3

3 Browser und Parser

3.1 Der Browser als Parser

Der Web Browser ist die meistverwendete Software in der Geschichte der Informatik. 2013 sind gemäss ITU über 2.7 Milliarden Menschen Online – also ca. 40% aller Menschen (in Europa sind es 75% aller Menschen, in Nordamerika 82% und in der Schweiz 85%), die alle im Minimum einen Web Browser verwenden. Gemäss Google soll sich diese Zahl in den nächsten Jahren verdoppeln. 1991 wurde von Tim Berners-Lee der erste Web Browser, ein einfacher HTML Editor, entwickelt. 1993 folgte mit Mosaic der erste graphische Web Browser. 1995 wurden Netscape und der Microsoft Internet Explorer in ihrer ersten Version ausgeliefert. Heute existieren über 60 verschiedene Web Browser und jedes Jahr werden fünf weitere entwickelt.

Ein Web Browser wird eingesetzt, um Informationen zu finden und darzustellen. Diese Seiten werden mittels HyperText Markup Language (HTML) beschrieben. In vielen Fällen wird die Darstellung mit Cascading Style Sheets (CSS) erweitert und die Interaktivität wird mit JavaScript verbessert.

Bei genauerem Hinsehen entpuppt sich ein Browser als Sprachinterpreter oder auch Parser. Jeder Browser transformiert Informationen, die in einer Wohldefinierten Sprache – z.b. HTML – formuliert sind.

Daraus folgt eine Reihe von Thesen:

- HTML ist eine kontextfreie Sprache, welche mittels XML-Schema definiert worden ist.
- Ein HTML Browser ist ein Parser, der eine kontextfreie formale Sprache verarbeitet.
- Der Web Browser hat den Compiler als wichtigstes Instrument zur Verarbeitung formaler Sprachen abgelöst.
- Der Web Browser ist die am weitesten verbreitete informationsverarbeitende Maschine.

3.1.1 Der Browser als sprachverarbeitende Maschine

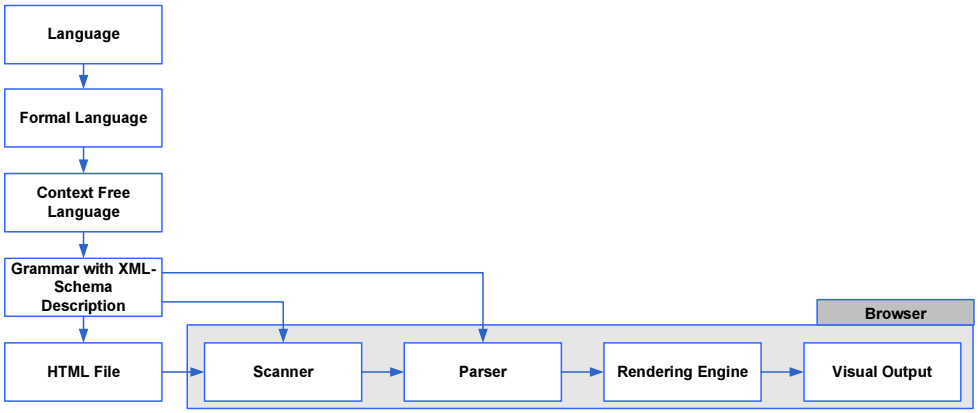

Abbildung 3.1 Der Browser als Sprachverarbeitende Maschine

Ein Web Browser kann als Sprachverarbeitende Maschine dargestellt werden (**Abbildung 3.1**). Die Beschreibung des Seitenaufbaus einer Web Page erfolgt in einer *Sprache*. Diese Sprache ist eine *formale Sprache*, da ein Browser sie widerspruchsfrei interpretieren können muss. Die *Grammatik* der Sprache ist mittels einer *XML-DTD* (XML Document Type Definition) beschrieben. Dieselbe Grammatik ist die Voraussetzung, dass der *Scanner* die zugelassenen Sprachelement erkennt und der *Parser* den Parse-Tree erzeugen kann, der vom *Visual Generator* interpretiert werden kann. Die Interpretation ist nichts anderes als der Aufbau der darzustellenden Seite.

3.1.2 SGML

Die Idee der Trennung von Form und Inhalt in elektronischen Dokumenten geht auf ein Meeting der Graphic Communications Association (GCA) in den Büroräumlichkeiten der Canadian Government Printing Office im September 1967 zurück. Das von William Tunnicliffe definierte Traktandum war: "The separation of the information content of documents from their format ". Der SGML Vorläufer GML (Generalized Markup Language) wurde von Charles Goldfarb, Edward Mosher and Raymond Lorie, Mitglieder des IBM Forschungsteams "law office information systems" 1969 definiert [Floyd 1998].

Bei SGML handelt es sich um die Standard Generalized Markup Language. Der ISO (ISO 8879-1986) Standard definiert SGML als "eine Sprache für Dokumentenrepräsentation, welche Markup formalisiert und von System- und Verarbeitungsabhängigkeiten löst". SGML erlaubt den Austausch von grossen und komplexen Datenmengen und vereinfacht den Zugriff auf sie. Zusätzlich zu den Möglichkeiten des deskriptiven Markups benutzen SGML-Systeme ein Dokumentenmodell, welches die Überprüfung der Gültigkeit eines Textelements in einem bestimmten Kontext erlaubt (Validierung).

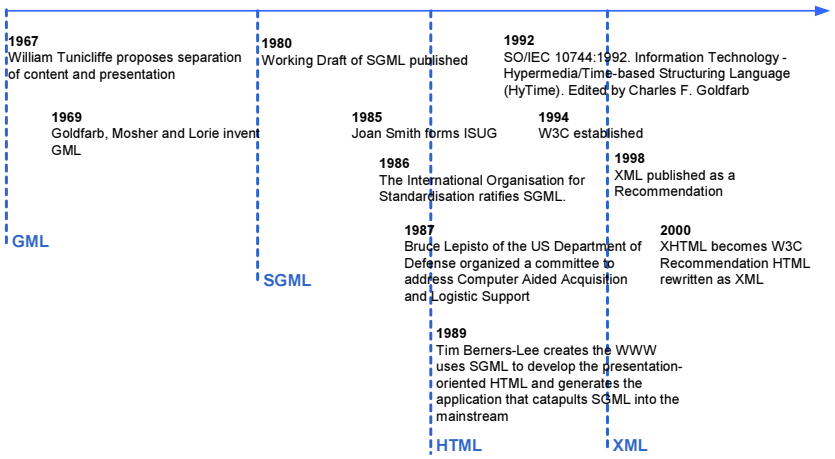

Abbildung 3.2 Die Entwicklung von SGML

Die geschichtliche Entwicklung von SGML ist in **Abbildung 3.2** dargestellt. SGML ist in gewissen Sinne der Vorläufer von XML, der formalen Basissprache des Internets.

SGML enthält Techniken, welche den Benutzer folgendes erlauben:

- Zusammenbinden von Dateien, zur Erstellung eines zusammenhängenden Dokuments.
- Einbindung von Illustrationen und Tabellen in Textdateien.
- Erzeugen von mehreren Versionen eines Dokuments in einer einzigen Datei.
- Hinzufügen von Kommentaren in die Datei.
- Kreuzreferenzen und Inhaltsverzeichnisse.

3.1.3 Web Browser Reference Architecture

Alan Grosskurth und Michael Godfrey von der School of Computer Science der University of Waterloo in Kanada haben eine Web Browser Reference Architecture entworfen [Grosskurth, Godfrey 2005], die alle grundlegenden Komponenten eines Web Browsers definieren. Diese Referenz soll dazu dienen, die bestehenden Web Browser besser zu verstehen und das Design künftiger Web Browser zu vereinfachen. Die Web Browser Reference Architecture ist in **Abbildung 3.3** dargestellt.

Die zentralen Elemente der Reference Architecture sind die Browser Engine und Rendering Engine. Die Browser Engine steuert das Gesamtsystem, sie ruft, abhängig vom Inhalt einer HTML, verschiedenen Parser, Interpreter oder Plugins auf, steuert die Darstellung und greift auf den Browser Cache zu, respektive verwaltet den Inhalt des Cache. Die Rendering Engine ist für das Parsing der HTML Dokumente zuständig ist. Die Rendering Engine ruft gegebenenfalls den JavaSrcipt Interpreter und den XML Parser auf und übergibt transformiert die in HTML, XML oder JavaScript formulierten Informationen in eine für den User verständliche visuelle Form.

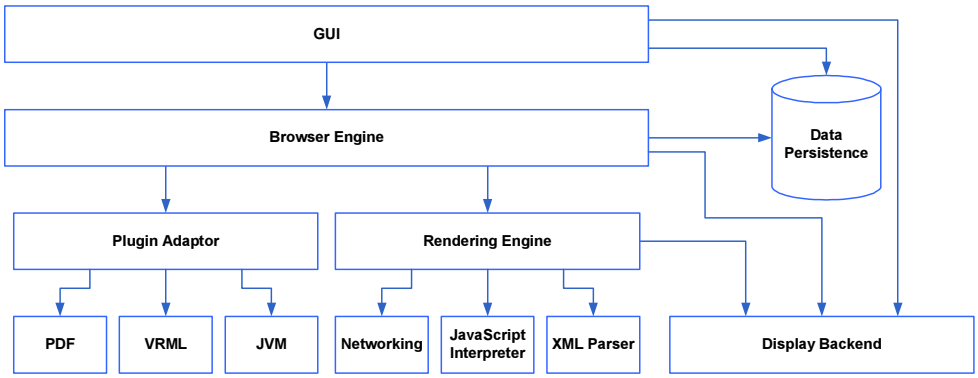

Abbildung 3.3 Browser Reference Architecture

Die logischen Komponenten der Web Browser Reference Architecture sind:

GUI: Das User Interface des Browsers.

- *Browser Engine:* High-Level Interface zur Manipulation und Steuerung der Rendering Engine und des Plugin Adaptors.
- *Rendering Engine:* Sie führt das Parsing eines HTML Dokumentes durch. Die Rendering Engine enthält den Scanner und den HTML und CSS Parser.
- *Plugin Adaptor:* Adaptor für die Darstellung und Interpretation von speziellen Dateiformaten (PDF, VRML) oder für spezielle Zwecke.
- *Networking:* Die Schnittstelle zum Netzwerk.
- *JavaScript Interpreter:* Clientside Computation wird durch die Programmiersprache JavaScript formuliert. Der Interpreter arbeitet die entsprechenden Statements ab und führt die entsprechenden Berechnungen aus.
- *Display Backend:* Schriften, graphische Primitiven, User Interface Komponenten (Widgets) und andere visuelle Routinen werden durch das Display Backend zur Verfügung gestellt.
- *Data Persistence:* Der Browser Cache

3.2 Natürliche Sprache – gesprochene Information

Die gesprochene Sprache ist nach wie vor das am meisten verwendete Instrument zum Austausch von Informationen. Die Voraussetzung für unseren Diskurs wird vom Sprachwissenschaftler F. De Saussure mit "Sprechen ist Handeln" umschrieben. Pragmatischer Begründungssatz: *Das Sprechen als Rede (Parole) ausgeführt ist beliebig oft wiederholbar, weil man die Sprache (Langue) beherrscht (***Abbildung 3.4***).*

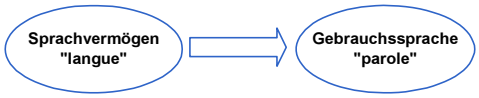

Abbildung 3.4 Zusammenhang und Verwendung der natürlichen Sprache

Der zentrale Unterschied zwischen natürlichen und formalen Sprachen liegt in der Dynamik des Kontextes und der Verwendung der entsprechenden Sprachen.

Natürliche Sprachen werden mittels Konstituenz (Wortfolgen), Dependenz (Abhängigkeiten) und Valenz (Kontext) beschrieben.

Formale Sprachen werden mittels Wörtern und Alphabeten beschrieben.

Die „Interpretation“ natürlicher Sprachen erfolgt durch Wissensbasierte Systeme während die „Interpretation“ formaler Sprachen durch die Realisierung von Automaten durchgeführt wird.

3.2.1 Definitionen

3.2.1.1 Konstituenz

Die Konstituenz ist eine Kategorisierung von Einheiten (Wörtern) durch Phrasen (Konstituenten = Wortfolgen). Verschiedene Einheiten werden aufgrund ihres analogen syntaktischen Verhaltens zu einer Kategorie zusammengefasst. Die Kriterien zur Zusammenfassbarkeit sind Ersetzbarkeit, Verschiebeprobe und Pronominalisierung.

3.2.1.2 Dependenz

Die Dependenz ist eine Relationierung von Einheiten (Wörtern). Die Instanzen einer Dependenz heissen Dependenzgrammatiken und sind hierarchisch angeordnet. Die Kriterien zur Anordnung sind die Abhängigkeit von Wortformen und die Bedingtheit des Auftretens von Wörtern mit anderen.

3.2.1.3 Valenz

Die Valenz erlaubt eine Zuordnung funktionaler Rollen der Einheiten mittels lexikalischer Information (Quantitative, Qualitative und Selektionale Valenz). Wörter werden über ihre Valenz aufgrund ihres Kontextes zugeordnet.

3.2.1.4 Lexem

Als Lexem wird in der Sprachwissenschaft die ungebeugte Grundform eines Wortes oder auch der Wortstamm bezeichnet.

3.2.2 Grammatik natürlicher Sprachen

Die zentrale Voraussetzung zum Informationsaustausch ist die Grammatik einer natürlichen Sprache. Der Philosoph G. W. F. Hegel definierte 1809 in seiner Gymnasialrede Grammatik: „*Die Grammatik hat nämlich die Kategorien, die eigentümlichen Erzeugnisse und Bestimmungen des Verstandes zu ihrem Inhalte, in ihr fängt also der Verstand selbst an, gelernt zu werden. (Ihre) Abstraktionen aber sind das ganz Einfache. Sie sind gleichsam die einzelnen Buchstaben,..., mit denen wir anfangen, um es zu buchstabieren, und dann lesen zu lernen.*“

Der Sprachwissenschaftler Noam Chomsky formuliert die Grammatik als Regelsystem: „*Unter einer generativen Grammatik verstehe ich einfach ein Regelsystem, das auf explizite und Wohldefinierte Weise Sätzen Struktur-Beschreibungen zuordnet.*“

Definition einer Dependenz Grammatik einer natürlichen Sprache:
Eine Dependenz Grammatik gibt die Regeln an, wie die Abhängigkeiten der Lexeme einer natürlichen Sprache über einem gegebenen Alphabet gebildet werden.

Formale Definition:
Eine Dependenz Grammatik DG ist ein Tupel (R, L, C, F)

R: Eine Menge der Dependenzregeln

L: Eine Menge von Lexemen

C: Eine Menge von Wortklassen

F: Eine Zusweisungsfunktion der Lexeme auf die Wortklassen F: L→C

3.2.3 Implizite Voraussetzung zu Kommunikation: Geltungsansprüche

Der deutsche Philosoph Jürgen Habermas versteht in seiner Theorie des kommunikativen Handelns die Rede als eine "System universaler und notwendiger Geltungsansprüche" [Habermas 1984]. Wenn die hörende Person einen von der sprechenden Person erhobenen Geltungsanspruch akzeptiert, erkennt er die Gültigkeit der geäusserten symbolischen Gebilde an. Dies bedeutet, dass ein Konsens durch Annerkennung der Geltungsansprüche besteht.

Die Geltungsansprüche bestehen aus den vier Bereichen Wahrhaftigkeit, Richtigkeit, Wahrheit der Aussage und die Syntax der Sprache, wie in **Abbildung 3.5** skizziert. Vorbedingung für jede Kommunikation ist die Verständlichkeit des Gesagten und die Akzeptanz der Geltungsansprüche. Im Rahmen der Kommunikation mit einer natürlichen Sprache spielen zusätzlich die korrespondierenden Intentionen, das Gewissheitserlebnis und die Erfahrungsgrundlage eine Rolle.

Bedingung der Kommunikation	Geltungsansprüche	Korrespondierende Intentionen	Gewissheits-erlebnis	Erfahrungs-grundlage
Verständlichkeit		etwas verstehen	nicht-sinnliche Gewißheit	Zeichenwahrnehmung
	Wahrhaftigkeit	jemandem glauben	Glaubensge-wissheit	Interaktionserfahrungen mit Personen und deren Äußerungen
	Richtigkeit	von etwas überzeugt sein	-	keine unmittelbare
	(Aussagen-) Wahrheit	etwas wissen	-	keine unmittelbare
	Syntax	etwas sehen, wahrnehmen	sinnliche Gewissheit	Ding-Ereignis-Wahrnehmung

Abbildung 3.5 Tafel der Geltungsansprüche von Habermas

Die Geltungsansprüche für die Kommunikation mit einer natürlichen Sprache sind:

- *Verständlichkeit:* Ein Satz ist grammatisch richtig und damit verständlich.
- *Wahrheit:* Eine Aussage ist wahr
- *Wahrhaftigkeit:* Die beabsichtigte Aussage ist wahrhaftig
- *Richtigkeit:* Eine Aussage ist korrekt

3.2.3.1 Die Regeln zur Verständlichkeit von Winograd

In seinem Buch "Language as a Cognitive Process" [Winograd 1983] umschreibt der amerikanische Wissenschaftler Terry Allen Winograd, Professor für Informatik an der Universität Stanford, unsere Sprachfähigkeit durch eine Reihe von allgemein anerkannten Regeln.

Regeln zur Anordnung der Wörter (Word order rules): Die Anordnung der Wörter in einem Satz kann die Bedeutung des Gesprochenen vollständig ändern. Diese Regeln sind von Sprache zu Sprache verschieden.

- *Vokabular und Wortstruktur (Vocabulary and Word Structure):* Zusätzlich zu der Form und der Bedeutung der Wörter bestimmt der Kontext die Bedeutung des Gesagten.
- *Semantik (Semantic Features):* Die Klassifikation der Wortarten ist entscheidend für die Zuordnung zur Bedeutung.
- *Referenzen (Reference):* Jede Sprache hat Mechanismen, die einen Bezug auf bereits für die sprechende oder für die zuhörende Person Bekanntem erlauben.
- *Zeit (Time):* Die Umschreibung oder auch die syntaktische Definition von Zeiten wird in jeder Sprache verwendet.
- *Gesprächsstruktur (Discourse Structure):* Um ein Gespräch zu verstehen, sind in einem bestimmten Kontext eine Menge von impliziten Annahmen notwendig, die auf dem bereits Gesagten beruhen.
- *Ausrufe (Attitude Message):* Ausrufe haben eine besondere Bedeutung, die nicht auf dieselbe Art und Weise wie die Bedeutung von Substantiven verstanden werden kann.

- *Betonung (Prosodic Conventions):* Die Betonung wird nur teilweise in der geschriebenen Sprache durch Satz-Zeichen umschrieben.
- *Sprachstile (Style Conventions):* Eine gesprochene Sprache ist nicht nur durch grammatikalische Regeln definiert. Zusätzlich werden situationsbezogen bestimmte Sprachstile eingesetzt.
- *Allgemeinwissen (World Knowledge):* Der Wissenstand der sprechenden Gesellschaft ist entscheidend für das Verständnis der Bedeutung des Gesprochenen.

3.2.4 Modellierung der natürlichen Sprache

Die Modellierung der natürlichen Sprache geht von einem kybernetischen Modell des „Language Users“ aus, welcher wiederum auf dem kybernetischen Modell des Menschen (siehe Einleitung) basiert. Das Gehörte wird über so genannte „Sound Patterns“ interpretiert und anschliessend über die inhaltliche Struktur in eine für das Hirn verarbeitbare Repräsentation überführt. Dieses innere Bild der äusseren Welt ist die Basis des Denkens und des Handelns, welche sich als Konsequenz aus dem Gehörten ergeben.

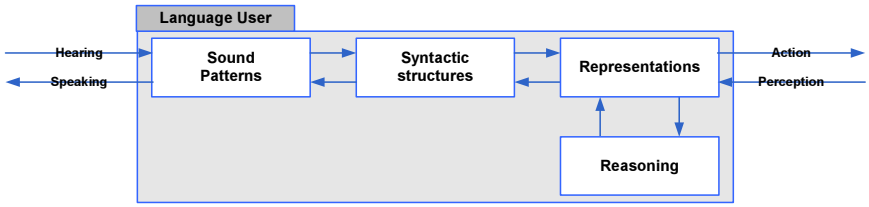

Abbildung 3.6 Der „Language Processing User“

Die Basisannahmen zur Modellierung natürlicher Sprachen sind:

- Die natürliche Sprache ist eine Menge von Informations-Ressourcen, die von der sprechenden Person manipuliert werden können.
- Es besteht eine gegenseitige Abhängigkeit zwischen der sprechenden Person, deren Intentionen, dem bereits Gesagtem, dem Kontext und der zuhörenden Person (z.b. dem Computer). Diese Reflexivität ist eine der zentralen Eigenschaften der natürlichen Sprachen.
- Sämtliche Prozesse und Repräsentationen, die mit der Kommunikation durch natürliche Sprachen verbunden sind, können als Datenstrukturen abgebildet werden.

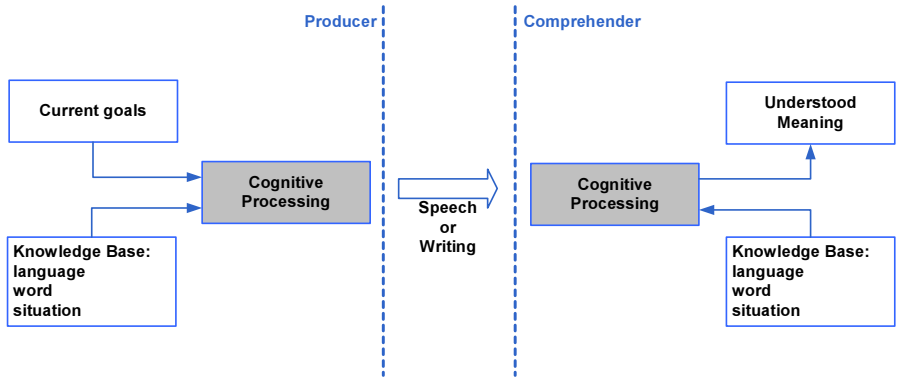

Abbildung 3.7 Basismodell der gemeinsamen Kommunikation

Die gemeinsame Basis zwischen *Producer* und *Comprehender* ist die gemeinsame Wissensbasis, die aus der Sprache, den Wörtern und der Situation besteht. Der *Producer* verfolgt mit dem Gesagten ein bestimmtes Ziel, welches er mittels kognitiver Prozesse in Sprache umsetzt. Der *Comprehender* versteht das Gesagt mittels kognitiver Prozesse. Ohne gemeinsame Basis ist ein Verständnis nicht möglich.

3.2.4.1 Natural Language Processing Systems

Die Problematik der Realisierung von Natural Language Processing Systems ist eng verbunden mit der Problematik der Künstlichen Intelligenz. Eine vollständige und fehlerfreie Umsetzung solcher Systeme wird erst dann gelingen, wenn die Realisierung von KI Systemen gelingen wird. Dies wiederum hängt vom Verständnis der Menschlichen Intelligenz ab. Das Verstehen einer natürlichen Sprache lässt sich aus Sicht einer Maschine auf die Verwaltung von Hypothesen reduzieren, während das Erzeugen einer natürlichen Sprache auf einen Entscheidungsprozess hinausläuft

3.2.4.2 Datalog

Datalog wurde als portables Natrual Language Processing System von General Motors Research Lab 1985 entwickelt. Datalog ist ein Sprachinterface zu einer Datenbank ("database dialogue") [Hafner, Godden 1985]. Datalog basiert auf einer "Cascaded ATN" (CATN) Grammatik, die 1980 vom William A. Woods, einem Sprachwissenschaftler der Universität Cambridge, entwickelt worden ist [Woods 1989]. Diese Grammatik beruht auf einem Netzwerk von verschiedenen Zuständen, die miteinander verbunden sind. Diese Verbindungen beschreiben Zustandsübergänge, die entweder einzelne Wörter interpretieren oder jedoch ganze Satzteile in einen anderen Bereich des Netzwerkes verschieben, damit diese interpretiert werden können. Die Interpretation natürlicher Sprache wird durch CATN erleichtert, da dieselbe Phrase durch mehrere Teilbäume der Grammatik interpretiert werden können. Eine schrittweise Verfeinerung der Interpretation erlaubt eine bessere Interpretation des Gesprochenen. Die Systemarchitektur von Datalog ist in **Abbildung 3.8** dargestellt.

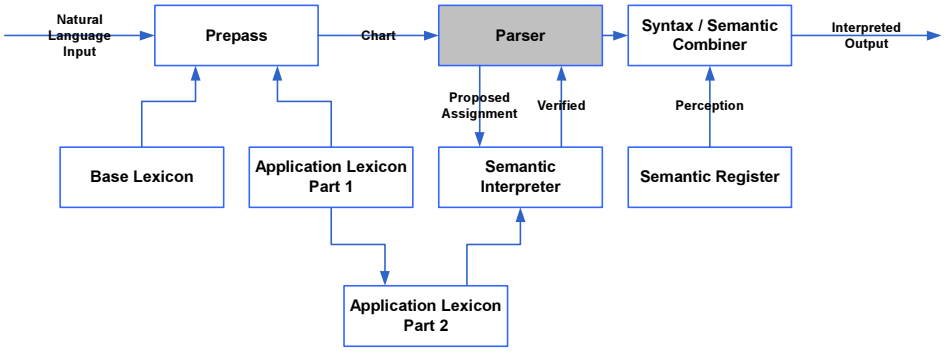

Abbildung 3.8 Datalog System Architektur

- *Prepass:* Der Input in natürlicher Sprache wird in eine so genannte "Chart" Struktur aufgeteilt. Diese Struktur beschreibt ein Netzwerk von Phrasen.
- *Parser:* Der Parser baut aufgrund der CATN Grammatik einen Parse-Tree auf.
- *Semantic Interpreter:* Diese Komponente interagiert mit dem Parser. Sie akzeptiert syntaktische Zuweisungen, die der Parser aufgrund der Grammatik vorschlägt und versucht aufgrund dieser Zuweisung eine logische Repräsentation der Phrase aufzubauen.

3.2.4.3 KBNL

Das System KBNL (Knowledge Based Natural Language) basiert auf einer Kombination von Supervised Learning und Knowledge Representation. Zentraler Bestanteil ist die interne mehrdimensionale Repräsentation (World Knowledge) in einem eigenen Format [Barnett 1990]. Das System ist in **Abbildung 3.9** dargestellt.

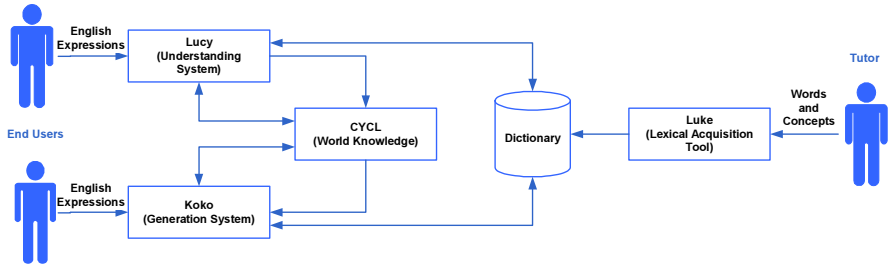

Abbildung 3.9 KBNL System Architektur

- *Lucy:* Ein Wissensbasiertes System, das die Englische Sprache versteht.
- *Koko:* Ein Wissensbasiertes System, welches Phrasen in Englischer Sprache generiert.
- *Luke:* Ein Werkzeug, welches ein Dictionary aufbaut, welches die Abbildung Sprache – Wissen enthält und von Lucy und Koko verwendet werden kann.

Das Modell der Sprachverarbeitung, welches von KBLN verwendet wird, basiert auf Wörterbüchern, Grammatiken und einem Regelwert zur semantischen Interpretation von Phrasen.

3.2.4.4 Natural Language Generation

Natural Language Generation (NLG) Systeme sind Applikationen, die verständliche Texte in einer natürlichen Sprache erzeugen können. Es werden zwei Arten von System unterschieden, die Assistent Systems, die einen Autor / eine Autorin in der Formulierung unterstützen und diejenigen Systeme, die direkt gesprochene Sprache generieren.

Tabelle 3.1 Assistant Systems:

Name	Jahr	Anwendung
FOG	1994	Unterstützung zur Bereitstellung von Wettervorhersagen
PlanDoc	1994	Vorschläge zur Dokumentation von Resultaten aus der Netzwerksimulation
AlethGen	1996	Vorschläge zur Beantwortung von Kundenanfragen und Reklamationen
Drafter	1995	Unterstützung bei der Erstellung von Software Manuals

Tabelle 3.2 Der Computer als Autor:

Name	Jahr	Anwendung
ModelExplainer	1997	Generierung der Beschreibung von Klassen in einem OO-System
Knight	1997	Erklärung von Informationen, die aus einer AI Knowledge Base stammen
LFS	1992	Zusammenfassung statistischer Daten
Piglet	1995	Erklärung der Krankheitsgeschichte für Patienten

3.3 Formale Sprachen – Informationen für Maschinen

Formale Sprachen sind Wohldefinierte Konstrukte, die mit Widerspruchsfreien Grammatiken definiert werden. Weit verbreitet sind Kontext-Freie formale Sprachen, wie die meisten Programmier- und Script-Sprachen oder auch HTML.

3.3.1 Definitionen

Alphabet
Ein Alphabet $\sum$ ist eine nichtleere, endliche Menge von Zeichen, auch Buchstaben (Letters) oder Terminale (Terminal) genannt.

Wort

Ein Wort w über einem Alphabet ∑ ist eine endliche Folge von Zeichen aus ∑.

∑* ist die Menge aller Worte über ∑.

Ein spezielles Wort ist, das leere Wort (der Länge 0) das aus keinem Zeichen aus ∑ besteht.

Sprache

Eine Sprache L ist eine Teilmenge von ∑*.

Grammatik

Eine Grammatik beschreibt Regeln, wie aus einen Alphabet ∑ die Worte w einer bestimmten Sprache L gebildet werden.

3.3.2 Grammatik formaler Sprachen

Sämtliche formalen Sprachen, die durch Maschinen verarbeitet werden, sind durch Grammatiken definiert. Eine Grammatik beschreibt also die entsprechende Sprache.

Definition einer Grammatik

Eine Grammatik gibt die Regeln an, wie eine formale Sprache über einem gegebenen Alphabet gebildet wird.

Formale Definition

Eine Grammatik G ist ein Tupel (∑, V, R, S)

∑: Das Alphabet, d.h. die Menge der Zeichen (Terminalzeichen, Keywords, Terminals)

V: Eine Menge von Variablen

R: Eine Menge von Regeln, die Produktionsregeln (derivation rules)

S: Das Startzeichen / Startsymbol

Die Eigenschaften einer Grammatik sind:

Keine Variable darf gleichzeitig Keyword sein.

Variablen sind Platzhalter, die in folgenden Ersetzungsschritten durch die Anwendung von Produktionsregeln ersetzt werden.

Die Menge von Regeln ist endlich.

Das Startzeichen ist ein Element der Variablenmenge.

Eine Grammatik G erzeugt eine Sprache L. Dabei gehört ein Wort w aus ∑* zur Sprache L.

Definition einer durch eine Grammatik erzeugte Sprache

$L(G) = \{w \varepsilon \sum^* \mid S \rightarrow^* w\}$

3.3.3 Chomsky Hierarchie

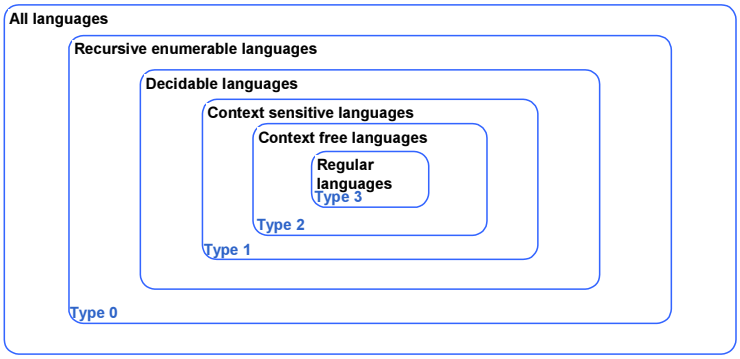

Abbildung 3.10 Chomsky Hierarchie der Sprachen

Noam Chomsky, Professor für Linguistik am MIT, definierte 1965 in seinem Artikel "Aspects of the Theory of Syntax" [Chomsky 1965] eine Hierarchie aller Sprachen (**Abbildung 3.10**).

- *All Languages:* Die Menge aller Sprachen.
- *Recursively Enumerable Language (Type 0):* Diese Sprachen sind durch Grammatiken definiert, für deren Produktionsregeln es keine Einschränkung gibt.
- *Decidable Languages:* Für die Sprachen des Typs 1, 2 und 3 gibt es einen Algorithmus, der bei Eingabe einer Grammatik und eines Wortes feststellt, ob das betreffende Wort Teil der Sprache ist oder nicht.
- *Context-Sensitive Language (Type 1):* Die kontextsensitiven Sprachen sind die Produktionsregeln der Grammatiken abhängig vom Kontext des gerade betrachteten Wortes.
- *Context-Free Language (Type 2):* Die Produktionsregeln von kontextfreien Grammatiken sind unabhängig vom Kontext des betrachteten Wortes. Es sind lediglich Variablen auf der linken Seite einer Produktionsregel erlaubt.
- *Regular Language (Type 3)*: Sprachen, die durch reguläre Grammatiken (Regular Expressions) beschrieben sind.
- Alle Programmier- und Script-Sprachen sind Context-Free Languages und werden mit CF-grammars (Kontext Freie Grammatiken) beschrieben.

3.3.4 Beschreibungsmöglichkeiten von Grammatiken

Grammatiken werden mit Hilfe einer so genannten Metasyntax (Syntax zur Beschreibung der Syntax von Sprachen) beschrieben.

3.3.4.1 Backus-Naur Form (BNF

Die Backus-Naur Form oder ursprünglich Backus Normal Form ist nach John Backus, dem Erfinder von Fortran benannt. Donald Knuth, der Mathematiker und Autor des Informatik-Klassikers "The Art of Computer Programming" [Knuth 1997] hat Peter Naur (Erfinder von Algol 60) als Coauthor dieser Metasyntax aufgeführt. Aus diesem Grunde ist der heute verwendete Name "Backus-Naur Form".

<symbol> ::= <expression with symbols>

<symbol>: Nonterminales Symbol

::= : Zuweisungszeichen

<expression with symbols>: Eine Sequenz von Symbolen und/oder Sequenzen von Symbolen, die mit "|" getrennt sind. Das Ganze kann eine Substitution des <symbol> auf der rechten Seite des Ausdruckes sein.

| : Auswahl

3.3.4.2 Extended Backus-Naur Form (EBNF)

Die Extended Backus-Naur Form (EBNF) wurde von Niklaus Wirth definiert und ist als ISO Standard Definiert [SOEBNF 1996].

<symbol> ::= <expression with symbols>

<symbol>: Nonterminales Symbol

::= : Zuweisungszeichen

<expression with symbols>: Eine Sequenz von Symbolen und/oder Sequenzen von Symbolen, die mit "|" getrennt sind. Das Ganze kann eine Substitution des <symbol> auf der rechten Seite des Ausdruckes sein.

| : Auswahl

[] : Optionale Symbole

* : Sequenz von 0 oder mehreren Ausprägungen eines Symbols (Kleene Closure)

<symbol>: N Ausprägungen eines Symbols.

<symbol>: M Ausprägungen eines Symbols.

Listing 3.1 Beispiel

```
<postal-address> ::= <name-part> <street-address> <zip-part>

<name-part> ::= <personal-part> <last-name> [<jr-part>] <EOL> | <person-
al-part> <name-part>
```

```
<personal-part> ::= <name> | <initial> "."
<street-address> ::= [<apt>] <house-num> <street-name> <EOL>
<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>
```

3.3.4.3 XML EBNF

Zur Definition von XML-DTD's wird eine leicht abgeänderte EBNF Notation verwendet.

<symbol> ::= <expression with symbols>

<symbol>: Nonterminales Symbol

::= : Zuweisungszeichen.

<expression with symbols> : Eine Sequenz von Symbolen .

#xN : N Definiert einen hexadezimaler Integer, die Anzahl vorangehender Nullen ist nicht signifikant.

[a-zA-Z], [#xN-#xN] : Definiert einen Bereich aller Characters zwischen (inklusive) den aufgeführ-ten Characters.

[abc], [#xN#xN#xN] : Aufzählung zugelassener Characters.

[^a-z], [^#xN-#xN] : Definiert alle Characters ausserhalb des angegebenen Bereichs.

[^abc], [^#xN#xN#xN] : Definiert alle Characters ausserhalb der Aufzählung.

"string" : Literal String innerhalb doppelter Hochkommatas.

'string' : Literal String innerhalb Hochkommatas.

A? : A oder kein Symbol; optional A.

A B : A gefolgt von B.

A | B : A oder B.

A – B : Alle Strings die mit A uebereinstimmen, jedoch nicht mit B.

A+ : Eine oder mehr Ausprägungen des Symbols A.

A* : Null oder mehr Ausprägungen des Symbols A.

/* ... */ : Kommentar.

[wfc: ...] : Wohldefinierte Bedingung (well-formedness constraint).

[vc: ...] : Validierte Bedingung (validity constraint).

Listing 3.2 Beispiel Ausschnitt aus der HTML-DTD

```
<!--    html.dtd

        Document Type Definition for the HyperText Markup Language
 (HTML DTD)
$Id: html.dtd,v 1.30 1995/09/21 23:30:19 connolly Exp $
Author: Daniel W. Connolly <connolly@w3.org>
See Also: html.decl, html-1.dtd
  http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html
```

```
-->
<!ENTITY % HTML.Version "-//IETF//DTD HTML 2.0//EN">
<!ENTITY % HTML.Recommended "IGNORE"
<![ %HTML.Recommended [
        <!ENTITY % HTML.Deprecated "IGNORE">
]]>
<!ENTITY % HTML.Deprecated "INCLUDE"
<!ENTITY % HTML.Highlighting "INCLUDE"
<!ENTITY % HTML.Forms "INCLUDE"
<!ENTITY % Content-Type "CDATA"
<!ENTITY % HTTP-Method "GET | POST"
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list " UL | OL | DIR | MENU " >
<!ENTITY % ISOlat1 PUBLIC
  "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML">
%ISOlat1;
<!ENTITY amp CDATA "&#38;"     -- ampersand          -->
<!ENTITY gt CDATA "&#62;"      -- greater than       -->
<!ENTITY lt CDATA "&#60;"      -- less than          -->
<!ENTITY quot CDATA "&#34;"    -- double quote       -->
<!--========== SGML Document Access (SDA) Parameter Entities =====-->
<!-- HTML 2.0 contains SGML Document Access (SDA) fixed attributes
in support of easy transformation to the International Committee
for Accessible Document Design (ICADD) DTD
 "-//EC-USA-CDA/ICADD//DTD ICADD22//EN".
<!ENTITY % SDAFORM  "SDAFORM  CDATA  #FIXED"
<!ENTITY % SDARULE  "SDARULE  CDATA  #FIXED"
<!ENTITY % SDAPREF  "SDAPREF  CDATA  #FIXED"
<!ENTITY % SDASUFF  "SDASUFF  CDATA  #FIXED"
<!ENTITY % SDASUSP  "SDASUSP  NAME   #FIXED"
<![ %HTML.Highlighting [
<!ENTITY % font " TT | B | I ">
<!ENTITY % phrase "EM | STRONG | CODE | SAMP | KBD | VAR | CITE ">
<!ENTITY % text "#PCDATA | A | IMG | BR | %phrase | %font">
<!ELEMENT (%font;|%phrase) - - (%text)*>
<!ATTLIST ( TT | CODE | SAMP | KBD | VAR )
        %SDAFORM; "Lit"
```

```
        >
<!ATTLIST ( B | STRONG )
        %SDAFORM; "B"
        >
<!ATTLIST ( I | EM | CITE )
        %SDAFORM; "It"
        >
<!ENTITY % pre.content "#PCDATA | A | HR | BR | %font | %phrase">
]]>
<!ENTITY % text "#PCDATA | A | IMG | BR">
<!ELEMENT BR    - O EMPTY>
<!ATTLIST BR
        %SDAPREF; "&#RE;"
        >
<!-- <BR>       Line break      -->
<!ENTITY % linkType "NAMES">
<!ENTITY % linkExtraAttributes
        "REL %linkType #IMPLIED
        REV %linkType #IMPLIED
        URN CDATA #IMPLIED
        TITLE CDATA #IMPLIED
        METHODS NAMES #IMPLIED
        ">
<![ %HTML.Recommended [
        <!ENTITY % A.content   "(%text)*"
]]>
<!ENTITY % A.content   "(%heading|%text)*">
<!ELEMENT A     - - %A.content -(A)>
<!ATTLIST A
        HREF CDATA #IMPLIED
        NAME CDATA #IMPLIED
        %linkExtraAttributes;
        %SDAPREF; "<Anchor: #AttList>"
>.
```

3.3.4.4 Syntax Diagramme

Eine weit verbreitete Anwendung der Darstellung einer Grammatik durch eine Metasyntax ist die Verwendung von Syntax-Diagrammen. Dabei kommen zwei Grundsymbole zum Einsatz. *Nonterminal Symbols:* Rechteckige Darstellung und *Terminal Symbols:* Rechteckige Darstellung mit abgerundeten Ecken

Der Einsatz von Syntax-Diagrammen erleichtert die Umsetzung von formatierten Informationen durch Computersysteme. Aus den einmal erarbeiteten Syntax-Diagrammen, können der Scanner und Parser direkt abgeleitet und implementiert werden.

3.3.4.5 Beispiel: Syntax Diagramm einer Luftfracht Meldung

Anhand zweier Meldungen aus dem IATA-Definition [IATA 1996], werden der Einsatz und die Nützlichkeit von Syntax-Diagrammen bei der Verarbeitung der Frachtinformationen durch Computer sichtbar. Aus einem Syntaxdiagramm kann direkt der Source Code eines Parsers abgeleitet werden, sofern dieser Parser nicht mit einem Parser Generator erstellt wird. Das Prinzip für die Codierung lautet; alle Nonterminal Symbols entsprechen einem Funktions- oder einem Methodenaufruf.

Die beiden Meldungen sind Status Update Message (FSU) und Status Answer Message (FSA):

- *FSU*: Status Update Message – Meldung über den Status einer zu transportieren Ware von der Fluggesellschaft an den Freight Forwarder geliefert.
- *FSA*: Status Answer Message – Statusmeldung als Antwort auf eine Statusanfrage durch den Freight Forwarder.

Die Syntax Diagramme sind nachfolgend in **Abbildung 3.11**, **Abbildung 3.12**, **Abbildung 3.13** und **Abbildung 3.14** dargestellt.

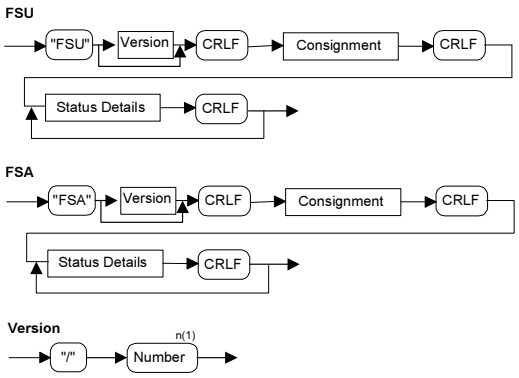

Abbildung 3.11 Oberste Ebene des Meldungsaufbaus

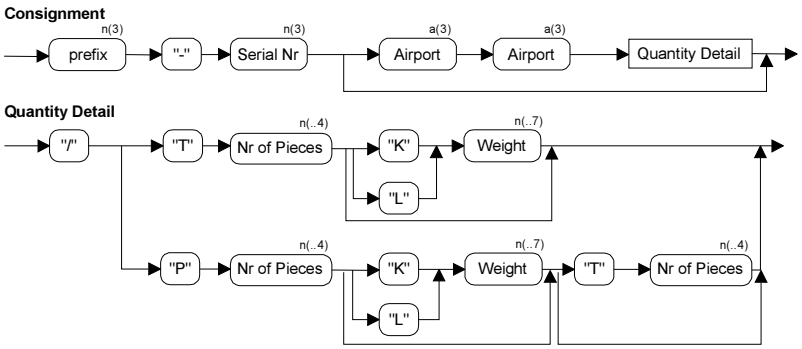

Abbildung 3.12 Consignment Nontermial

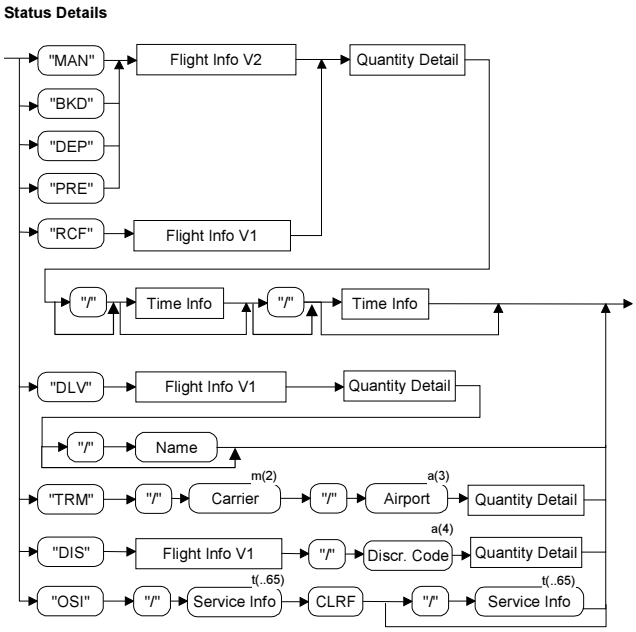

Abbildung 3.13 Status Details Nontermial

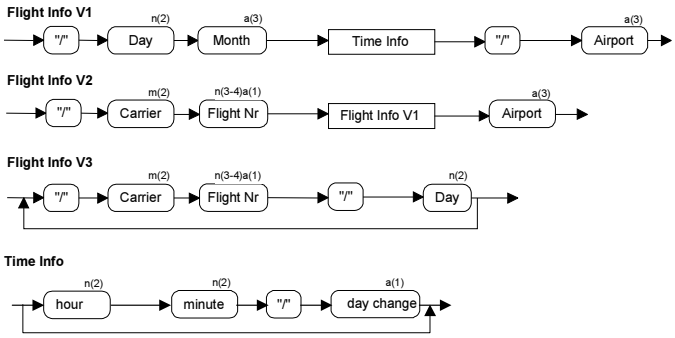

Abbildung 3.14 Flight Information und Time Information Nontermial

3.4 Scanner und Parser

Scanner und Parser sind Instrument zur Aufbereitung von Informationen, die in einer formalen Sprache formuliert worden sind. Sie führen die lexikalische und syntaktische Analyse oder Aufbereitung eines gegebenen Inputs durch, um die nachfolgend geordnete Abarbeitung zu garantieren.

3.4.1 Scanner

Der Scanner führt die lexikalische Analyse durch. Er kontrolliert den Input Stream auf zulässige Zeichen, entfernt überflüssige Zeichen und führt das Error-Handling durch. Der typische Aufbau eines Scanners ist in **Abbildung 3.15** dargestellt.

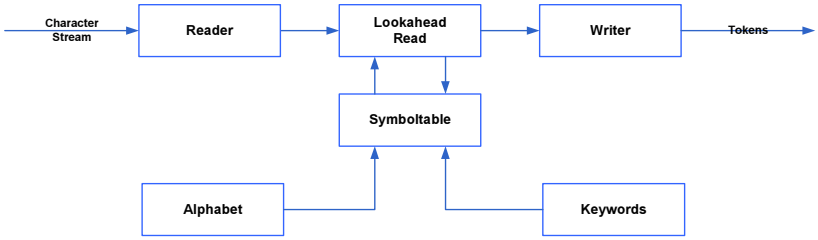

Abbildung 3.15 Struktur eines Scanners

- *Reader:* Einlesen des Character Streams.
- *Lookahead Read:* Prüfen und isolieren des nächsten Tokens.
- *Symboltable:* Tabelle mit allen Keywords, anderen zulässigen Zeichen oder Zeichenketten und Trennzeichen. Sie wird während dem Scanning aufgebaut.
- *Writer:* Schreiben der einzelnen Token.

Die Information, die als Input-Stream vorliegt, wird von einem Scanner entgegengenommen und Zeichen für Zeichen abgearbeitet.

Die Tätigkeiten eines Scanners:

- Bestimmt, ob ein bestimmtes Zeichen ein terminales Symbol ist oder ob es ein Lexem ist. Lexeme und Keywords werden in einer Symboltabelle gehalten.
- Entfernt syntaktisch überflüssige Zeichen wie Leerzeichen, Tabulatoren und Kommentare.
- Weist unzulässige Zeichen als Fehler zurück.
- Kontrolliert allenfalls die Ausrichtung der Lexeme.

3.4.2 Parser

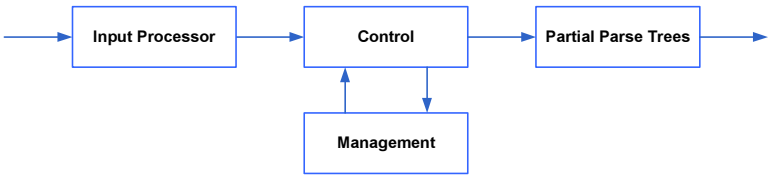

Abbildung 3.16 Generalisierte Struktur eines Parsers

Der Parser prüft die korrekte Syntax anhand der gegebenen Grammatik (**Abbildung 3.16**).

- *Management:* Ein Mechanismus zur Verwaltung (Speichern, Ersetzen, Löschen) eines Parse-Trees.
- *Control:* Ein Kontrollsystem zur Steuerung der Verwaltung durch den Baum.
- *Input Processor:* Ein Mechanismus, der die Input Tokens vom Scanner entgegennimmt.
- *Partial Parse Trees:* Die Bereits abgearbeiteten Teilbäume.

Die Tätigkeiten eines Parsers:

- Der Parser prüft die syntaktische Korrektheit des bereits in Tokens vorhandenen Input-Streams.
- Um die syntaktische Korrektheit zu testen, wird ein Parse-Tree (Zerlegungsbaum) aufgebaut.

3.4.3 Parse-Tree

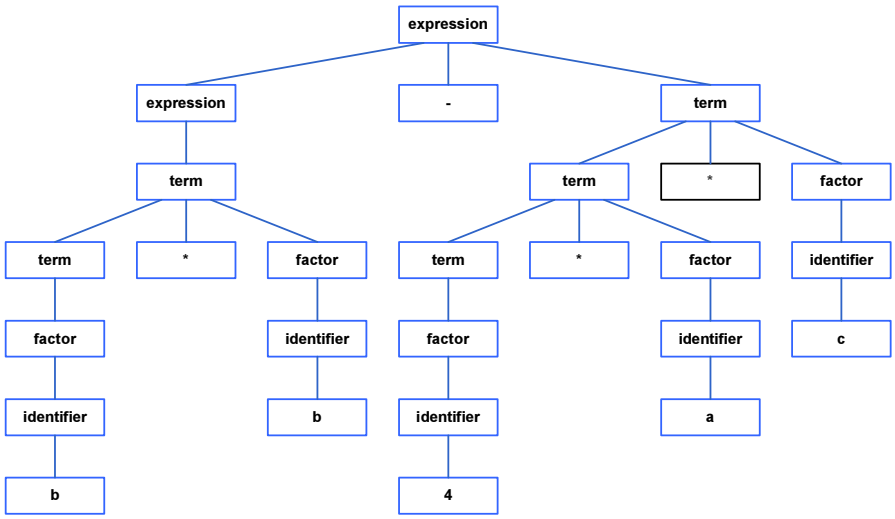

Abbildung 3.17 Parse-Tree des Ausdrucks: b*b-4*a*c

Ein Parse-Tree ist ein Baum mit folgenden Eigenschaften:

- Die Wurzel ist das Startsymbol S der Grammatik.
- Die Inneren Knoten sind Nonterminals.
- Die Blätter sind Terminals oder leer.

Die Bildung von Teilbäumen erfolgt nach den Produktionsregeln der Grammatik. Ein Parse-Tree kann auf zwei Arten erzeugt werden. Als Bottom-Up Parse-Tree oder als Top-Down Parse-Tree. Ein Beispiel eines Parse-Trees ist in **Abbildung 3.17** skizziert.

3.4.3.1 Bottom-Up Parse-Tree

Die Bootom-Up Zerlegung beginnt mit dem zu analysierenden Wort. Es werden jeweils die Teile des Worts ausgesucht, die auf der rechten Seite einer Produktionsregel der Grammatik stehen. Diese Teile werden durch die linke Seite einer grammatikalischen Regel ersetzt. Dies geschieht so lange, bis das Startsymbol übrig bleibt (Reduction).

3.4.3.2 Top-Down Parse-Tree

Die Top-Down Zerlegung beginnt mit dem Startsymbol. Das gegebene Wort wird jeweils von links nach rechts geste-stet, ob eine der Produktionsregeln den Beginn des Wortes erzeugen kann. Falls ja, wird diese Regel angewandt (Production) und mit der Betrachtung des restlichen Wortes Fortgefahren. Die Zuordnung der Produktionen einer Ebene erfolgt rekursiv bis zu den Blättern des Parse-Tree.

3.4.4 Suchstrategien für Parser

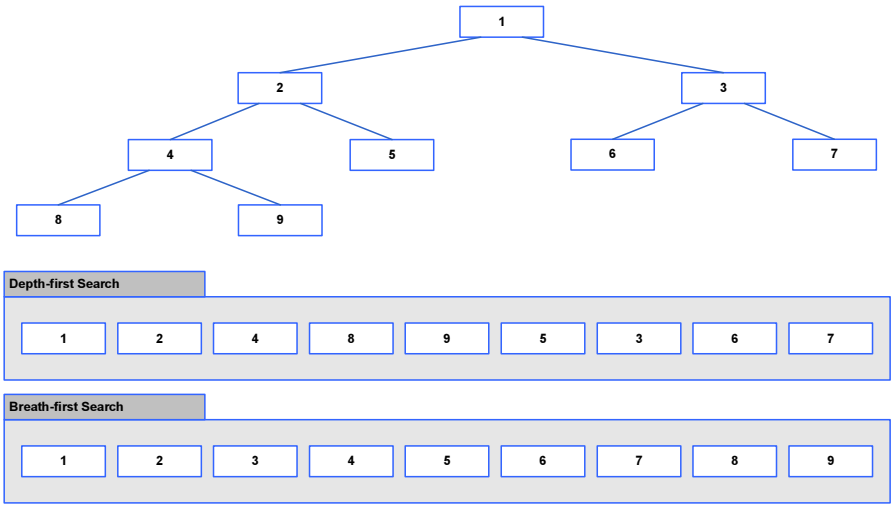

Abbildung 3.18 Suchmethoden

- *Depth-first Search:* Die Tiefensuche durchläuft einen Grafen vom Ausgangsknoten in die Tiefe, das heisst bis zu den Blättern des Graphen. Der Vorteil der Tiefensuche ist die Proportionalität des Speicherbedarfs zur Baumgrösse.
- *Breath-first Search:* Die Breitensuche durchläuft einen Grafen vom Ausgangsknoten in die Breite, das heisst es werden zunächst alle Knoten besucht, die in einem Schritt zu erreichen sind, dann erst diejenigen, die mit zwei Schritten zu erreichen sind. Der Vorteil der Breitensuche ist die Tatsache, dass die einfachste Lösung schneller gefunden wird.

Beide Suchmethoden (**Abbildung 3.18**) weisen exponentielles Zeitverhalten auf, falls sie nicht eingeschränkt werden.

3.4.5 Übersicht Parsing Techniken

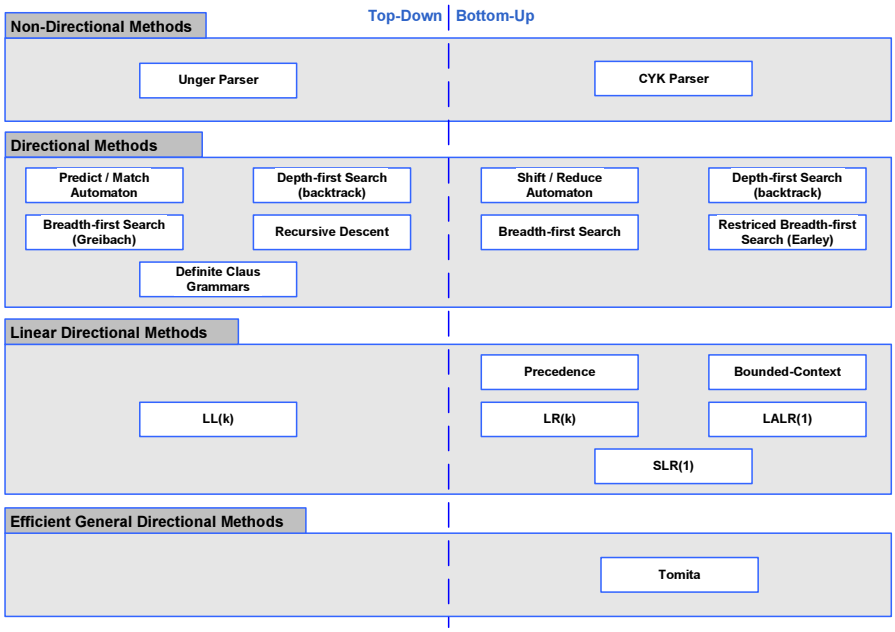

Abbildung 3.19 Parsing Techniken [Grune, Jacobs 1990]

Es existieren eine Vielzahl von Parsing Techniken. Die wichtigsten sind in **Abbildung 3.19** dargestellt.

- *Non-Directional Methods:* Der Zugriff auf den Input (Token-Stream) erfolgt in beliebiger Reihenfolge. Diese Methode setzt voraus, dass sich der gesamte Input im Speicher befindet.
- *Directional Methods:* Der Input wird schrittweise von links nach rechts durch den Parser abgearbeitet. In seltenen Fällen erfolgt die Verarbeitung von rechts nach links. Diese Methode hat den Vorteil, dass der Parser seine Arbeit beginnen kann, sobald der erste Token vorliegt.
- *Linear Directional Methods:* Lineare Parsing-Methoden begrenzen die Suche auf eine bestimmte Anzahl Tokens, die eingelesen werden, um die Suche in Parse-Tree zu begrenzen.
- *Efficient General Directional Methods:* Diese Methode generalisiert die Art und Weise, wie ein Parse-Tree durchlaufen werden kann. Das heisst, die Grammatik des Inputs ist nicht begrenzt auf deterministische Alphabete. Aus diesem Grunde eignet sich diese Parsing-Methode für natürliche Sprachen.

3.4.6 Eine Auswahl wichtiger Parsertypen

3.4.6.1 LL(k) Parser

Definition eines LL(K) Parsers:
Ein LL(k) ist ein Top-Down Parser, der eine eine LL(K) Grammatik verarbeitet.

Definition LL(k) Grammatik
Eine kontextfreie Grammatik G(∑, V, R, S) heißt LL(k)-Grammatik, wenn ein Ableitungsschritt eindeutig durch k Symbole der Eingabe (lookahead) bestimmt ist.

3.4.6.2 LR Parser

Ein LR Parser ist ein Bottom-Up Parser mit folgenden Eigenschaften:

LR-Parser können für praktisch alle Sprachen, für die kontextfreie Grammatiken existieren, konstruiert werden.

LR-Parser sind allgemeiner als viele andere Techniken, besitzen aber dennoch die gleiche Effizienz.

LR-Parser erkennen Syntaxfehler zum frühest möglichen Zeitpunkt.

Definition LR(k) Grammatik
Eine kontexfreie Grammatik G(∑, V, R, S) heißt LR(k)-Grammatik, wenn sie mit einem LR-Parser mit einem Lookahead von k Symbolen geparst werden kann.

3.4.6.3 LALR Parser

Die LALR-Parserkonstruktionstechnik ist die am häufigsten benutzte Technik zur Generierung von Syntaxanalysetabellen für LR-Parser. Zum einen werden von einem LALR-Parser die gängigsten Programmkonstrukte von Programmiersprachen erkannt, zum anderen ist die Anzahl der Zustände der Parsertabellen deutlich kleiner als die von Tabellen, die durch die kanonische Konstruktionsmethode erzeugt werden.

3.4.6.4 SR Parser

Ein SR-Parser (Shift/Reduce) besteht aus einem Eingabeband, einem Stack und einer Datenstruktur, die Informationen liefert, die zur Entscheidung der nächste Aktion in Abhängigkeit des aktuellen Zustands des Parsers während des Parsevorganges benötigt werden. Diese Entscheidung umfasst, ob ein weiteres Symbol eingelesen (shift) oder eine bestimmte Regel angewandt wird (reduce). Dabei können Informationen "unten" im Stack benutzt werden, um richtige Regel zur Reduktion anzuwenden, auch wenn das entsprechende Symbol nicht in die Reduktion einbezogen ist.

3.4.7 XML-Parser

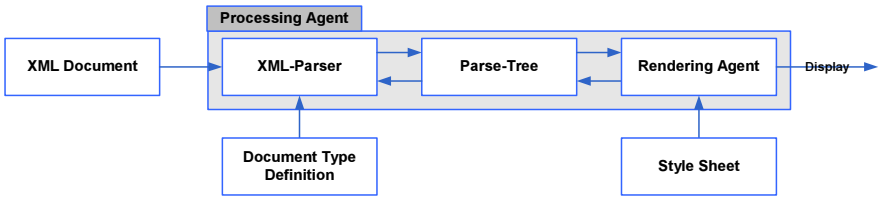

Abbildung 3.20 Verarbeitung eines XML-Dokuments [Röllinghoff 2000]

XML-Parser werden in zwei unterschiedliche Typen unterteilt:

- *Nichvalidierende Parser:* Diese Parser überprüfen lediglich, ob das Dokument wohlgeformt ist.
- *Validierende Parser:* Diese Parser prüfen zusätzlich, ob das XML-Dokument, welches als Input-Stream geparst werden soll, an die vordefinierte DTD (Document Type Definition) hält.

3.4.7.1 Definition: Ein wohlgeformtes XML-Dokument:

Ein Dokument gilt als wohlgeformt, wenn folgende Voraussetzungen erfüllt sind:

- Alle Attributwerte müssen in Anführungsstrichen stehen.
- Jedes Tag muss ein Ende-Tag besitzen (
</br> oder nur
).
- Elemente müssen sauber ineinander eingebettet sein (nicht <sprache> <deutsch> </sprache> </deutsch>).
- Der Name eines Attributs kommt innerhalb eines Elements nicht mehr als einmal vor.
- Ein Attributwert darf keinen Verweis auf ein externes Entity enthalten.
- Ein Attributwert darf kein "<" enthalten.

XML-Parser gibt es in zwei grundlegenden Ausprägungen:

- *SAX:* SAX (Simple API for XML) erlaubt das schrittweise Parsen von XML Dokumenten, ohne dass der gesamte Parse-Tree des Dokumentes im Speicher aufgebaut werden muss.
- *DOM:* DOM (Document Object Model)

3.4.8 SAX

SAX (Simple API for XML) ist auf Initiative der XML-DEV Mailingliste 1997 entstanden und definiert Schnittstellen für das Parsen von XML-Dokumenten [Megginson 1998]. Das SAX-API stellt einen Mechanismus zum Parsen von XML-Dokumenten zur Verfügung. Das Dokument wird Element für Element abgearbeitet und kann, da SAX ein aktives Parser-API ist, Events erzeugen, die auf ein konkretes XML Element abgestimmt sind. Diese Events können von einem Programm weiterverarbeitet werden, ohne dass ein ganzer Dokumentenbaum im Speicher aufgebaut werden muss.

SAX eignet sich dafür, schnell und effizient XML-Daten zu lesen. Sollen XML-Daten lediglich eingelesen werden, um sie dann in einer Anwendung zu verarbeiten, so ist SAX die ideale Lösung dafür. Die Umwandlung der Struktur eines XML-Dokumentes oder die Ausgabe des XML-Dokuments ist jedoch nicht die Stärke von SAX.

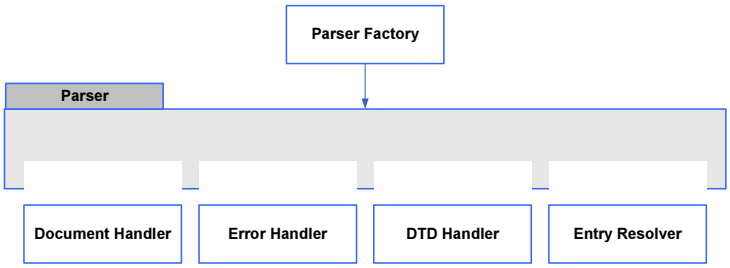

Abbildung 3.21 Prinzip von SAX

Die wichtigsten Handler von SAX sind der Document Handler zur Verarbeitung von Dokumenten, der Error Handler zur Fehlerverarbeitung, der DTD Handler zur gesteuerten Verarbeitung von SAX Formaten und der Entity Resolver zur Verarbeitung externer Referenzen (**Abbildung 3.21**).

- *Document Handler:* Dieser Handler ist das zentrale Instrument zur Verarbeitung. Er realisiert die Methoden *startDocument* und *endDocument*. Jeder einzelne XML Tag wird mit den beiden Methoden *startElement* und *endElement* abgearbeitet. Erkennt der Parser ein Tag, wird die Methode *startElement* mit dem Namen des Tags und sämtlichen Attributwerten als Parameter aufgerufen. Die Verarbeitung wird in dieser Methode gesteuert. Analog dazu wird die Methode *endElement* aufgerufen. Erkennt der Parser hingegen freien Text, der zwischen Tags steht, wird die Methode *Charakters* aufgerufen, findet er eine Processing Instruction die Methode *processingInstruction*. Ausserdem

gibt es noch die Methoden *setDocumentLocator* (Position im XML-Dokument) und *ignorableWhitespace* .

- *Error Handler:* Dieser Handler behandelt alle Fehler, die beim Parsen auftreten. Dafür gibt es die Methoden *warning*, *error* und *fatalError*, je nach Fehlerklassifizierung.
- *DTD Handler:* Dieser Handler wird aufgerufen, wenn der Parser in einer DTD auf ein „unparsed entity“ (Binärdaten) trifft. Diese Elemente können mittels speziellen Methoden behandelt werden.
- *Entity Resolver:* Der *EntityResolver* erlaubt die Auflösung externer Referenzen.

3.4.9 DOM

Das Document Object Model (DOM) ist eine vom World Wide Web Consortium definiertes API für XML- und HTML-Dokumente. XML bedeutet hier nicht nur Text-, sondern allgemeine Dateiformate, die auch beispielsweise Vektorgrafiken beinhalten können.

Das DOM definiert Schnittstellen und Methoden zum erzeugen, durchsuchen, zugreifen, ändern und löschen von Dokumentinhalten. Das Serialisieren der Dokumentstruktur (z.B. zum speichern in eine Datei) ist im DOM nicht definiert. Das DOM ist eine abstrakte Schnittstellenbeschreibung in OMG IDL (Object Management Group Interface Definition Language), die zur Anwendung in einer konkreten Sprache implementiert sein muss. Dokumente werden unabhängig von der internen Datenstruktur der jeweiligen Implementation nach außen objektorientiert repräsentiert. Alle Objekte des Dokuments sind in eine hierarchische Baumstruktur eingegliedert und werden im DOM Knoten genannt.

DOM garantiert strukturellen Isomorphismus, d.h. jedes XML-Dokument hat eine eindeutige Struktur, die in jeder DOM-Implementation genau gleich aussieht. DOM definiert Objekte, Schnittstellen und Semantiken (das Verhalten der Objekte beim Aufruf von Methoden), nicht aber interne Datenstrukturen oder Funktionen der Implementationen. Insbesondere kann DOM auf bestehende API’s aufgesetzt werden, um diese zu standardisieren [Wood et al. 1999].

DOM ist diejenige Dokumentstruktur, die von jedem modernen Web Browser verarbeitet werden kann. Das Modell selbst ist relativ komplex und in einer mehrere hundert Seiten umfassende Spezifikation beschrieben.

3.4.9.1 DOM Module

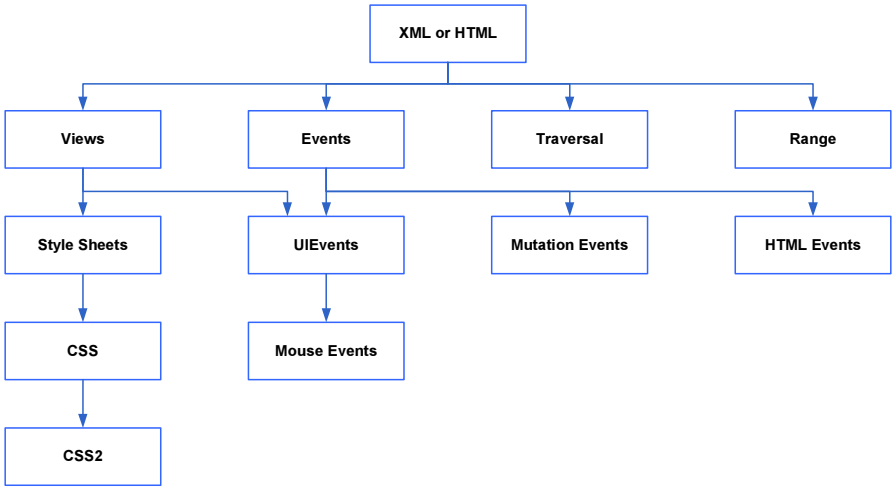

Abbildung 3.22 Die Document Object Model Module

Herzstück einer DOM Implementation sind die DOM Module, die als Übersicht in **Abbildung 3.22** dargestellt sind.

Eine DOM-Implementation muss in jedem Fall das Module XML (Core) realisieren.

- *Views:* Dieses Modul wird realisiert, wenn ein Dokument angezeigt werden soll.
- *Stylesheets:* Basis-Interface für verschiedene Formatierungs-Sprachen.
- *Events:* Allgemeine Schnittstelle für Ereignisse.
- *Traversal:* Dieses Module realisiert die Art und Weise, wie ein bestimmter Parse-Tree durchlaufen werden soll.
- *Range:* Dieses Modul dient zum Verarbeiten von Dokumentbereichen.

3.5 Fragen zum Kapitel

Nr	Frage
1	Wesshalb ist ein Web Browser eine Sprachverarbeitende Maschine?
2	Was ist SGML?
3	Kombinieren Sie das Modell des Language Processing User mit dem kybernetischen Modell des Menschen und erstellen Sie eine entsprechende Skizze.
4	Was ist die Theorie des kommunikativen Handelns?
5	Was beschreibt eine Cascated ATN Grammatik?
6	Erklären Sie den Unterschied zwischen Datalog und KBNL?
7	Was unterscheidet eine formale Sprache von einer natürlichen Sprache?
8	Wie ist die Browser Engine der Browser Reference Architecture aufgebaut?
9	Die Grammatik einer natürlichen Sprache unterscheidet sich vor allem in einem Punkt von derjenigen einer formalen Sprache. In welchem?
10	Beschreiben Sie die Tätigkeiten eines Scanners.

3.6 Übung zum Kapitel

Nachfolgend finden Sie den Artikel „Beyond Programming Languages“ von Terry Winograd, einem der führenden KI Pioniere.

Lesen Sie den Artikel durch und beantworten Sie folgende Fragen:

1. Was versteht Winograd unter den drei grundlegenden Veränderungen in der Programmierung?

2. Was ist sein Lösungsvorschlag?

3. Wie ist Ihre persönliche Meinung dazu und welchen Einfluss glauben Sie, hat der Artikel auf die folgende Entwicklung der Programmiersprachen eingeübt?