{2a} \pi_3 \vee \pi_1 \pi_{2b} \pi_3$$

Bleibt als Problem die kleenesche Hülle. Da die Anzahl der Wiederholungen des Operanden dabei nicht feststeht, ist eine Aufzählung aller einfachen Expansionen nicht möglich. Allerdings bieten moderne Datenbankmanagementsysteme eine Fixpunktrekursion als Sprachmittel, mit der sich das Problem lösen lässt – allerdings sind diese rekursiven Anfragen nicht sehr effizient.

$$\langle \text{PathRegExpr} \rangle ::= \rightarrow \{ - \langle \text{InnerPathExpr} \rangle \underbrace{\quad}_{\text{ }^*} \}$$

$$\langle \text{InnerPathExpr} \rangle ::= \rightarrow \langle \text{InnerPathFactor} \rangle \underbrace{\quad}_{\text{ }^*}$$

$$\langle \text{InnerPathFactor} \rangle ::= \rightarrow \langle \text{PathTail} \rangle \underbrace{\quad}_{\text{ }^*}$$

7.4.14 Leerraum (Whitespace)

Leerraum (Leerzeichen, Tabulatoren, Zeilenumbrüche, Kommentare) kann wie in XPath¹ *zwischen* Terminalen auftreten und wird bei der Analyse der Sprache ignoriert. Leerraum kann ggf. erforderlich sein, um aufeinanderfolgende Tokens voneinander zu unterscheiden.

Kommentare werden ebenfalls wie Leerraum behandelt und in (: ... :) eingeschlossen.

$$\langle \text{Comment} \rangle ::= \rightarrow (: - \langle \text{Comment Content} \rangle - :)$$

¹ <http://www.w3.org/TR/2005/WD-xpath20-20050404/#IgnorableWhitespace>

7.5 Erweiterbarkeit

Der Entwurf der Sprache erfolgte – auch in Anbetracht der noch frühen Phase, in der das DDD-Projekt sich befindet – vor dem Hintergrund der Erweiterbarkeit. So lassen sich etwa Erweiterungen des Datenmodells oder zusätzliche Knotenrelationen durch neue Achsen realisieren oder weitere statistische Auswertungen mit neuen Generator-Funktionen ergänzen.

Die Orientierung der Anfragesprache an Knoten und Pfaden lässt ein graphisches Frontend ähnlich etwa TIGERin (Voormann, 2002) zu (vgl. Abschnitt 9).

7 *Anfragesprache*

8 Implementierung

Für die Implementierung der Sprache war insbesondere zu beachten, dass die effiziente Speicherung und Anfrage der Sprach- und Graphdaten in einem Relationalen Datenbankmanagementsystem Gegenstand künftiger Forschung innerhalb des DDD-Projekts ist und die im Produktionseinsatz zu verwendende Methode der Speicherung noch nicht feststeht. Unter diesem Gesichtspunkt wird die Sprache deshalb so implementiert, dass die Komponente zur konkreten Generierung von SQL-Anfragen austauschbar bleibt.

Die Übersetzung von *DDDquery* nach SQL läuft dementsprechend in folgendem dreistufigen Prozess ab:

1. Ein Parser baut zunächst aufgrund einer komplexen Grammatik (vgl. Abschnitt A) einen abstrakten Syntaxbaum T .

Dabei wird die vollständige, in Kapitel 7 ab Seite 79 vorgestellte Grammatik inklusive der möglichen Kurzschreibweisen wie etwa $\langle AbbrAxisStep \rangle$ oder $\langle RelSymb \rangle$ verwendet.

2. Ein Postprozessor vereinfacht diesen Syntaxbaum dann so zu einem Baum T' , dass möglichst von der konkreten Syntax abstrahiert und nur noch die Semantik repräsentiert wird. Insbesondere werden hier die Kurzschreibweisen in ihr Langform-Äquivalent gewandelt, so werden in T' beispielsweise die Schreibweisen *verse/word* und *element(verse)/child::element(word)* auf gleiche Weise repräsentiert. Im Ergebnissyntaxbaum sind dann die bloßen Abkürzungszwecken dienenden Elemente nicht mehr vorhanden und müssen von nachfolgenden Stufen damit auch nicht mehr behandelt werden.

Darüberhinaus übernimmt der Postprozessor die statische Typprüfung und generiert ggf. zusätzliche Schritte zur Typkonvertierung (vgl. Abschnitt 7.4.3.3).

3. Dieser vereinfachte Syntaxbaum bildet dann die Grundlage für die weitere Verarbeitung der Suchanfrage, etwa zur Übersetzung nach SQL.

8 Implementierung

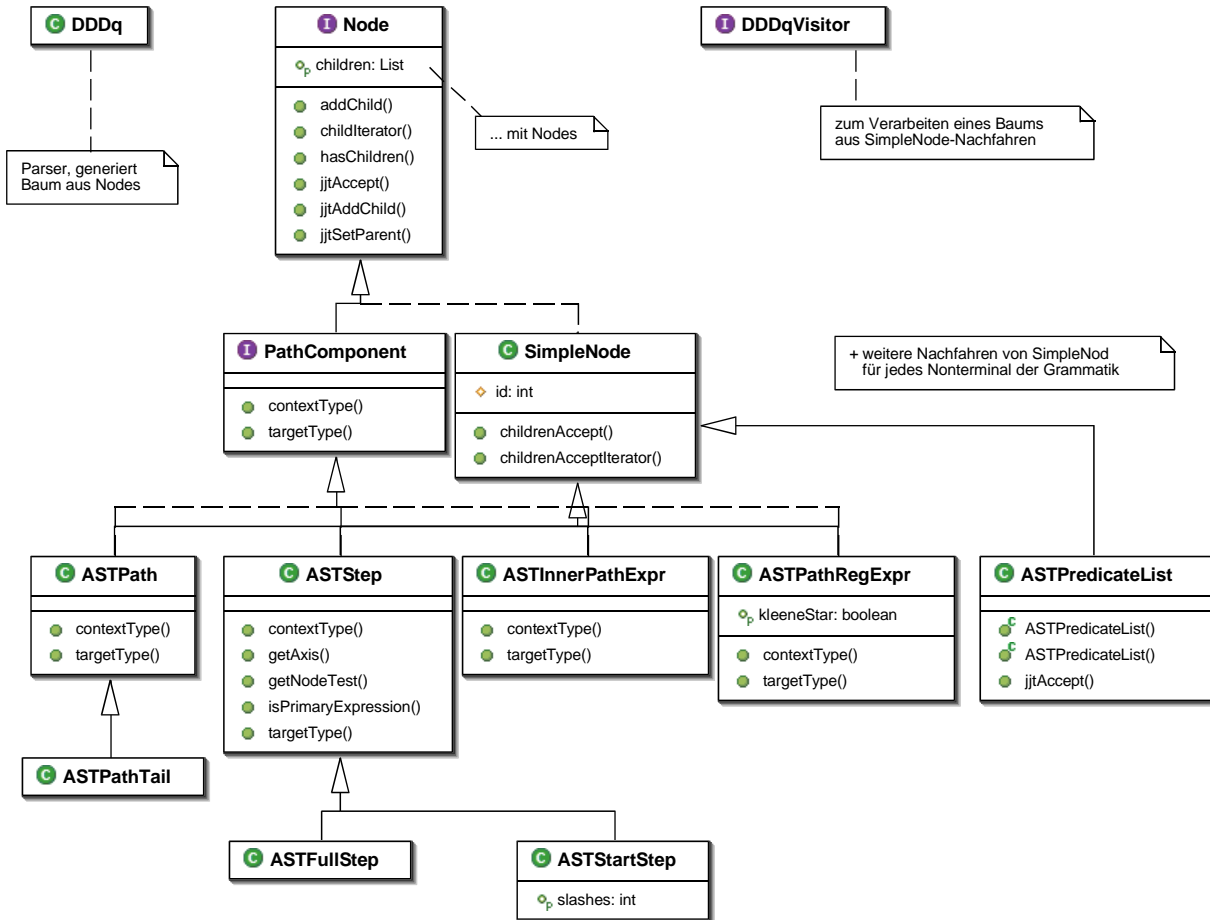


Abbildung 8.1: Ausschnitt aus dem Modell des abstrakten Syntaxbaums

Auf diese Weise kann die Komponente aus Punkt 3 ausgetauscht werden, um z. B. diverse Speichermethoden zu evaluieren, ohne dass der Rest der Sprachverarbeitung geändert werden muss.

8.1 Parser und abstrakter Syntaxbaum

Der Parser für die Sprache wurde mithilfe des Compilergenerators JavaCC¹ und dessen Erweiterung für Syntaxbäume JJTree realisiert.

¹ <https://javacc.dev.java.net/>

Dabei existiert für jedes Nonterminal der Grammatik eine Klasse als Nachfahre von *SimpleNode*. Die Instanzen all dieser Klassen – die Knoten des Baumes – können jeweils eine sortierte Liste von Kindern besitzen, die die Ableitung des Nonterminals repräsentiert.

Die Klassen besitzen ggf. zusätzliche Attribute und Methoden, die gewisse Eigenschaften des Nichtterminals repräsentieren. So kann zum Beispiel für eine Achse deren Art ermittelt werden. Um die Verarbeitung zu erleichtern, wurden für Nichtterminale mit ähnlichen Eigenschaften Interfaces eingefügt (z. B. *PathComponent* für alle Komponenten eines Pfades, die jeweils über einen Kontext- und einen Zieltyp verfügen) Zum Teil werden auch einzelne Knoten nicht direkt von *SimpleNode* abgeleitet, sondern es wurde ein zusätzliches Zwischenobjekt eingeführt (etwa für *Steps*).

In Abbildung 8.1 auf der vorherigen Seite sind zur Illustration einige der Objekte des abstrakten Syntaxbaums dargestellt. Aus Platzgründen wurde auch nur ein Teil der jeweils vorhandenen Methoden in das Diagramm aufgenommen.

Die Weiterverarbeitung des abstrakten Syntaxbaums erfolgt mithilfe des Visitor-Patterns (Appel, 2002; Gamma et al., 2001). Auf diese Weise können unterschiedliche Datenbank-Backends implementiert werden, ohne den Code zum Parsen der Anfragen und zum Aufbau und Vereinfachen des Syntaxbaums verändern zu müssen.

8.2 Normalisierung des abstrakten Syntaxbaums

Im zweiten Verarbeitungsschritt wird der vom Parser erzeugte Syntaxbaum normalisiert. Im Wesentlichen finden hier drei Anpassungen statt:

1. Die Kurzschreibweisen werden aufgelöst. Danach enthält der abstrakte Syntaxbaum jene Knotentypen nicht mehr, die nur zu Zwecken der kürzeren oder bequemer Schreibeise in die Syntax aufgenommen worden sind.

Die betroffenen Syntaxelemente sind in Kapitel 7 entsprechend gekennzeichnet.

2. Die Markierungssyntax wird aufgelöst. Nach der Prozessierung ist jeder auszugebende Schritt mit einer entsprechenden Markierung versehen.

In einem Ausdruck wie page#+/line/verse enthalten also (dank des #+) *alle* Schritte eine Markierung.

3. Konvertierungsschritte werden eingeführt. Dieses Verfahren ist genauer in Abschnitt 7.4.3.3 auf Seite 89 beschrieben und wird durch die Umstrukturierung der betroffenen Teil-Syntaxbäume realisiert.

8.3 Umsetzung von Anfragen und Ausgaben in einem RDBMS

Die vorgeschlagene Umsetzung in einem RDBMS basiert im Kern auf der Basis eines Intervallansatzes mit einer Preorder-Postorder-Numerierung der Elemente, wie sie bereits in Vitt (2004) umgesetzt wurde. Relationen auf der Basis einer horizontalen Navigation werden jedoch grundsätzlich auf der Basis von Spans implementiert.

Die Implementierung, die im Folgenden näher beschrieben wird, setzt einen Kernumfang der Sprache um und soll insbesondere die Rekonstruktion von Daten zu dem in Kapitel 4 entwickelten Ergebnisschema, und hier insbesondere die Selektion von Kontextinformationen mithilfe von Markern (vgl. Abschnitt 7.4.6 ab Seite 99) illustrieren.

Die Rekonstruktion der Graphdaten (in einer serialisierten XML-Form, vgl. Kapitel 4 ab Seite 45 und Vitt, 2004) beruht auf der Selektion von Tupeln, die die jeweils auszugebenden Knoten markieren; die Strukturinformationen werden anhand der Pre- und Postorderkennzahlen rekonstruiert, wie dies in Vitt (2004) gezeigt wurde.

Zusätzlich ist noch das Einfügen der Markierungsinformationen (Abschnitt 7.4.6 ab Seite 99) bzw. eine entsprechende Beschränkung der Ergebnistupelliste nötig. Dies kann beispielsweise durch zusätzliche Bedingungen in der **SELECT**-Klausel der SQL-Anfrage geschehen.

8.3.1 Datenbankschema

Das Datenbankschema (vgl. Abbildung 8.2 auf der nächsten Seite) orientiert sich an dem von Vitt (2004) sowie Faulstich et al. (2005) vorgestellten Schema und berücksichtigt darüberhinaus die Besonderheiten des Datenmodells aus Kapitel 2.

8.3.1.1 Graphstruktur

Für die Graphstruktur wird eine Elementtabelle vorgesehen, die zu jedem Knoten Pre- und Postorderrang sowie Knoten-ID und Elementtyp vorsieht. Weiterhin wird für die nichttransitiven hierarchischen Anfragen

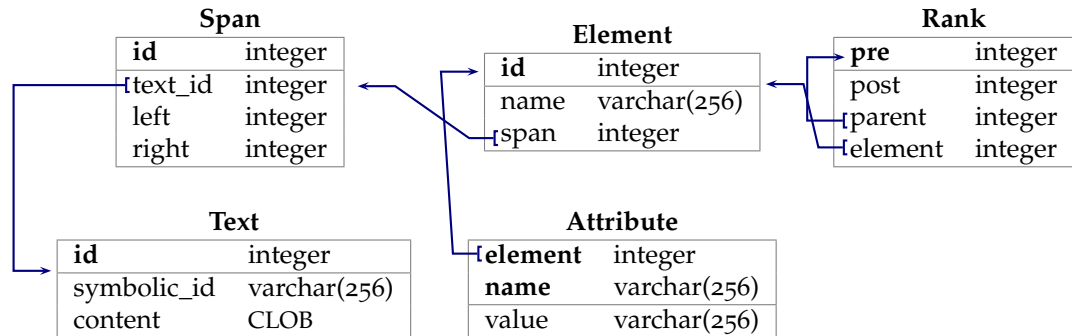


Abbildung 8.2: Grundlegendes Datenbankschema

(parent, child, sibling, ...) eine Referenz auf den Elternknoten eingefügt. Darüberhinaus gibt es Felder, um einem Annotationselement einen Span zuzuordnen.

8.3.1.2 Spans

Spans treten in der Anfragesprache immer *gebunden* auf: Entweder durch die Assoziation mit einem Annotationselement oder einem Align-Objekt, durch die Volltextsuche oder durch Relation mit einem anderen Span.

Lediglich bei letzterer Relation können theoretisch »beliebige« Spans erzeugt werden. Es gibt jedoch auch hier eine maximale äußere Grenze, die sich durch den Span des gesamten aktuellen Textes bestimmt.

Bei der Behandlung von Spans werden, auch ob der Komplexität der Suchergebnisse, deshalb nur solche Spans betrachtet, die in der Datenbank materialisiert sind, oder die unmittelbar aus solchen Spans berechnet werden können.

Angenommen, diese Beschränkung existiere nicht. Im folgenden Beispielausdruck:

$$\underbrace{\text{word}[\text{@pos}=\text{'NN'}]/\text{element-span}::\text{span}()}_{{\pi \leadsto S}} / \underbrace{\text{contained-in}::\text{span}()}_{{\pi' \leadsto S'}} \#$$

expandiere der erste Teilpfad π zu einer Menge von Spans S . Sei $s_0 = (t_0, l_0, r_0)$ ein beliebiges Element von S . Sei ferner $|t_0|$ die Länge des durch t_0 bezeichneten Texts. Dann sind alle Spans s' mit $s' = (t_0, l', r')$ und $0 \leq l' \leq l_0$ sowie $r_0 \leq r' \leq |t_0|$ Element des Gesamtausdrucks und (wegen des #) als Ergebnisspan auszugeben. Das kann bei entsprechend langem

8 Implementierung

Text eine sehr große Zahl von Spans, die in weiten Teilen übereinstimmen, ergeben, und es ist sicher nicht ein erwünschtes Ergebnis.

Die mit einem Annotationselement verbundenen Spans werden in einer eigenen Span-Tabelle in der Datenbank repräsentiert. Für Spans, die dynamisch durch Volltextsuche erzeugt werden, existieren Tabellenfunktionen für das RDBMS in Oracles PL/SQL.

8.3.1.3 Texte

Die Texte werden in einer eigenen Tabelle gespeichert, als Primärschlüssel dient eine Text-ID, die auch von den Spans in den anderen Tabellen entsprechend referenziert wird.

8.3.1.4 Headerdaten etc.

Die Headerangaben werden wie Texte, Spans und Annotationselemente behandelt.

8.3.1.5 Layer

Die jeweilige Annotationsebene kann jeweils entweder einem Element zugeordnet werden (und in deren Tabelle gespeichert werden), oder es findet gar keine explizite Speicherung anders als über ein Layer-Element statt. Da den vergleichsweise wenigen zu erwartenden unmittelbar benannten Layer-Anfragen (mit der layer-Achse) das Anbringen eines zusätzlichen Layer-Feldes in der Elementtabelle nicht angemessen erscheint, wird darauf verzichtet und Layer werden einfach als gewöhnliche Elemente in der Elementtabelle betrachtet. Die Anfragen mit der layer-Achse können dann in entsprechende Strukturanfragen gewandelt werden.

8.3.1.6 Alignments

Auch Alignments werden wie Elemente behandelt. Die Implementierung insbesondere mit Berücksichtigung der Alignments von Subspans kann mithilfe von Tabellenfunktionen erfolgen.

8.3.2 Rekonstruktion des Anfrageergebnisses in XML/gXDF

8.3.2.1 Strukturdaten

Um aus tupelförmigen Ergebnisse einer SQL-Anfrage wieder eine XML- bzw. Baumstruktur, wie sie das Ergebnisschema fordert, zu erzeugen, wird auf eine ähnliche Weise verfahren, wie dies schon in Vitt (2004) geschehen ist: Die Tupel werden (schon vom RDBMS) nach ihrem Preorder-Rang sortiert, der auch der Reihenfolge der Elemente im XML-Ergebnis entspricht, und mithilfe der Pre- und Postorderränge und eines Stacks der zur Zeit offenen Elemente wird entschieden, wie viele der zur Zeit offenen Elemente vor dem Einfügen des aktuellen Elements erst geschlossen werden müssen (indem ihr End-Tag in den Ausgabestrom eingefügt wird).

Das Format, in dem die Tupel aus der Datenbank übernommen werden, wird in Abschnitt 8.3.3.1 auf der nächsten Seite beschrieben.

Mithilfe der dort vorhandenen Marker-ID und der Informationen über den zugehörigen Pfadschritt der Anfrage werden dabei Markierungsattribute in die Ausgabe übernommen.

8.3.2.2 Textabschnitte

Für die Rekonstruktion der Textinhalte wird eine eigene einfache SQL-Anfrage gestellt. Bei der im vorhergehenden Abschnitt genannten Umsetzung der Anfragen ist durchaus davon auszugehen, dass zahlreiche einander überlappende Spans zurückgeliefert werden.

Man bedenke etwa eine Anfrage auf einer Annotationsebene zur physischen Struktur, in der alle Nachfahren eines page-Knotens mitsamt der assoziierten Spans zurückgeliefert werden sollen. Zu den Nachfahren können beispielsweise zahlreiche lines gehören, von denen jede wiederum mehrere words besitzt; und all diese Objekte sind mit Subspans des ursprünglichen zum page-Elements gehörenden Spans assoziiert.

Die hohe Redundanz der zu extrahierenden und übetragenden Spans soll hier vermieden werden, zumal ohnehin ein Format mit Standoff-Annotation erzeugt wird (vgl. Abschnitt 4.3.1). Deshalb wird bei der Rekonstruktion der Annotationsdaten eine Liste aller selektierten Texte mit ihren jeweiligen *maximal auszugebenden Intervallen* (vgl. Definition 9 auf Seite 58) geführt, die die Grundlage für die zweite Anfrage bildet.

8.3.3 Umsetzung der SQL-Anfragen

Die SQL-Anfragen sollen möglichst so erzeugt werden, dass für jede *DDDquery*-Query nur *eine* SQL-Anfrage für die Knotenstruktur (und eine weitere für die Textinhalte) generiert wird.

Damit sollen die Möglichkeiten der Optimierer aktueller RDBMS möglichst zum Tragen kommen können.

8.3.3.1 Rückgabeformat

Die aus den SQL-Anfragen gewonnenen Daten werden dann an einen Algorithmus zur Serialisierung in die XML-Form übergeben. Das Abfrageergebnis hat dabei den in Abbildung 8.3 dargestellten Aufbau.

Tabelle	Feld	Beschreibung
rank	pre	<i>Preorder-Rank</i>
	post	<i>Postorder-Rank</i>
element	node_id	<i>ID des selektierten Elements</i>
	element	<i>Name / Typ des selektierten Elements</i>
span	text_id	<i>Text</i>
	left	<i>Beginn</i>
	right	<i>Ende</i>
attribut	attribute	
	value	
(Query)	marker	<i>Marker-ID für Ausgabe, bez. Schritt</i>

Abbildung 8.3: Ergebnisse der SQL-Anfragen

8.3.3.2 Schritte und Pfade

Das Grundprinzip der Anfragegenerierung sei hier an einer Umsetzung des Anfragefragments `//element(page)#(a) / child::element(line)#(b)` vorgestellt, die in Listing 8.1 auf der nächsten Seite dargestellt ist.

- Für jeden Schritt in der ursprünglichen Anfrage wird ein *Aliassatz* angelegt. Ein Aliassatz ist eine Menge von Tabellenaliasen, in der zu jeder Tabelle aus dem Datenbankschema maximal ein Alias vorhanden ist.

Im Beispiel sind dies die Aliassätze $\mathcal{A}_1 = \{\text{rank1}, \text{element1}\}$ und $\mathcal{A}_2 = \{\text{rank2}, \text{element2}\}$.

8.3 Umsetzung von Anfragen und Ausgaben in einem RDBMS

Listing 8.1: Umsetzung der Anfrage page/line

```
-- DDDquery expression 'page#(a)/line#(b)'  
SELECT DISTINCT  
    rank0.pre, rank0.post, element0.id, element0.name,  
    span0.text_id, span0.left, span0.right,  
    attribute0.name, attribute0.value,  
    CASE  
        WHEN rank0.pre = rank1.pre  
        THEN 1  
        WHEN rank0.pre = rank2.pre  
        THEN 2  
    END AS marker_no  
FROM rank rank0, element element0, span span0, attribute attribute0,  
    rank rank1, element element1,  
    rank rank2, element element2  
WHERE rank2.parent = rank1.pre      /* (1) A_1 parent A_2 */  
    AND element1.name = 'page'      /* (2) Knotentests */  
    AND element2.name = 'line'  
    AND element1.id = rank1.element /* Join zwischen (1) u. (2) */  
    AND element2.id = rank2.element  
    AND (  
        (  
            rank0.pre = rank1.pre  
            AND element0.id = element1.id  
        )  
        OR  
        (  
            rank0.pre = rank2.pre  
            AND element0.id = element2.id  
        )  
    )  
    AND span0.id = element0.span /* vgl. SELECT-Zeile */  
    AND element0.id = attribute0.element  
ORDER BY rank0.pre ;
```

- entsprechend der Bedingungen des *Knotentests* werden entsprechende Bedingungen an die jeweiligen Aliase hinzugefügt (hier die beiden Bedingungen auf `elementn.name`).
- Aus der *Achse*, die zwei Schritte miteinander verbindet, werden entsprechende Joinbedingungen zwischen den beiden zugehörigen Aliassätzen generiert. Wie dies für die hierarchischen Anfragen mit Pre- und Postordercodierung geht, ist etwa in bei Grust et al. (2004) oder Vitt (2004) beschrieben.

In diesem Fall ist das der Join `rank2.parent = rank1.pre`.

- Für je zwei Tabellenalias a, a' aus *demselben* Aliassatz, für die noch kein Join existiert, werden Joins über die entsprechenden Fremdschlüsselbeziehungen generiert.

8.3.3.3 Umsetzung der Markierungen

Für die Umsetzung von Markierungen sollen zwei Dinge erreicht werden:

- Alle auszugebenden Tupel sind in denselben Spalten, dem in Abschnitt 8.3.3.1 auf Seite 116 vorgestellten Schema folgend, auszugeben und entsprechend dem Preorder-Rang sortiert.
- In einem Feld dieses Rückgabeformats ist für jedes Tupel vermerkt, durch welchen Schritt (bzw. von welchem Aliassatz) es selektiert wurde. Anhand dieser Information kann dann das entsprechende Marker-Attribut in der XML-Form hinzugefügt werden.

Im trivialen Fall, dass nur ein Schritt zur Ausgabe markiert worden ist, wird einfach der zu diesem Schritt gehörige Aliassatz in die **SELECT**-Klausel übernommen. Gegebenenfalls sind die notwendigen Aliase und fehlenden Joins über die Fremdschlüsselbeziehungen zu ergänzen.

Sind mehrere Schritte markiert, wie dies im Beispiel der Fall ist, so wird ein zusätzlicher Ausgabe-Aliassatz \mathcal{A}_0 generiert, der dann in die **SELECT**-Klausel übernommen wird. In einer Alternative wird dieser Aliassatz mit allen potentiell auszugebenden Aliassätzen verbunden, und schließlich werden noch die fehlenden Joins entlang der Fremdschlüsselbeziehungen ergänzt.

Das Marker-Feld in der **SELECT**-Klausel wird mit einer **CASE**-Anweisung analog zu dieser Alternative gesetzt, wie dies im Beispiel der Fall ist.

8.3.3.4 Suche / Span-Tests

Für die Span-Tests wurden eine Reihe von Tabellenfunktionen (in PL/SQL) implementiert, die die zu einem bestimmten Muster passenden Spans aus entsprechenden Texten extrahieren. So liefert beispielsweise die SQL-Anfrage

```
SELECT t.id, s.*
FROM   text t, TABLE(spans_re(t.content, 'E[^_]*_.azun')) s
```

mithilfe der Tabellenfunktion `span_re` alle Spans aus der Texttabelle, auf die der reguläre Ausdruck aus dem zweiten Argument von `span_re` passt.

Um die Text- und damit Tupelzahl zu begrenzen, auf die die Funktion angewandt wird, ist ggf. eine Ergänzung um eine zusätzliche **WHERE**-Bedingung sinnvoll:

```
SELECT t.id, s.*
FROM   text t, TABLE(spans_re(t.content, 'E[^_]*_.azun')) s
WHERE   REGEXP_LIKE(t.content, 'E[^_]*_.azun');
```

8.3.3.5 Variablen

Da laut Sprachdefinition alle Schritte, die an dieselbe Variable gebunden werden, auf denselben Knoten jedes Ergebnisses zutreffen müssen (vgl. Abschnitt 7.4.7 auf Seite 101), wird einfach für alle Schritte, die an dieselbe Variable gebunden sind, derselbe Aliassatz verwendet.

8.3.3.6 Einfache Prädikate

Prädikate, die lediglich aus einem einfachen Pfad bestehen (und damit de facto eine Existenzbedingung stellen), können einfach wie im Abschnitt über Pfade und Schritte beschrieben umgesetzt werden. Der Aliassatz, auf den sich die durch die Achse des ersten Schritts des Pfadprädikats induzierten Joins beziehen, ist der Aliassatz des Kontextschritts des Prädikats (also des Schritts, von dem das Prädikat ein Teil ist). Für komplexe Prädikate sind ggf. Subqueries zu generieren.

8.4 Umsetzung mit LoToS

LoToS (Faulstich und Leser, 2005) ist ein System zur Umsetzung von in Prädikatenlogik formulierten Anfragen nach SQL. Zudem wurde mit

8 Implementierung

LoToS / DDDSearch eine linguistische prädikatenlogische Anfragesprache auf der Basis von LoToS entwickelt.

Grundsätzlich ist eine Benutzung von LoToS für *DDDquery* wünschenswert, da das System nicht nur die Generierung von SQL-Anfragen, sondern auch einige Optimierungen wie etwa zur Vermeidung von Selbst-joins bereits enthält. In Bezug auf die Anforderungen von *DDDquery* gibt es jedoch einige Hürden, die zu überwinden sind:

- LoToS arbeitet tupelbasiert, es liefert *ein* Tupel pro möglichem Ergebnis, während *DDDquery* (im Rahmen der Marker) an Zwischenergebnissen interessiert ist.
- Es ist möglich, Zwischenergebnisse auszugeben, indem man sie als freie Variablen in die Anfrage mit aufnimmt. Dies demonstriert die Anfrage:

$$\begin{aligned} Q_7(VP, V, NP, PP) \equiv & \text{element}('VP', VP) \wedge \text{element}('V', V) \wedge \\ & \text{element}('NP', NP) \wedge \text{element}('PP', PP) \wedge \\ & \text{ancestor}(VP, V) \wedge \text{ancestor}(VP, NP) \wedge \text{ancestor}(VP, PP) \wedge \\ & \text{element_span}(VP, VPS) \wedge \text{element_span}(V, VS) \wedge \\ & \text{element_span}(NP, NPS) \wedge \text{element_span}(PP, PPS) \wedge \\ & \text{prefix}(VS, VPS) \wedge \text{immediately_precedes}(VS, NPS) \wedge \\ & \text{immediately_precedes}(NPS, PPS) \wedge \text{suffix}(PPS, VPS) \end{aligned}$$

Die LoToS-DDD-Search-Implementierung übersetzt dies allerdings nach

```
SELECT
    element1.id, element2.id, element3.id, element4.id
FROM
    rank ancestor1, rank descendant1, rank ancestor2,
    rank descendant2, rank ancestor3, rank descendant3,
    element element1, element element2, element element3,
    element element4
WHERE element1.name="VP"
AND element2.name="V"
AND element3.name="NP"
AND element4.name="PP"
...
```

– die vier Variablen erscheinen also als Spalten in der Ausgabe, während für die XML- bzw. gXDF-Serialisierung (vgl. Vitt, 2004) die Ausgabe in denselben Spalten, jedoch ggf. mit einer Zuordnung

zu Variablen oder sonstigen Markierungen in einer dedizierten Zusatzspalte erforderlich ist.

- Für den Rekonstruktionsalgorithmus sind nicht allein die Tupel-IDs, wie von LoToS / DDD-Search geliefert, sondern auch weitere Informationen wie Pre- und Postorderrank notwendig.

Um also Anfrageergebnisse so zu liefern, dass sie zur Konstruktion des Ergebnisschemas weitergenutzt werden können, sind tiefere Eingriffe in LoToS oder ein umfangreiches Postprocessing nötig, weshalb für diese Arbeit nicht auf LoToS zurückgegriffen wird.

8 Implementierung

9 Zusammenfassung und Ausblick

Mit der vorliegende Diplomarbeit wurde ein Anfragesystem aus Anfragesprache und Format für die Anfrageergebnisse entworfen, das im in Kapitel 1 vorgestellten Projekt *Deutsch Diachron Digital* (DDD) als Ebene zwischen grafischen Benutzungsschnittstellen und einem Datenbank-Backend fungieren kann. Dabei soll die Anfragesprache zumindest von fortgeschrittenen Benutzern auch unmittelbar zur Formulierung komplexer Suchanfragen benutzt werden können.

In Kapitel 2 wurde das dem DDD-Korpus und somit auch dem Anfragesystem zugrundeliegende Datenmodell detailliert vorgestellt und dessen besondere Eigenschaften wie Elemente und Spans zur Realisierung mehrerer Annotationsebenen oder Alignments zur Verbindung unterschiedlicher Texte erläutert.

Kapitel 3 stellte existierende Benutzungsschnittstellen für historische und andere linguistische Korpora vor. Die Erkenntnisse aus diesem Kapitel vor allem über verschiedene Darstellungsformen in GUIs – etwa der KWIC-Darstellung für Suchwörter in ihrem Kontext oder Frequenztabeln mit aggregierten Suchergebnissen – bildeten eine wichtige Basis für die Anforderungsanalyse für das Datenschema der Anfrageergebnisse.

Letzteres wird in Kapitel 4 entworfen und vorgestellt und bildet eine Erweiterung des bereits im Projekt verwendeten internen Austauschformats gXDF. Annotationen und Markierungen von Suchergebnissen werden unterstützt und getrennt von den Texten als sog. *Standoff-Annotation* eingebunden, um so problemlos mehrere Annotationsebenen anzubieten.

Existierende linguistische *Anfragesprachen* werden in Kapitel 5 vorgestellt. Gemeinsam mit dem Datenmodell und den Erkenntnissen aus dem Entwurf des Ergebnisschemas bildet dieses Kapitel eine Grundlage der Anforderungsanalyse für die Anfragesprache in Kapitel 6.

Entsprechend den dort zusammengetragenen Anforderungen wurde dann in Kapitel 7 die Anfragesprache selbst entworfen – eine an XPath orientierte, aber an die vorgenannten Anforderungen wie etwa Treffer- und Kontextmarkierungen und insbesondere das Datenmodell besonders unterstützende Sprache. In diesem Kapitel wird auch die Syntax der

Sprache detailliert beschrieben.

Kapitel 8 erläutert schließlich die Implementierung des Anfragesystems, insbesondere in Hinblick auf die Konstruktion der Anfrageergebnisse im entworfenen Ergebnisschema. Dabei wurden Parser und Backend getrennt. Für letzteres wurde ein intervallbasierter Ansatz wie schon in Vitt (2004) entwickelt, und mit Features für die Realisierung besonderer Fähigkeiten von *DDDquery* wie der Rückgabe ganzer Teilkorpora, die zudem unterschiedlich markiert werden sollen, auf der Basis von Oracle implementiert.

Ausblick

Die effiziente Implementierung von Baum- und Graphanfragen in relationalen Datenbanksystemen ist aktives Forschungsthema (vgl. etwa Trißl und Leser, 2005; Grust et al., 2004) und soll auch Gegenstand der Forschung im DDD-Projekt sein. Hier ist wohl der dringendste Bedarf an einer Weiterentwicklung von *DDDquery*.

Die modulare Implementation auf der Basis einer Trennung von Parser und Datenbankbackend mithilfe des Visitor-Patterns ermöglicht in diesem Zusammenhang die Realisierung verschiedener effizienter Mechanismen zur Datenbankspeicherung und -Anfrage, ohne dass die eigentliche Sprache dazu verändert werden muss.

Darüberhinaus lässt der vorgestellte Entwurf über die Definition neuer Achsen, Knotentests oder Funktionen auch die Erweiterung der Sprache etwa um neue Relationen oder Aggregationen zu, sollte sich im Zuge des Projektes der Bedarf dazu ergeben.

Graphische Variante der Anfragesprache

Für TIGER existiert mit TIGERin (Voormann, 2002) ein Werkzeug, um Suchanfragen grafisch durch Zeichnen eines Graphen zu formulieren, auch ICECUP III kennt eine solche Möglichkeit (vgl. Abschnitt 3.2.1 auf Seite 37). Ein solches Werkzeug bietet Anwendern die Möglichkeit, komplexe Anfragen formulieren zu können, ohne dass dazu die Syntax der Anfragesprache erlernt werden muss; darüberhinaus können grafisch formulierte Anfragen näher an des Benutzers mentalen Modell des Korpus liegen als eine zwangsweise serialisierte und auf Buchstaben und Sonderzeichen reduzierte Anfragesprache.

DDD*query* lässt sich prinzipiell in eine grafische Form umformulieren: Die Knotentests und Prädikate können Knoten des Anfragegraphen bilden, die Achsen die Kanten; die komplexeren durch die Variablen ermöglichten Graphstrukturen ließen sich i. A. direkt zeichnen, Ausgabe-markierungen könnten ein zusätzliches Eingabeelement in den Knoten bilden. Lediglich die regulären Pfadausdrücke, insbesondere in der Variante mit Kleenescher Hülle, bedürften noch einer gesonderten grafischen Darstellungsform.

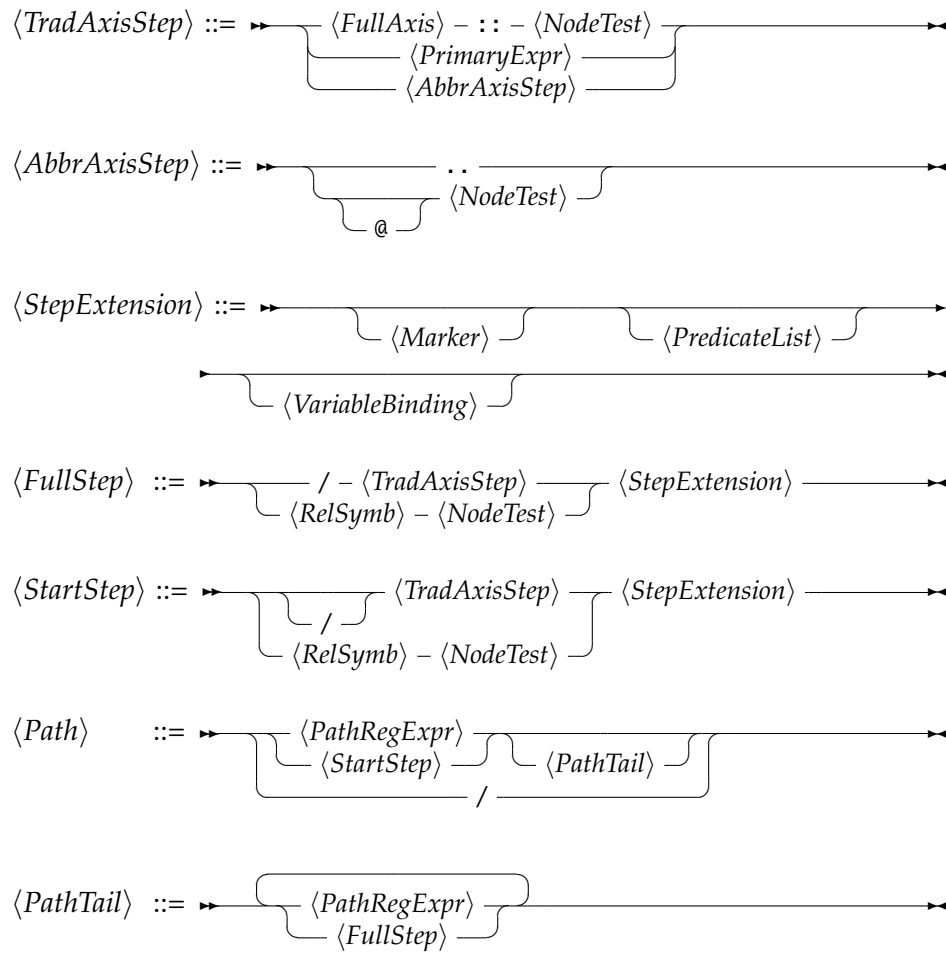
Der Entwurf einer grafischen Anfrageplattform ist jedoch nicht Gegenstand dieser Arbeit.

9 *Zusammenfassung und Ausblick*

Anhang A

Grammatik

A.1 Schritte und Pfade



A.1.1 Reguläre Pfadausdrücke

$$\langle \text{PathRegExpr} \rangle ::= \rightarrow \{ - \langle \text{InnerPathExpr} \rangle \}^* \rightarrow$$

$$\langle \text{InnerPathExpr} \rangle ::= \rightarrow \langle \text{InnerPathFactor} \rangle \{ \langle \text{OR} \rangle - \langle \text{InnerPathFactor} \rangle \}^* \rightarrow$$

$$\langle \text{InnerPathFactor} \rangle ::= \rightarrow \langle \text{PathTail} \rangle \{ \langle \text{AND} \rangle - \langle \text{PathTail} \rangle \}^* \rightarrow$$

A.1.2 Detaillierte Bestandteile eines Schrittes

$$\langle \text{VariableBinding} \rangle ::= \rightarrow \$ - \langle \text{Name} \rangle \rightarrow$$

$$\langle \text{PredicateList} \rangle ::= \rightarrow \{ \langle \text{Predicate} \rangle \}^* \rightarrow$$

$$\langle \text{Predicate} \rangle ::= \rightarrow [- \langle \text{Expression} \rangle -] \rightarrow$$

$$\langle \text{Marker} \rangle ::= \rightarrow \{ \{ \# \}^* \langle \text{MarkerNameSpec} \rangle \}^* \rightarrow$$

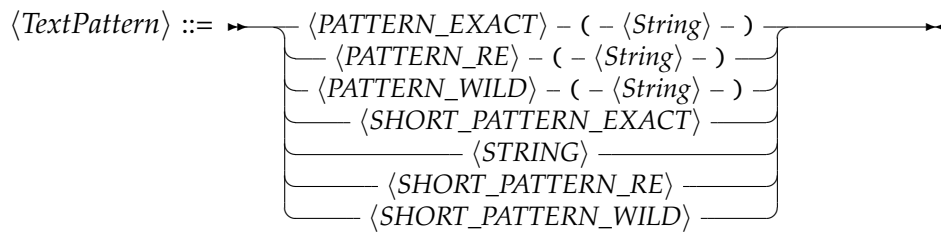
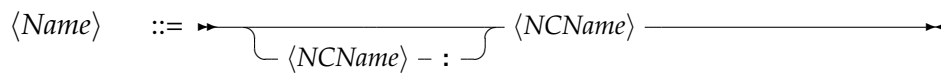
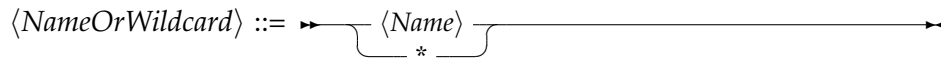
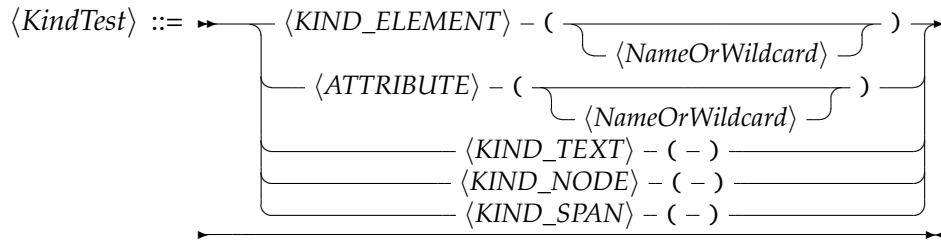
$$\langle \text{MarkerNameSpec} \rangle ::= \rightarrow (\langle \text{Name} \rangle) \rightarrow$$

Für die Grammatik von $\langle \text{FullAxis} \rangle$ und $\langle \text{RelSymb} \rangle$ sei auf die Auflistung der Achsen in Abschnitt 7.4.4.3 ab Seite 92 verwiesen.

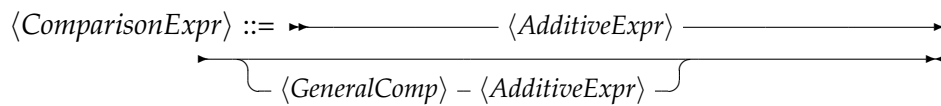
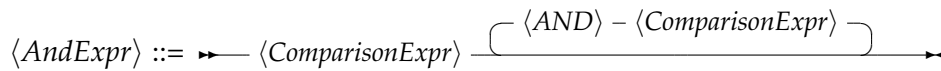
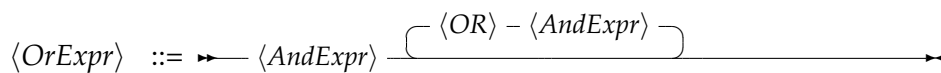
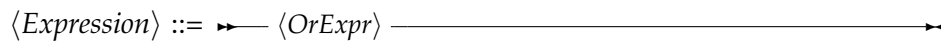
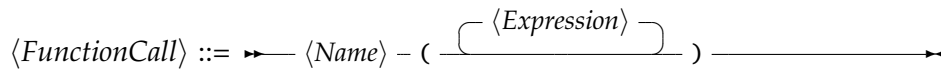
$$\langle \text{Greediness} \rangle ::= \rightarrow \{ \langle \text{GREEDY} \rangle \}^* \rightarrow$$

$$\langle \text{NodeTest} \rangle ::= \rightarrow \{ \langle \text{NameTest} \rangle \}^* \rightarrow$$

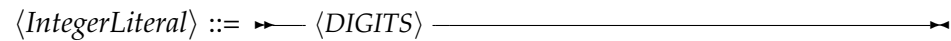
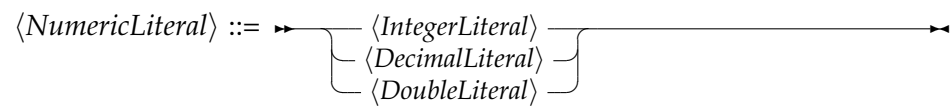
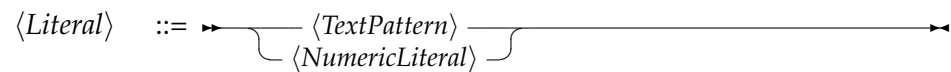
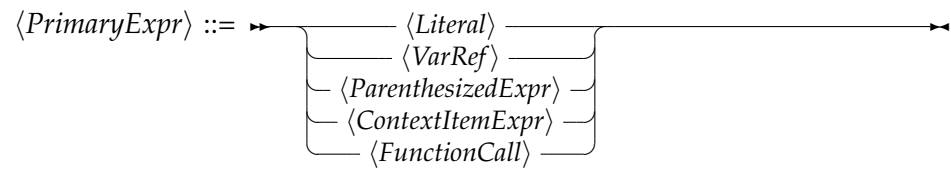
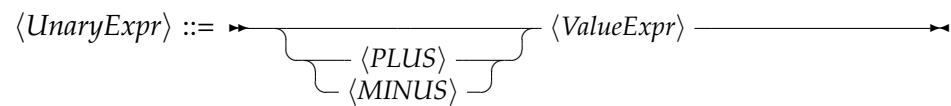
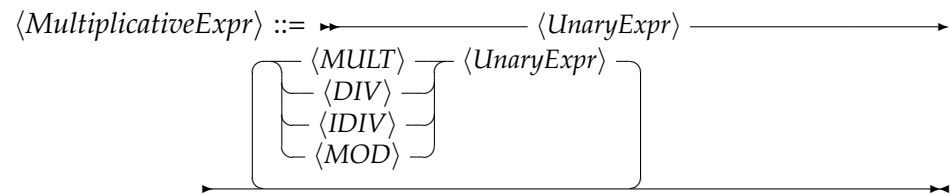
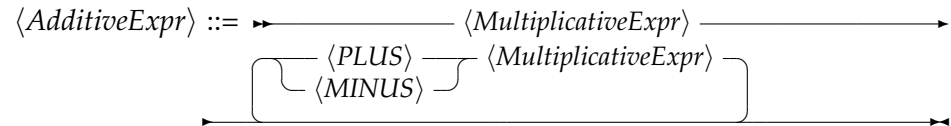
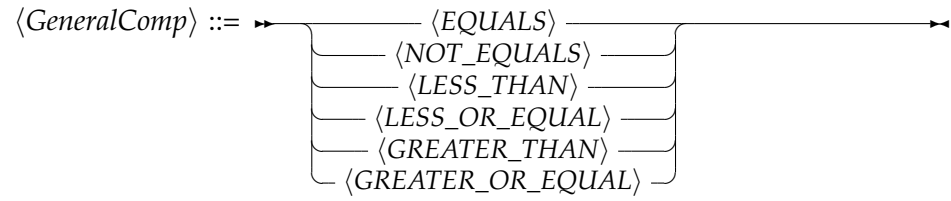
$$\langle \text{NameTest} \rangle ::= \rightarrow \langle \text{NameOrWildcard} \rangle \rightarrow$$



A.2 Ausdrücke



Anhang A Grammatik



$\langle \text{VarRef} \rangle ::= \blacktriangleright \langle \text{VariableBinding} \rangle \longrightarrow \blacktriangleright$

$\langle \text{ParenthesizedExpr} \rangle ::= \blacktriangleright (- \langle \text{Expression} \rangle -) \longrightarrow \blacktriangleright$

$\langle \text{ContextItemExpr} \rangle ::= \blacktriangleright . \longrightarrow \blacktriangleright$

$\langle \text{String} \rangle ::= \blacktriangleright \langle \text{STRING} \rangle \longrightarrow \blacktriangleright$

A.3 Startsymbol

$\langle \text{Start} \rangle ::= \blacktriangleright \langle \text{Path} \rangle - \backslash \mathbf{n} \longrightarrow \blacktriangleright$

Literaturverzeichnis

Appel, Andrew W. (2002): *Modern compiler implementation in Java*. Cambridge Univ. Press.

Berglund, Anders, Boag, Scott, Chamberlin, Don, Fernández, Mary F., Kay, Michael, Robie, Jonathan und Siméon, Jérôme (2005): XML Path Language (XPath) 2.0. W3C Working Draft. URL <http://www.w3.org/TR/2005/WD-xpath20-20050404/>.

Bird, Steven, Chen, Yi, Davidson, Susan, Lee, Haejoong und Zheng, Yifeng (2005): Extending XPath to Support Linguistic Queries. In *Workshop on Programming Language Technologies for XML (PLAN-X)*. Long Beach, California. URL <http://www ldc.upenn.edu/Projects/QLDB/lpath.pdf>.

Bow, Cathy, Hughes, Baden und Bird, Steven (2003): Towards a General Model of Interlinear Text. In *Proceedings of EMELD 2003*. URL <http://www.emeld.org/workshop/2003/bowbadenbird-paper.pdf>.

Burnard, Lou et al. (2005a): Xaira CQL Wiki. Work in progress, URL <http://tonydodd.net/wiki/index.php?XairaCqlWiki>.

Burnard, Lou et al. (2005b): The Xaira Project. URL <http://www.oucs.ox.ac.uk/rts/xaira/>.

Buxton, Stephen und Rys, Michael (2003): XQuery and XPath Full-Text Requirements. W3C Working Draft. URL <http://www.w3.org/TR/2003/WD-xquery-full-text-requirements-20030502/>.

Cassidy, Steve (2002): XQuery as an Annotation Query Language: a Use Case Analysis. In *Proceedings of LREC*. Las Palmas, Spain. URL <http://www ldc.upenn.edu/Projects/QLDB/cassidy-lrec.pdf>.

Cassidy, Steve (2003): Generalizing XPath for Directed Graphs. In *Proceedings for the Extreme Markup Languages conference*. URL

- <http://www.mulberrytech.com/Extreme/Proceedings/html/2003/Cassidy01/EML2003Cassidy01.html>.
- Cassidy, Steve und Harrington, Jonathan (2001): Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1-2):61–77. URL <http://www.sciencedirect.com/science/article/B6V1C-4tSBGXX-4/2/d47065f2eb20f6d80ceda3aa222fcab7>.
- Clark, James und DeRose, Steve (1999): XML Path Language (XPath). Version 1.0. W3C Recommendation. URL <http://www.w3.org/TR/1999/REC-xpath-19991116>.
- Dipper, Stefanie, Faulstich, Lukas, Leser, Ulf und Lüdeling, Anke (2004): Challenges in Modelling a Richly Annotated Diachronic Corpus of German. In *Workshop on XML-based richly annotated corpora*. Lisbon, Portugal. URL http://www.informatik.hu-berlin.de/Forschung_Lehre/wbi/publications/2004/xbraco4_final.pdf.
- Ewert, Stefan (2005): *The CQP Query Language Tutorial*. Institut für Maschinelle Sprachverarbeitung (IMS), Universität Stuttgart. URL <http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/CQPTutorial/cqp-tutorial.pdf>.
- Faulstich, Lukas (2005): The Corpus Data Model CDM2. Work in progress.
- Faulstich, Lukas C. und Leser, Ulf (2005): Implementing Linguistic Query Languages Using LoToS. To be published.
- Faulstich, Lukas C., Leser, Ulf und Lüdeling, Anke (2005): Storing and Querying Historical Texts in a Relational Database. Informatik-Bericht 176, Institut für Informatik der Humboldt-Universität, Berlin.
- Gamma, Erich, Helm, Richard, Johnson, Ralph und Vlissides, John (2001): *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, Bonn. ISBN 3-8273-1862-9.
- Glück, Helmut (2000): *Metzler Lexikon Sprache*. Nummer 34 in Digitale Bibliothek. Metzler, Stuttgart.
- Grust, Torsten, Keulen, Maurice Van und Teubner, Jens (2004): Accelerating XPath evaluation in any RDBMS. *ACM Trans. Database Syst.*, 29(1):91–131. ISSN 0362-5915. DOI: 10.1145/974750.974754. URL <http://www.inf.uni-konstanz.de/~grust/files/accelerating-locsteps.pdf>.

- Iacob, Emil (2005): The Extended XPath language (EXPath) for querying Concurrent Markup Hierarchies. URL <http://dblab.csr.uky.edu/~eiacoo/docs/expath/>.
- Iacob, Ionut E. und Dekhtyar, Alex (2005): Towards a Query Language for Multihierarchical XML: Revisiting XPath. In *The Eighth International Workshop on the Web and Databases (WebDB 2005)*. URL <http://dblab.csr.uky.edu/~eiacoo/publications/webdb05.pdf>.
- Ide, Nancy, Bonhomme, Patrice und Romary, Laurent (2000): XCES: An XML-based Standard for Linguistic Corpora. In *Proceedings of the Second Language Resources and Evaluation Conference (LREC)*, Seiten 825–830. URL <http://www.cs.vassar.edu/~ide/papers/xces-lrecoo.ps>.
- Kay, Michael et al. (2003): XSL Transformations (XSLT) Version 2.0. W3C Working Draft. URL <http://www.w3.org/TR/2003/WD-xslt2-20031112/>.
- Kroymann, Emil, Thiebes, Sebastian, Lüdeling, Anke und Leser, Ulf (2004): Eine vergleichende Analyse von historischen und diachronen digitalen Korpora. Technical Report 174, Institut für Informatik, Humboldt-Universität, Berlin. URL <http://www.deutschdiachrondigital.de/publikationen/TRHistorischeKorpora.pdf>.
- König, Esther, Lezius, Wolfgang und Voormann, Holger (2003): *TIGER-Search 2.1 User's Manual*. IMS, Universität Stuttgart. URL <http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/manual.shtml>.
- Lai, Catherine und Bird, Steven (2004): Querying and Updating Treebanks: A Critical Survey and Requirements Analysis. In *Proceedings of the Australasian Language Technology Workshop*.
- Lezius, Wolfgang (2002): *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Dissertation, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Stuttgart.
- Lüdeling, Anke, Donhauser, Karin et al. (2003): Deutsch Diachron Digital – Ein digitales Referenzkorpus für das Deutsche. Faltblatt. URL <http://www.deutschdiachrondigital.de/publikationen/DeutschDiachron.pdf>.
- Lüdeling, Anke, Poschenrieder, Thorwald und Faulstich, Lukas (2004): DeutschDiachronDigital – Ein diachrones Korpus des Deutschen. *Jahrbuch für Computerphilologie*. URL <http://www.deutschdiachrondigital.de/publikationen/ddd-computerphilologie.pdf>.

- Marx, Maarten (2004): Conditional XPath, the first order complete XPath dialect. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Seiten 13–22. ACM Press, New York, NY, USA. ISBN 158113858X/04/06. DOI: <http://doi.acm.org/10.1145/1055558.1055562>.
- Randall, Beth (2005): *CorpusSearch 2 User's Guide*. University of Pennsylvania. URL <http://corpussearch.sourceforge.net/CS-manual/Contents.html>.
- Rohde, Douglas L. T. (2005): *TGrep2 User Manual*. Version 1.15, URL <http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf>.
- Schmidt, Thomas (2003): Visualising Linguistic Annotation as Interlinear Text. Arbeiten zur Mehrsprachigkeit Serie B (46), Universität Hamburg, Hamburg. URL <http://www.rrz.uni-hamburg.de/exmaralda/Daten/4D-Literatur/Visualising-final.pdf>.
- Sperberg-McQueen, C.M. und Huitfeldt, Claus (2004): *Digital Documents: Systems and Principles: 8th International Conference on Digital Documents and Electronic Publishing, DDEP 2000; 5th International Workshop on the Principles of Digital Document Processing, PODDP 2000*, Band 2023 von *Lecture Notes in Computer Sciences*, Kapitel GODDAG: A Data Structure for Overlapping Hierarchies, Seiten 139–160. Springer. URL <http://www.springerlink.com/openurl.asp?genre=article&id=98J1VBU5NBY73UL3>.
- Taylor, Claire Louise (2003): *XSLT as a Linguistic Query Language*. Honours thesis, Department of Computer Science and Software Engineering, University of Melbourne. URL <http://eprints.unimelb.edu.au/archive/00000500/>.
- TEI-P5 (2005): *TEI: The P5 Release*. Text Encoding Initiative Consortium. URL <http://www.tei-c.org/P5/>.
- Trißl, Silke und Leser, Ulf (2005): Querying Ontologies in Relational Database Systems. In Bertram Ludäscher und Louiqa Raschid (Herausgeber), *Data Integration in the Life Sciences: Second International Workshop*. Springer. ISBN 3-540-27967-9. DOI: 10.1007/11530084_7.
- Vitt, Thorsten (2004): *Speicherung linguistischer Korpora in Datenbanken*. Studienarbeit, Humboldt-Universität zu Berlin. URL <http://www.informatik.hu-berlin.de/~vitt/stud/studienarbeit/slk.pdf>.

Voormann, Holger (2002): *TIGERin – Grafische Eingabe von Suchanfragen in TIGERSearch*. Diplomarbeit, Fakultät Informatik, Universität Stuttgart.
URL <http://www.ims.uni-stuttgart.de/projekte/corplex/paper/voormann/tigerin-diplom.pdf>.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 18. Oktober 2005

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 18. Oktober 2005