Systematics - Ein XML-basiertes Internet-Datenbanksystem für klassifikationsgestützte Sprachbeschreibungen*

Matthias Nickles
Institut für Informatik, Technische Universität München
D-80290 München, nickles@cs.tum.edu

Zusammenfassung

Der Bericht schildert die wesentlichen informatischen Merkmale des XML-basierten Client-Server-Datenbanksystems *Systematics*, das für das linguistische DFG-Projekt *Allgemein-Vergleichende Grammatik* (AVG) entwickelt wurde, um den Aufbau einer universellen Datenbank für die Beschreibungen natürlicher Sprachen in einem einheitlichen Format zu ermöglichen. Im Rahmen der Entwicklung wurde als Beschreibungshilfe für die interaktive Erzeugung und Abfrage der semistrukturierten Datenbank-Einträge eine in Form eines *XML-Schemas* textuell repräsentierbare graphische Datenstruktur (*Be-Have-Baum*) definiert, die im Bericht ebenfalls eingeführt wird. Die Software wurde für die Beschreibung natürlicher Sprachen entwickelt, ist aber im Prinzip auch für andere Klassifikations-Anwendungen geeignet.

Schlüsselbegriffe: XML-Datenbanken, XML-Schemata, graphische Abfragesprachen, Klassifikationssysteme

1 Einführung

Das vorliegende Papier beschreibt das prototypische Internet-Datenbanksystem *Systematics*, das vom Autor für das von Prof. Dietmar Zaefferer geleitete und am Institut für Theoretische Linguistik der Ludwig-Maximilians-Universität München angesiedelte DFG-Projekt *Allgemein-Vergleichende Grammatik 2.0* (kurz *AVG-Projekt*, englisch: *Cross-Reference Grammar* (*CRG*)) [11] entwickelt wurde[1]. Das AVG-Projekt hat sich zum Ziel gesetzt, eine universelle Datenbank für die einheitliche Beschreibung von natürlichen Sprachen jeder Art zu schaffen. Entsprechend dem Projektziel lautete die informatische Aufgabenstellung, ein Datenbanksystem zu entwerfen und zu implementieren, das die Erstellung, Speicherung und Abfrage von Sprachbeschreibungen und die Erstellung eines Schemas für derartige Daten erlaubt[2]. Systematics ist allerdings nicht auf ei-

*Paper-Revision 2

[1] Der Name 'Systematics' bezieht sich in diesem Dokument immer auf die vom Autor entwickelte Version 1.0 der Software, nicht auf spätere Versionen.

[2] Vorliegender Text befasst sich ausschließlich mit informatischen Aspekten des Projekts. Informationen zur linguistischen Anwendung von Systematics sind unter [11] zu finden.

ne linguistische Anwendung beschränkt, sondern kann - zumindest im Prinzip - auch für andere Klassifikationsaufgaben verwendet werden. Die Benutzergruppe der *Schema-Ersteller* (vorrangig AVG-Mitarbeiter) konstruiert das Beschreibungs-Schema in Gestalt eines hierarchischen Klassifikationssystems und stellt es der zweiten, heterogenen und weltweit verteilten Benutzergruppe der *Sprachbeschreiber* und *Abfrager* (auch *Datenbank-Consultants* genannt) zur Verfügung. Die Sprachbeschreiber fügen der Datenbank Schema-konforme, aber auch in gewissen Grenzen vom Schema abweichende Beschreibungen von Phänomen natürlicher Sprachen hinzu. Solche Beschreibungen bestehen im wesentlichen aus der Benennung der zu beschreibenden Sprache, der Angabe einer Kategorie, die als Instanz des Beschreibungs-Schemas das jeweilige Sprachphänomen möglichst genau eingrenzt und seine Existenz in der betreffenden Sprache ausdrückt, und einem illustrierenden Beispiel für das beschriebene Phänomen. In Systematics fallen alle drei Teilaspekte unter dem Begriff der Kategorienauswahl und technisch unter dem Begriff der baumförmigen *Selektion* zusammen. Selektionen stellen die Einträge der Datenbank dar, die von der Gruppe der Abfrager ausgelesen wird, aber auch die Abfragen selbst. Die Abfrager fragen Datenbankinhalte mittels *unvollständiger Selektionen* (Query-Selektionen), die als Muster mit den gesuchten Datenbank-Einträgen matchen, ab[3]. Die Eintragung und Abfrage von Selektionen ist technisch so ähnlich, dass die Sprachbeschreiber und Abfrager weitgehend einheitlich als eine Hauptbenutzergruppe behandelt werden können. Dateneintrag und -abfrage erfolgt mit *demselben* Tool und erfordert in beiden Fällen genau dieselbe Art von Interaktion des Anwenders mit der Software, während die Schemaerstellung ein grundlegend anderer Vorgang ist und mit einem anderen Programm - dem Schemaeditor - erfolgt. Das Schema dient als Hilfestellung für die manuelle und interaktive Formulierung von Selektionen, d.h. für die clientseitige Erstellung und Abfrage von Phänomenbeschreibungen, wird aber auf der Seite der Datenbank (dem serverseitigen "Backend" der Software) technisch nicht benötigt, da die Daten in *semistrukturierter* Form als XML-Dokument vorliegen.
Die besonderen informatischen Herausforderungen bei der Realisierung geeigneter Tools für die geschilderten Aufgaben lagen vor allem darin, dass für das Beschreibungs-Schema ein graphisches Nicht-Standardformat (eine Variante sogenannter *Und-Oder-Bäume*) verwendet werden sollte, das sich in der vorangehenden Phase des AVG-Projekts bereits für die Darstellung linguistischer Kategorien zur universellen Sprachbeschreibung bewährt hatte [1, 2, 3], und die interaktive, graphische Eingabe und Abfrage der Daten weltweit via Internet ohne besondere technische Installationen seitens der Sprachbeschreiber oder Abfrager möglich sein sollte. Außerdem sollten die zu entwickelnden Tools von Linguisten bedienbar sein, d.h. von einer Anwendergruppe (den Schema-Erstellern auf der einen und den Sprachbeschreibern bzw. Abfragern auf der anderen Seite), die zum überwiegenden Teil aus informatischen Laien besteht, was es erforderlich machte, trotz der Komplexität der zu bearbeitenden Daten

[3] Aus Anwendersicht ist dies ungefähr vergleichbar mit der *Query-by-Example*-Technik relationaler Datenbanksysteme, wobei unserem graphischen Beschreibungsschema dort die *Formulare* zur Dateneingabe und -darstellung entsprechen.

eine einfache Bedienbarkeit sicherzustellen. Aufgrund dieser besonderen informatischen Anforderungen, die den bis dato einmaligen Charakter des AVG-Projekts widerspiegeln, schied die direkte Verwendung gängiger relationaler oder objektrelationaler Standard-DBMS und ihrer Zugriffstools und -sprachen (z.B. SQL) aus. Gleichwohl sollte wegen der zu erwartenden großen Menge anfallender Daten eine spätere Einbindung leistungsfähiger Standard-DBMS als Backend-Datenspeicher möglich sein, wobei die Verwendung objektorientierter oder objektrelationaler DBMS wegen der hierarchischen (baumförmigen) Struktur der Sprachbeschreibungen naheliegt.
Bei der Entwicklung von Systematics konnte auf umfangreiche Vorarbeiten zurückgegriffen werden, die vom AVG-Projekt in enger Zusammenarbeit mit dem Lehrstuhl Prof. Bry des Instituts für Informatik der Ludwig-Maximilians-Universität geleistet wurden. Dazu zählt insbesondere die theoretische Spezifikation von elementaren *Und-Oder-Bäumen* für die Schema-Repräsentation zur Darstellung von linguistischen Kategorien [1, 2], die Implementierung eines prototypischen Prolog-basierten Datenbank-Backends für Selektionen über Und-Oder-Bäumen [2] und die vom Autor dieses Papiers durchgeführte Entwicklung des prototypischen Werkzeugs *Categories* zur graphischen Darstellung, Bearbeitung und Verwendung einer für das AVG-Projekt erweiterten Variante solcher Bäume bei der Phänomenbeschreibung [3].
Obwohl sich Und-Oder-Bäume als grundsätzlich geeignet erwiesen haben und die damals entwickelten Software-Prototypen trotz mangelnder Eignung für größere Datenmengen und einigen funktionalen Einschränkungen zumindest einen "proof of concept" darstellen, zeigten sich einige gravierende prinzipbedingte Nachteile, die die notwendige Weiterentwicklung und ein Zusammenwirken mit fremden Tools sehr stark erschwert hätten: Den Phänomenbeschreibungen mittels Und-Oder-Bäumen liegt ein proprietäres Datenmodell zugrunde, das einige unnötige Beschränkungen aufweist, und die damit erstellten Beschreibungen liegen in einem entsprechend proprietären Format vor (eine detaillierte Beschreibung von Und-Oder-Bäumen findet sich in der folgenden Sektion). Dementsprechend ist auch das Tandem aus Prolog-Datenbanksystem und Categories inkompatibel zu gängigen Datenbeschreibungsstandards wie z.B. XML. Dazu kommt der Umstand, dass Categories in einem einzigen Java-Applet sowohl die Schema-Erstellung als auch die Sprachbeschreibung und Beschreibungs-Abfrage kombiniert. Durch die fehlende Modularität wurde jede der beiden Hauptbenutzergruppen durch einen für den jeweiligen Einsatzzweck unnötigen Overhead (der sich aus Sicht der Sprachbeschreiber und Abfrager insbesondere durch hohe Applet-Downloadzeiten äußerte) belastet und der Ausbau der Komponente für die Schema-Ersteller unmöglich gemacht. Diesen Nachteilen wurde mit Systematics dadurch begegnet, dass zum einen zur Schema-Repräsentation die Und-Oder-Bäume durch sogenannte $Be-Have-B\ddot{a}ume$ (in einigen früheren Projekt-Veröffentlichungen auch 'erweiterte Und-Oder-Bäume' genannt) ersetzt wurden, die einerseits kompatibel zu Und-Oder-Bäumen sind, andererseits aber Eigenschaften gängiger Datenbeschreibungs-Formate (objektorientierte Klassensysteme und kontextfreie Grammatiken) in sich vereinen und sich auf naheliegende Weise durch *XML-Schemata* [4, 5] darstellen lassen, was die Repräsentation von Selektionen in Gestalt von XML-Dokumenten er-

laubt und Systematics prinzipiell für eine Zusammenarbeit mit jeder XML-fähigen Software öffnet (z.B. XML-fähigen Datenbanksystemen wie Oracle 8i/9i). Da Be-Have-Bäume auf (eingeschränkte) XML-Schemata abgebildet werden, können Phänomenbeschreibungen im Prinzip *beliebige* XML-Dokumente darstellen. Dies ist wohl die wesentliche technische Besonderheit gegenüber anderen XML-basierten linguistischen DBMS, die zumeist spezielle XML-Teilsprachen mit eigenen Namensräumen verwenden. Als weitere Verbesserung wurden zwei getrennte Tools für die beiden Hauptbenutzergruppen entwickelt, die eine deutlich größere Funktionalität im Vergleich mit Categories bei einfachster Installation und Bedienung besitzen. Um nicht nur die Sprachbeschreibung und Abfrage durch AVG-externe Linguisten zu erlauben, sondern auch die Schema-Bearbeitung, wurde auch der sogenannte *Schemaeditor* als verteilte, Internetfähige Anwendung gestaltet. Viele wesentliche Beschränkungen von Categories wurden damit behoben, allerdings haben die Systematics-Tools in der Version 1.0 nach wie vor einen mehr oder weniger prototypischen Charakter. Während der Schemaeditor für die Schema-Erstellung und -Bearbeitung seit längerem erfolgreich in der Praxis eingesetzt wird und nur Detailverbesserungen bedarf, handelt es sich bei den Komponenten für die Sprachbeschreiber und Abfrager (das sogenannte *Webinterface* und die XML-QL [6] -basierte Beschreibungs-Datenbank) um ausgesprochene Prototypen, die für große Datenmengen nicht geeignet sind. Daher konzentriert sich die gegenwärtige Weiterentwicklung von Systematics (auf die hier nicht eingegangen werden kann) im wesentlichen auf die Einbindung eines objekt-relationalen DBMS mit dem Ziel der Verbesserung der Performanz bei der Verarbeitung großer Datenmengen (vgl. Sektion 4).

Das weitere Papier gliedert sich wie folgt: Sektion zwei beschreibt die Formate von Schema und Daten (Selektionen) und zeigt, wie diese Datenstrukturen in XML textuell repräsentiert werden können. Be-Have-Bäume als Beschreibungsschemata werden dabei aus dem Datenmodell der Vorgängersoftware Categories entwickelt. In Sektion drei wird die Architektur der verteilten, aus verschiedenen Tools für die einzelnen Benutzergruppen bestehenden Anwendung Systematics im Überblick vorgestellt. Mit einer kritischen Betrachtung von Systematics und einem Ausblick auf notwendige Verbesserungen in Sektion 4 schließt das Papier.

2 Schema und Selektionen

Das Beschreibungs-Schema (das, wie in 2.2 gezeigt wird, nicht unbedingt auch als Datenbank-Schema angesehen werden kann) hat in unserem Kontext die Aufgabe, die manuelle, interaktive Erstellung von Schema-konformen Selektionen zu unterstützen. Als tauglich für diese Aufgabe haben sich bereits vor der Entwicklung von Systematics Und-Oder-Bäume erwiesen, die seit längerem in der Künstlichen Intelligenz als Entscheidungsbäume verwendet werden. Dieser Baumtyp erlaubt nicht nur die hierarchische Anordnung von linguistischen Kategorien als Knoten und die Generierung von Selektionen in Gestalt von Knotenmengen durch einen manuellen Top-Down-Durchlauf durch den Baum durch den Anwender, sondern dadurch dass es zwei wesentliche Knotentypen

(*Und*- und *Oder*-Knoten) gibt, auch eine flexiblere und weniger redundante Repräsentation im Vergleich zu einfachen hierarchische Kategoriensystemen, die nur einen einzigen Knotentyp kennen (etwa entsprechend den Oder-Knoten) [1, 2]. Für die Verwendung mit dem Systematics-Vorgänger Categories wurden die elementaren Und-Oder-Bäume, wie sie die KI kennt, um zwei Knotentypen (*Textblätter* und *nicht-exklusive Oder-Knoten*) erweitert [3] und für Systematics zu *Be-Have-Bäumen* weiterentwickelt, um die Ausdrucks- und Modellierungsmöglichkeiten von *XML-Schemata* zur Verfügung zu stellen (die einige wesentliche Merkmale objektorientierter Klassensysteme und Grammatiken formaler Sprachen[4] umfassen) und XML-Schemata zur textuellen Repräsentation der Bäume und Schema-Instanzen (im Prinzip beliebige XML-Dokumente) zur textuellen Repräsentation der Selektionen (Sprachbeschreibungen und Queries) verwenden zu können.

2.1 Kategoriensysteme und Und-Oder-Bäume

Im folgenden werden Und-Oder-Bäume, die eine Teilmenge der Be-Have-Bäume darstellen, über den Begriff des *Kategoriensystems* motiviert und informell charakterisiert, bevor in 2.2 der Baumtyp zu Be-Have-Bäumen erweitert und exakt definiert wird. Die grundlegenden Charakteristika und Verfahren bleiben dabei gültig, so dass die folgenden Abschnitte auch als Einführung in die Thematik der Be-Have-Bäume gelesen werden kann.
Eine *Kategorie* lässt sich als eine Klasse von Merkmalen auffassen, die ein bestimmtes *Objekt* (in unserem Fall ein linguistisches Phänomen) eingrenzen, ein Kategoriensystem als Menge zusammengehöriger Kategorien. Um eine Beschreibung eines Objekts durchführen zu können, wird ein passendes Kategoriensystem vorgegeben. Die Aufgabe des Beschreibers ist es danach, aus diesem Kategoriensystem eine geeignete, das Objekt möglichst eng eingrenzende Kategorie auszuwählen. Diese Kategorie dient dann als formale Repräsentation der charakterisierten Sache und kann so in einer Datenbank abgelegt und maschinell ausgewertet werden. Kategoriensysteme sollten dazu möglichst wenig redundant sein und leicht zu warten (d.h. Änderungen und Hinzufügungen sollten mit möglichst geringem Aufwand verbunden sein), sowie vor allem den interaktiven Beschreibungsprozess, d.h. die Selektion einer geeigneten Kategorie durch den Anwender, unterstützen.
Um interaktive Beschreibungen zu ermöglichen, stellen wir Kategoriensysteme als Und-Oder-Bäume dar, bei denen es sich um Entscheidungsbäume handelt. Das besondere Merkmal eines solchen Baums ist, dass alle inneren Knoten entweder *Und-Knoten* oder *(exklusive) Oder-Knoten* sind (tatsächlich implementiert Categories noch weitere Knotentypen, die aber die Ausdrucksstärke der Bäume nicht erhöhen, sondern lediglich praktische Abkürzungen für bestimmte Formationen anderer Knoten sind). Ein solcher Baum repräsentiert das Kategoriensystem: *Aspekte* und *alternative Ausprägungen* ("Alternativen") der zu beschreibenden Objekte respektive ihrer Klassen werden durch die Kanten bzw.

[4] Der Begriff 'Grammatik' bezieht sich hier nur auf formale Grammatiken zur "technischen" Modellierung von Selektionen, nicht auf die modellierte "allgemein-vergleichende Grammatik".

Knoten dargestellt[5]. Aspekte ordnet man als Kinder von Und-Knoten an, Alternativen dagegen als Kinder von Oder-Knoten. Alternativen und Aspekte, die nur relevant sind, wenn in den Kategorien, in denen sie vorkommen können, auch bestimmte andere Merkmale enthalten sind, werden im selben Teilbaum als Nachkommen dieser Merkmale angeordnet.

Abbildung 1 zeigt einen Oder-Baum zur Modellierung eines Kategoriensystems zur Beschreibung von Fahrzeugen. Jedes Fahrzeug besitzt die Aspekte 'Fahrzeugart' und 'Motor', die beide spezifiziert werden müssen. Die Fahrzeugart ist entweder als 'Motorrad', 'PKW' oder 'LKW' ausgeprägt, der Motor entweder vom Typ 'Hubkolben' oder vom Typ 'Wankel'. Ein Hubkolbenmotor wird durch seine Aspekte 'Zündung' und 'Anzahl Zylinder' festgelegt, die Zündung wiederum erfolgt nach Art eines 'Otto'- oder 'Diesel'- Hubkolbenmotors (d.h. sie wird durch Zündkerzen oder Verdichtung ausgelöst). Ein 'Wankel'-Motor wird hier dagegen nur durch seine 'Anzahl Scheiben' näher beschrieben.

Um mit einem Und-Oder-Baum ein Objekt zu beschreiben, müssen passende Knoten ausgewählt werden, angefangen bei der Wurzel und fortschreitend in Richtung der Blätter des Baums: Wie man gesehen hat, verbinden Oder-Knoten alternative Ausprägungen der zu beschreibenden Gegenstände. Bereits ausgewählte Oder-Knoten repräsentieren somit Entscheidungs-Orte, an denen jeweils eines der Kinder des Knotens auszuwählen ist. Dagegen müssen bei einem bereits ausgewählten Und-Knoten, der ja gleichzeitig relevante Aspekte verbindet, auch jeweils alle Kinder ausgewählt werden. Eine Entscheidung muss hier also nicht getroffen werden. Somit kann der gesamte Beschreibungsvorgang als eine Folge von getroffenen Entscheidungen bei Oder-Knoten verstanden werden. Der Beschreiber folgt dabei dem hierarchischen Aufbau des Baums (beginnend beim Wurzelknoten), weil Entscheidungen ggf. erst dann relevant werden, wenn andere Entscheidungen, die im selben Teilbaum näher bei der Wurzel liegen, bereits getroffen wurden. Die ausgewählten Knoten und ihre verbindenden Kanten werden vom Selektionstool visuell hervorgehoben (natürlich können irrtümlich selektierte Knoten auch wieder de-selektiert werden). Um auf diese Weise anhand des Beispielbaums 'Fahrzeuge' ein konkretes Fahrzeug zu charakterisieren (d.h. eine geeignete Kategorie festzulegen, der das Fahrzeug zugeordnet werden kann), folgt man bei der Auswahl von Knoten der hierarchischen Aufeinanderfolge der möglichen Entscheidungen von der Wurzel zu den Blättern: Bei Und-Knoten sind alle ausgehenden Kanten weiterzuverfolgen; 'Fahrzeugart' und 'Motor' sind also Aspekte, die beide beschrieben werden müssen. 'Hubkolben' und 'Wankel' sind dagegen nicht miteinander vereinbar, ebensowenig wie 'Motorrad' und 'PKW'. Hier ist jeweils eine Entscheidungen über das weitere Vorgehen notwendig. Die Beschreibung der Eigenschaften des Aspektes 'Zündung' (also 'Otto' oder 'Diesel' usw.) ist nur relevant, wenn als Motor-Typ bereits 'Hubkolben' gewählt wurde. Dagegen kann die Entscheidung

[5]Die Begriffe "Aspekte", "Alternativen" und "Ausprägungen" werden hier nur informell eingeführt. In 2.2 erhalten sie eine formale Bedeutung im Zusammenhang mit Be-Have-Bäumen.

für eine Fahrzeugart (PKW, Motorrad oder LKW) unabhängig von der Wahl des Motortyps getroffen werden[6].
Formal ist ein derartiger Beschreibungsprozess die Zusammenstellung einer geeigneten Teilmenge von Knoten des Und-Oder-Baums unter der Berücksichtigung der bereits angedeuteten und im folgenden eindeutig festlegenden Regeln. Ein solcher Teilbaum heißt *Selektion* und repräsentiert je mindestens eine Kategorie aus dem Kategoriensystem. Selektionen sind durch folgende Eigenschaften gekennzeichnet, die im wesentlichen auch für Selektionen über Be-Have-Bäumen zutreffen (vgl. 2.2.1):

- der Wurzelknoten ist enthalten
- ist ein Und-Knoten enthalten, so auch alle seine Kinder
- ist ein (exklusiver) Oder-Knoten enthalten, so ist maximal eines seiner Kinder enthalten[7]
- ist ein Knoten nicht enthalten, so auch keines seiner Kinder

In der Praxis erlaubt das Selektionstool (für Und-Oder-Bäume Categories, das "Webinterface" im Fall von Systematics) das beliebige Auswählen von Knoten durch Mausklick und stellt daraufhin eine Selektion her, die den obigen Regeln entspricht, den Selektionszustand des ausgewählten Knoten (wenn gemäß den Selektionsregeln möglich) umkehrt und gleichzeitig so ähnlich wie möglich zur vorherigen Selektion ist (diese also möglichst minimal erweitert bzw. reduziert).
Falls zu jedem Oder-Knoten in der Selektion genau ein Kind enthalten ist, spricht man von einer *vollständigen* Selektion. Ist dagegen mindestens zu einem selektierten Oder-Knoten kein Kind in der Selektion enthalten, so nennt man die Selektion *partiell* oder *unvollständig*.
Ist kein Kind eines Oder-Knotens in der Selektion enthalten, so heißt die Selektion *minimal*. Man erkennt, dass sich zwei Selektionen zum selben Baum überhaupt nur durch das Vorhandensein von Oder-Knoten unterscheiden können. Eine vollständige Selektion repräsentiert genau eine Kategorie, eine partielle Selektion eine Menge von Kategorien oder auch eine "unschärfere" Kategorie (beide Sichtweisen sind legitim). Je mehr Knoten eine Selektion enthält, in der eine andere enthalten ist, desto *verfeinerter* (spezieller) ist sie gegenüber der enthaltenen Selektion, wodurch eine Teilordnung der Selektionen definiert ist (vgl. 2.2).
Partielle Selektionen sind nützlich, um mehrere Kategorien mittels einer einzigen Selektion "abgekürzt" darzustellen und (dies ist die sehr stark überwiegende Verwendung) um als Query an die Selektionen-Datenbank die Ergänzungsmöglichkeiten der partiellen Selektion hin zu mehreren vollständigen Selektionen zu

[6] Um Abhängigkeiten modellieren zu können, die sich nicht aus der Baumstruktur selbst ergeben, wie z.B. in unserem Beispiel die Unmöglichkeit eines Motorrades mit Dieselmotor, kennt Categories *logische Constraints*, auf die an dieser Stelle aber nicht eingegangen werden kann. Näheres dazu ist in [3] zu finden.

[7] Die Exklusivität betrifft nur die Selektierbarkeit dieser Knoten. Eine Kategorie kann eine Ausprägung mehrerer Ober-Kategorien gleichzeitig sein (vergleichbar mit der objektorientierten $Mehrfachvererbung$).

erfragen. Dagegen werden in die Selektionen-Datenbank der Sprachbeschreibungen üblicherweise nur vollständige Selektionen eingetragen (Systematics unterscheidet allerdings beide Selektionstypen technisch nicht grundsätzlich, vgl. 2.2). Selektiert man als Abfrager z.B. nur die Knoten "Fahrzeugart" und "PKW", so erhält man mit dieser partiellen Selektion als Query genau die Selektionen aus der Datenbank, die PKWs beschreiben. Der Motortyp wurde durch die Query nicht spezifiziert, so dass jeder gespeicherte "Motor" zurückgeliefert wird, der zu einem PKW gehört (solch eine mögliche Ergebnisselektion zeigt die rechte Seite der Abbildung 1)[8]

Neben den beiden genannten Knotentypen kennen die AVG-Tools noch die Knotentypen *Textblatt*[9], *Tabelle* (nur Systematics) und *nicht-exklusiver Oder-Knoten* (deren Kinder wir *Optionen* nennen). Obwohl sie für den linguistischen Anwender eine große Rolle spielen (mittels Textblättern und Tabellen werden zu selektierten Sprach-Phänomenen Beispiele (natürlichsprachliche Sätze) angegeben), gehen wir im Rahmen der Beschreibung von Und-Oder-Bäumen nicht näher auf sie ein, da sie sich theoretisch-informatisch als Abkürzungen anderer Knotentypen fassen lassen: Ein Textblatt als Oder-Knoten mit allen möglichen Zeichenketten bis zu einer bestimmten Länge als Kinder und eine Tabelle als Und-Knoten mit Textblättern als Kinder (Tabellenzellen). Nicht-exklusive Oder-Knoten erlauben wie (exklusive) Oder-Knoten die Auswahl von Kindern, diese können aber im Unterschied zu normalen Oder-Knoten unabhängig voneinander selektiert werden, d.h. auch zwei und mehr Kinder eines nicht-exklusiven Oder-Knotens können gleichzeitig selektiert sein. Ein solcher Knoten kann durch einen Und-Knoten ersetzt werden, der für jedes Kind des nicht-exklusiven Oder-Knotens genau einen (exklusiven) Oder-Knoten hat, dessen einziges Kind wiederum das jeweilige Kind des nicht-exklusiven Oder-Knotens ist.
Für den Linguisten dienen Selektionen zur Auswahl von Phänomenen natürlicher Sprachen. Allerdings enthält in Categories (und in analoger Weise in Systematics) der Und-Oder-Bäum des AVG-Projekts den "Phänomen"-Teilbaum nicht als Wurzel, sondern als ein Kind eines Und-Knotens, der die eigentliche Wurzel des Baums darstellt. Weitere wesentliche Kinder der Wurzel sind "Language" (ein Textblatt zur Angabe der Sprache, in der das selektierte Phänomen vorkommt), "Example" (urspünglich ein Textblatt, später eine sogenannte Partiturtabelle, in das bzw. die man ein Beispiel für das Phänomen eintragen kann), und "Author" (ein Textblatt zum Eintrag des Namens des jeweiligen Sprachbeschreibers). Lässt man als Abfrager eines dieser Textblätter leer, können Queries der Art "In welchen Sprachen tritt ein bestimmtes Phänomen auf?" oder "Nenne einen Beispielsatz für ein bestimmtes Phänomen!" formuliert werden.

2.2 Klassensysteme und Be-Have-Bäume

Im folgenden wollen wir den eher informellen Begriff 'Kategoriensystem' durch *Klassensystem* ersetzen, und daraus sogenannte *Be-Have-Bäume* entwickeln,

[8] Beispiele für linguistische Selektionen finden sich unter [18].
[9] Blätter können aber auch Und-/Oder-Knoten ohne Kinder sein.

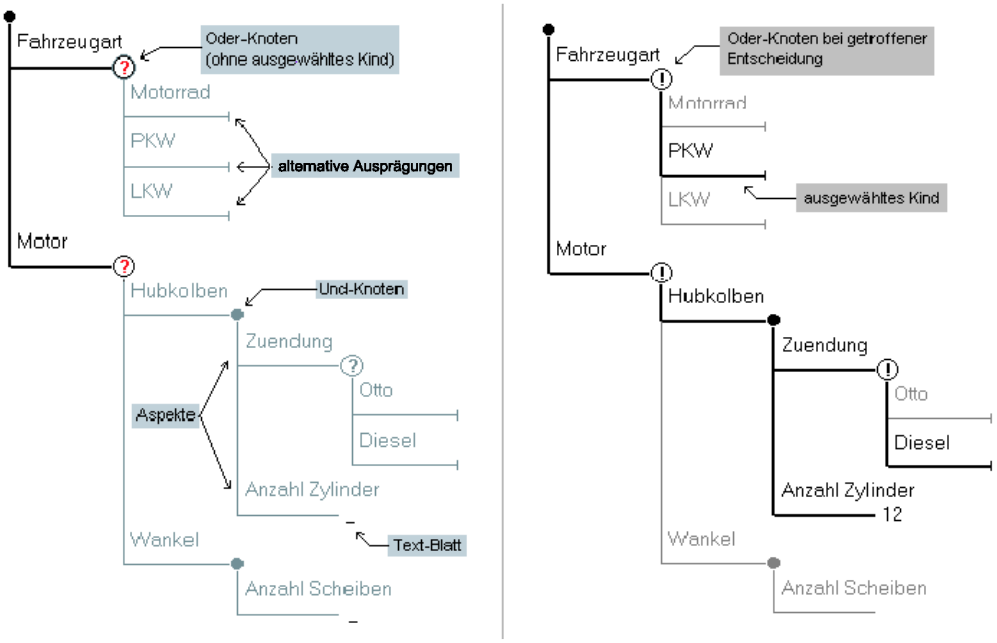

Abbildung 1: *Und-Oder-Baum 'Fahrzeug' (links ohne, rechts mit Selektion)*

welche die Und-Oder-Bäume ablösen. Das den Be-Have-Bäumen zugrundeliegende Datenmodell entlehnt zwar, wie man sehen wird, auch einige Aspekte kontextfreier Grammatiken, die von *XML-Dokumententypdefinitionen* (*DTDs*) stammen und nicht direkt objektorientiert modellierbar sind, die für den Anwender intuitiv wohl verständlichste Interpretation eines Be-Have-Baum ist aber die objektorientierte, so dass diese Sichtweise hier am Anfang stehen soll. Die Vergleichbarkeit mit den Klassensystemen objektorientierter Programme oder Datenbanken ist aber nur eingeschränkt möglich, vor allem da Be-Have-Bäume keine *Methoden* beschreiben.

Ein Klassensystem (auch *Schema* genannt) kann vereinfacht als Tupel (t, v, a) einer nicht-leeren Menge von Klassen t und zweier gerichteter Graphen v und a repräsentiert werden, wobei jede Klasse in jedem Graphen durch genau einen Knoten repräsentiert wird und die Kanten in v die direkten Vererbungsbeziehungen (*is-a*-Beziehungen) und die Kanten in a die Enthaltensbeziehungen (*has-a*-Beziehung, Aggregation) von je einer Klassen und einer Klasse eines (unbenannten) Feldes der ersten Klasse festlegen[10]. Ein Paar (t_1, t_2) in der Kantenmenge von v legt also fest, dass t_1 eine direkte Oberklasse von t_2 ist (transitive Vererbungsbeziehungen sind nicht als Tupel in v enthalten, spielen aber später eine Rolle). t_2 heißt *Ausprägung* der Oberklasse und aller Klas-

[10]Die Systematics-Tools erlauben auch das mehrfache Vorkommen einer Klasse als Klasse verschiedener Felder derselben Klasse. Um die Einführung von ansonsten für unsere Zwecke überflüssigen Variablennamen zu vermeiden, identifizieren wir aber o.B.d.A. jedes Feld nur indirekt über seine Klasse(-n), so dass die Unterscheidung zweier Enthaltenseins-Beziehungen nur durch Unterscheidung der dabei verknüpften Klassen möglich ist.

sen, von denen die Oberklasse eine Ausprägung ist. In der Kantenmenge von a bedeutet eine Paarung (t_1, t_2), dass t_2 in t_1 als Typ eines Feldes (*Aspekt*) enthalten ist. a enthält dabei auch solche Enthaltenseinsbeziehungen, die t_1 von einer Oberklasse erbt. Vererbung, wie sie in v definiert ist, meint genau die *Erweiterung* einer Unterklasse gegenüber einer Oberklasse um (Klassen von) Feldern (sogenannte *strikte Vererbung*), was gleichbedeutend mit einer *Einschränkung* der Klasse als Menge ihrer Instanzen ist. Das Überschreiben oder Entfernen von Klassen-Members ist nicht möglich (anders verhält es sich bei Selektionen, wie gleich beschrieben wird). v ist azyklisch und aufgrund einer gemeinsamen Oberklasse (d.h. der Wurzel der Vererbungshierarchie; bei Java z.B. die Klasse `Object`) zusammenhängend, a möglicherweise nicht-adjazent, sowie möglicherweise zyklisch zur Ermöglichung rekursiver Enthaltenseinsbeziehungen.

Ein derartiges Klassensystem ist prinzipiell geeignet als "Ersatz" für Und-Oder-Bäume, da es mittels v die Hierarchien der alternativen Ausprägungen und mittels a die der Aspekte in einem Und-Oder-Baum modellieren kann. Grob gesprochen, werden die Knoten im Und-Oder-Baum als benannte Klassen gedeutet, und die Kanten als Tupel in v bzw. a. Um die Klassen des Klassensystems von *Selektionen*, also den vom Sprachbeschreibern und Abfragern gebildeten Objekt-Kategorien, die wir ebenfalls als Klassen bezeichnen, zu unterscheiden, nennen wir die Klassen in v und a *benannte Klassen* (der Begriff "Ausprägung" meint aber nur eine Unterklasse einer benannten Klasse). Die Bildung von Selektionen kann nun als rekursive Auswahl von benannten Klassen beschrieben werden, bei der schrittweise zur Menge bereits selektierter benannter Klassen benannte direkte Unterklassen schon selektierter benannter Klassen sowie benannte direkte Unterklassen ihrer Aspekte hinzugefügt werden. Am Anfang dieses Prozesses sind nur eine bestimmte festgelegte benannte *Wurzelklasse* und ihre Aspekte selektiert, entsprechend der Wurzel des Und-Oder-Baums (in unserem Beispiel in 2.1 ist dies die Klasse 'Fahrzeug'). Die Erweiterung einer Selektion durch Hinzufügung von benannten Klassen (*Verfeinerung*) kann ebenfalls als eine Form der Vererbung zwischen Selektionen gedeutet werden, wobei jede Selektion genau die Selektionen als Unterklassen besitzt, in der sie als Knotenmenge enthalten ist. Selektionen werden so als Mengen (d.h. Klassen bzw. Kategorien) anderer Selektionen interpretierbar, aber auch als Mengen von Objekten der benannten Klassen. Durch diese Mengensichtweise auf Selektionen können wir später sehr einfach die Antwortmenge von Query-Selektionen mittels Mengen-Inklusionen beschreiben. Während aber die Vererbung benannter Klassen in v nur die Erweiterung um Aspekte kennt, kann eine Selektion auch durch Auswahl einer Ausprägung eines selektierten Aspekts als verfeinernde *Überschreibung* der Aspekt-Klasse (bzw. des Feldes, dessen Klasse der Aspekt ist) gebildet werden. Im Vererbungsmechanismus liegt somit der wesentliche Unterschied zwischen benannten Klassen und Selektionen: Während benannte Klassen erben, in dem sie die Menge der Enthaltensseinsbeziehungen der Oberklasse(-n) *erweitern*, kann eine "Unter-Selektion" sowohl durch Erweiterung gebildet werden, als auch durch Überschreibung (gdw. eine benannte Unterklasse eines vorher selektierten Aspekts selektiert wird). Eine bijektive Abbildung von der Menge der benannten Klassen auf die Menge aller Selek-

tionen ist zwar nicht möglich (z.B. gibt es in unserem Beispiel keinen eigenen Klassennamen für genau die Klasse der 'Fahrzeuge mit einem Wankelmotor'. Umgekehrt gibt es keine Selektion der Klasse 'Fahrzeugart', da diese Klasse nur als Aspekt auftaucht). Selektionen lassen sich aber textuell als *verschachtelte* Terme zusammengesetzt aus den Namen benannter Klassen eindeutig repräsentieren. Dies wird in Categories durch Prolog-Terme getan, und geschieht in Systematics entsprechend mit Verschachtelungen von XML-Elementen statt mit Termen, siehe 2.2.2.[11].

Einer Selektion über einem Und-Oder-Baum entspricht in unserem Klassensystem die Klassen-Auswahl, die bei der Erzeugung eines Objekts (Klasseninstanz) (einschließlich der Belegung von Feldern bzw. Membervariablen mit Objekten[12]) getroffen werden muss: Wird ein Objekt generiert, so wird dabei eine bestimmte benannte Klasse gewählt (und durch *Instanzierung* dieser Klasse das Objekt physisch erstellt), was die vorherige Wahl der Oberklassen dieser Klasse und deren Oberklassen usw. impliziert, sowie ggf. die der Unterklassen der Klassen der Felder der Objektklasse. Dabei ist die Wahl einer bestimmten alternativen Ausprägung unter einem Oder-Knoten gleichbedeutend mit der Bildung einer Unterklasse der benannten Klasse und gleichzeitig mit der Verfeinerung der bisherigen Selektion zu einer neuen Selektion, die von der vorigen Selektion erbt[13]. In diesem Sinne stellt eine in die Datenbank eingetragene vollständige Selektionen als Singleton ein einzelnes Objekt dar, und eine partielle Selektion, mit der man andere Selektion erfragen kann, eine Klasse von Objekten, und damit die Menge aller ihrer Unterklassen und Objekte. Die Deutung von Klassen bzw. Kategorien als Objektmengen und gleichzeitig als Mengen von verfeinerten Klassen ähnelt dem *YAT*-Modell für semistrukturierte Daten [12].

Um analog zu einem Und-Oder-Baum in einem manuellen Top-Down- Entscheidungsprozess der akkumulierenden Knotenauswahl eine Selektion erstellen zu können, muss das Klassensystem als ein geeigneter Entscheidungsbaum, den wir *Be-Have-Baum* nennen wollen, dargestellt werden. Da der Systematics-Schemaeditor gleichzeitig auf einen Be-Have-Baum (zur graphischen Darstel-

[11] Eine Termdarstellung erzeugt man für Und-Oder-Bäume wie für Klassensysteme bzw. Be-Have-Bäume dadurch, dass man hinter jeden Namen eines selektierten Knotens bzw. einer selektierten benannten Klasse in Klammern die Namen der selektierten Unterknoten bzw. -klassen und Aspekte setzt. Unvollständige Teilselektionen werden durch Platzhalter (_) markiert. Z.B. ergibt sich so für die Selektion 'Fahrzeuge mit einem Wankelmotor' der Term `Fahrzeug(Fahrzeugart(_), Motor(Wankel(AnzahlScheiben(_))))`.

[12] Etwa wie in Java mit dem Konstruktoraufruf in `fieldC = new(SubC);`, falls `fieldC` als vom Typ `C` deklariert wurde und `SubC` eine Unterklasse von `C` ist. Der Vergleich mit dem Java-Konzept der Variablen-*Überdeckung* ist eingeschränkt ebenfalls möglich, nicht aber mit der *Überladung* von Methoden.

[13] Betrachtet man eine Klasse als Menge ihrer möglichen unterschiedlichen Instanzen, so ist eine Oberklasse mindestens so mächtig wie jede ihrer Unterklassen, da jedes Objekt einer Unterklasse auch Instanz der Oberklasse ist. Betrachtet man alle Klassen, die über Unterklassen verfügen, als abstrakt (d.h. als nicht direkt instanzierbar, was aber keine zwingende Einschränkung ist), so entspricht diesem Vorgang genau die Erstellung einer vollständigen Selektion.

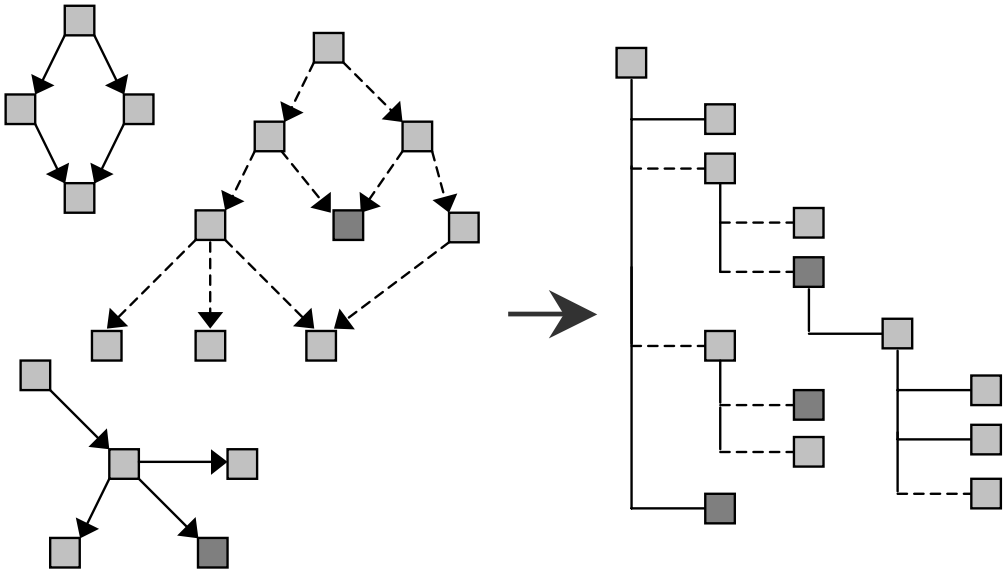

Abbildung 2: *Transformation eines Klassensystems in einen Be-Have-Baum. Die Kästchen symbolisieren benannte Klassen, die durchgezogenen Pfeile Aspektbeziehungen, die gestrichelten Pfeile direkte Vererbungsbeziehungen.*

lung) und ein dazu isomorphes, vor dem Benutzer verborgenes XML-Schema (zur internen Repräsentation, Speicherung als Datei und Übertragung via Internet) operiert, geschieht dies in der Praxis nicht in einem separaten Prozess, sondern verschränkt.

Der Algorithmus für die Umwandlung eines Klassensystems in einen Be-Have-Baum kann informell folgendermaßen skizziert werden (vgl. dazu Abbildung 2): v und a werden durch Auffaltung zu gerichteten Wurzelbäumen umgeformt (d.h. die Graphen werden durch Hinzufügung von neuen Knoten für alle Knoten mit mehrfachen zulaufenden Kanten und Zyklen zu (u.U. unendlich tiefen) Bäumen expandiert), bei denen alle Knoten durch die Namen der entsprechenden benannten Klassen markiert sind (Knoten-*Label* kennzeichnen also die Klassen des Klassensystems).

Die entstandenen Bäume werden dann zu einem einzigen Baum überlagert, in dem der aus v entstandene Baum b_v mehrfach von der Wurzel zu den Blättern durchlaufen wird, und bei jedem Durchlauf für jeden Knoten k in b_v mit Label t_1 eine Kante zu einem neuen Knoten mit Label t_2 hinzugefügt wird, falls in b_v oder den aus a entstandenen Bäumen eine Kante von einem anderen Knoten mit Label t_1 zu einem mit Label t_2 existiert, aber in b_v noch keine Kante von einem Knoten, der mit k nur über nicht aus a stammende Kanten (also Vererbungskanten) verbunden ist, zu einem Knoten mit Label t_2 existiert. Dabei werden Kanten, die durch Übername von Kanten aus a entstanden sind, als *have*-Kanten bezeichnet und die nachfolgenden Knoten als *Aspektknoten*, die anderen aus dem Klassensystem erzeugten Kanten und Knoten als *be*-Kanten

bzw. *Ausprägungsknoten*[14]. Diese Erweiterung von b_v geschieht solange, bis die Größe der Kantenmenge von b_v dabei unverändert bleibt. Der hieraus hervorgehende Be-Have-Baum b_v stellt direkte Enthaltenseinsbeziehungen, die eine Klasse von einer (direkten oder indirekten) Oberklasse erbt, als Kinder eines Unterklasse-Knotens dar (wie in a), gdw. der Knoten der Unterklasse nicht über be-Kanten mit dem Oberklasse-Knoten verbunden ist. Andernfalls stellt der Baum vererbte Aspekte nur als Kinder der Oberklasse-Knoten dar, was zu einer großen Platzersparnis führt. Wird vom Sprachbeschreiber ein Unterklassen-Knoten selektiert, der mit einer Oberklasse über be-Kanten verbunden ist, und sollen durch Selektion Aspekte der Unterklasse beschrieben werden, so geschieht dies so nahe wie möglich an der Baumwurzel bei Aspektknoten des Oberklasse-Knotens.

Beispiel:

Angenommen, 'Motor' in Abbildung 1 hätte einen Aspekt 'Material', und 'Hubkolbenmotor' und 'Wankelmotor' sind Unterklassen von 'Motor', so würden im Be-Have-Baum Material'-Knoten nur als Kind von 'Motor' auftauchen, nicht aber als Kinder von 'Hubkolbenmotor' und 'Wankelmotor', *ausgenommen* unter 'Hubkolbenmotor'- oder 'Wankelmotor'-Knoten, die *nicht* mit einem vorangehenden 'Motor'-Knoten nur über be-Kanten verbunden sind.

Systematics kennt neben Ausprägungs- und Aspektknoten noch die von Und-Oder-Bäumen her bekannten sogenannten Optionen (den zugehörigen Knotentyp nennen wir zur Unterscheidung von seiner Klasse *Optionsknoten*), sowie (exklusive) *Alternativen* bzw. *Alternativknoten*, die dieselbe Selektionslogik wie Ausprägungsknoten haben (maximal ein Kindknoten diesen Typs kann gleichzeitig selektiert sein), sich aber dahingehend unterscheiden, dass sie in keinem Fall Knoten graphisch erben. Auch die spätere Umsetzung in XML-Schemata ist grundlegend anders. Alternativen können mit den Argumenten des '|'-Symbols in Grammatiken formaler Sprachen (z.B. BNF-Notation) verglichen werden. Alternativknoten entsprechen mit ihrer einfacheren Semantik eher den Kindern von Oder-Knoten als den Ausprägungsknoten, zur Umsetzung von Oder-Knoten (genau gesagt: von deren Kindern) können aber im Prinzip beide Knotentypen alternativ verwendet werden, da ihre Selektionslogik dieselbe ist (näheres zu Alternativen siehe unten). Auch eine informelle Interpretation von Kanten zu Alternativknoten als vererbende "be-Kanten" durch den Anwender ist u.U. durchaus sinnvoll, allerdings werden nur Ausprägungen in XML-Schemata tatsächlich durch Vererbung von sogenannten *content models* von XML-Elementen, vgl. 2.2.2, umgesetzt. Grundsätzlich würde wie bei einem Und-Oder-Baum zur Erzeugung und Abfrage von Selektionen Alternativknoten ausreichen, allerdings haben Ausprägungsknoten den großen Vorteil, dass sie zur faktischen Vererbung von Knoten führen, was die Modellierung von

[14] In den Menüs der Systematics-Programme heißen Klassen "Types" und Ausprägungen "Type extensions", weil diese Begriffe in der XML-Schema-Definition benutzt werden, vgl. 2.2.2. Wir verwenden den Begriff "Typ" dagegen zur Einteilung der Knoten in Aspektknoten, Ausprägungsknoten usw.

Be-Have-Bäumen stark vereinfacht (der Schemaeditor fügt "ererbte Knoten" automatisch hinzu). Außerdem ist die XML-Darstellung von selektierten Ausprägungsknoten u.U. platzsparender (vgl. 2.2.2). Die Optionsknoten, die den Kindern nicht-exklusiver Oder-Knoten in Und-Oder-Bäumen entsprechen und im Gegensatz zu 'exklusiven' Oder-Knoten eine Selektion unabhängig von ihren Geschwistern erlauben, gehen nicht aus dem Klassensystem hervor, können aber (wenn man auf eine eigene Semantik für Optionen verzichten will) durch Aspektknoten ersetzt werden, die jeweils genau eine Alternative als Kind besitzen. Solche "Optionen" wurden in die Definition von Be-Have-Bäumen aufgenommen, da sich nicht-exklusive Oder-Knoten in Categories als in der Praxis sehr nützlich erwiesen haben.

Tabelle 1 (in Sektion 2.2.2) zeigt die informellen Entsprechungen von Be-Have-Bäumen, Klassensystemen, Grammatiken und Logik im Überblick.

Die wichtigsten Merkmale eines Be-Have-Baums lassen sich wie folgt zusammenfassen:

- Jeder Knoten ist mit dem Namen einer *benannten Klasse* markiert (Knoten-Label), die im Baum an Knoten auch mehrfach vorkommen kann.
- Der *Knotentypen* symbolisiert die Verwendung des Knotens in Bezug auf die Klasse des Elternknotens. Eine bestimmte benannte Klasse kann als Klasse mehrerer Knoten verschiedenen Typs gleichzeitig auftreten.
  Es gibt vier Knotentypen, die durch Kennzeichnung der zulaufenden Kanten symbolisiert werden: *Ausprägungsknoten* und *Aspektknoten* entstammen objektorientierten Klassensystemen, während die Semantik von *Alternativknoten* und *Optionsknoten* an Modellierungsmöglichkeiten von Grammatiken angelehnt sind.
- Kommt eine Klasse irgendwo im Baum als Klasse eines Aspektknotens vor, so heißt die hinführende Kante have-Kanten und die Klasse des Knotens *Aspekt* der Klasse des Elternknotens und aller ihrer Unterklassen. Kommt eine Klasse irgendwo im Baum als Klasse eines Ausprägungsknotens vor, so heißt die hinführende Kante be-Kanten und die Klasse *Ausprägung* der Klasse des Elternknotens und aller ihrer Oberklassen.
- Verschiedene Knotentypen dürfen unter Kindern eines bestimmten Knotens *gemischt* werden. Gleichnamige Knoten dürfen - entsprechend mehreren Feldern derselben Klasse, aber mit verschiedenen Variablennamen - als Geschwister-Knoten vorkommen (dies macht allerdings nur bei Optionsknoten und Aspektknoten Sinn, da von allen Ausprägungknoten und Alternativknoten unter den Kindern eines Knotens nur maximal einer in einer Selektion vorkommen kann und die Teilbäume mit Wurzeln gleichnamiger Knoten i.a deckungsgleich sind, siehe die nächsten Punkte).
- Vererbungsbeziehungen zwischen benannten Klassen führen zu einer entsprechenden Erweiterung der Menge der Kindknoten der Knoten der erbenden Klassen, da alle Aspekte, Alternativen und Optionen der

Oberklasse(-n) vererbt werden. Nicht graphisch dargestellt wird diese Vererbung aber unter den Knoten der Unterklasse, die mit der/den Oberklassen(-n)-Knoten nur über be-Kanten (Vererbungskanten) verbunden ist.

- Knoten mit gleichem Label (d.h derselben Klasse) sind Wurzeln deckungsgleicher Teilbäume (ausgenommen bei einem dieser Knoten kommen geerbte Aspekte/Optionen/Alternativen als Kindknoten vor, und bei einem anderen nicht). Gleichnamige Knoten sind also Repräsentanten für ein und dieselbe benannte Klasse. Dadurch können Verweisstrukturen modelliert werden, die über Bäume hinausreichen - ein Be-Have-Baum stellt faktisch einen gerichteten Graphen dar, in dem ein Knoten auch *mehrere* zulaufende Kanten haben kann. Auf die Selektionsmöglichkeiten hat dies keinen Einfluss, d.h. Kind-Knoten können unabhängig von den Kindern gleichnamiger Knoten selektiert werden.

Formal definieren wir einen Be-Have-Baum $T = (K, E, r, P)$ als einen gerichteten Wurzelbaum mit einer nicht-leeren Menge K der mit den Klassennamen benannten Knoten, einem Wurzelknoten r, einer Kantenmenge E mit gerichteten Kanten (a, b), $a, b \in K$, und einer Zerlegung P der Knotenmenge mit $P = (V, V', A, O)$, $V \cup A \cup V' \cup O = K \setminus \{r\}$ und $V \cap A \cap V' \cap O = \emptyset$, sowie

$$type(k) := \begin{cases} 0 & : \quad k \in V \\ 1 & : \quad k \in V' \\ 2 & : \quad k \in A \\ 3 & : \quad k \in O \end{cases}$$

[15] Knoten vom Typ 0 heißen *Ausprägungsknoten*, die zulaufenden Kanten *be*-Kanten, Knoten vom Typ 1 sind *Alternativknoten*, Knoten vom Typ 2 heißen *Aspektknoten*, ihre zulaufenden Kanten *have*-Kanten. Knoten vom Typ 3 heißen *Optionsknoten*.

$\{(k_1, k_2) : k_1 \in K, k_2 \in V\}$ heißt die *Vererbungshierarchie der benannten Klassen* von T und $\{(k_1, k_2) : k_1 \in K, k_2 \in A\}$ die Menge der *Enthaltenseinshierarchien* von T.

$C := \{label(k) : k \in K\}$ heißt die Menge der Namen der *benannten Klassen* in T (kurz "Klassen", obwohl dann u.U. die Gefahr der Verwechslung mit Selektionen besteht).

$c_1 \rhd c_2 :\Leftrightarrow \exists k_1 \in K, k_2 \in V : (k_1, k_2) \in E \wedge label(k_1) = c_1 \wedge label(k_2) = c_2$ definiert die direkte Vererbungsbeziehung $\rhd$ zweier benannter Klassen (vgl. dazu die folgende Bedingung 3). $c_1 \rhd_+ c_n :\Leftrightarrow \exists c_2, ...c_{n-1} \in C : c_1 \rhd c_2 \wedge c_2 \rhd c_3 ... \wedge$

[15] Kinder eines Knotens unterscheiden sich hier nur durch den Klassennamen und den Knotentyp, wodurch wir uns o.B.d.A. die Einführung von Variablennamen ersparen. Die Systematics-Tools erlauben aber auch das mehrfache Vorkommen derselben Klasse als Geschwisterknoten gleichen Knotentyps, was allerdings nur bei Aspekten und Optionen sinnvoll ist.

$c_{n-1} \rhd c_n$ definiert die transitive Vererbung.

Für Knotenrelationen gilt analog $k_1 \rhd k_2 :\Leftrightarrow k_1 \in K, k_2 \in V, (k_1, k_2) \in E$ und $k_1 \rhd_+ k_n :\Leftrightarrow k_1 \rhd k_2 \wedge k_2 \rhd k_3 ... \wedge k_{n-1} \rhd k_n$.

Wir definieren außerdem noch die *Signatur* eines Knotens als $sig(k) := (label(k), type(k))$ mit $k \in K$.

Zusätzlich gelten für T folgende Bedingungen (*Be-Have-Baum-Regeln*):

1. $\forall k_1, k_2, k_a \in K, (k_1, k_a) \in E : label(k_1) = label(k_2) \wedge$
   $\neg\exists k_{ac} \in K : (label(parent(k_{ac})) \rhd_+ label(k_1) \wedge k_{ac} \in A \cup O \cup V' \wedge sig(k_{ac}) = sig(k_a))$
   $\Rightarrow \exists k_b, (k_2, k_b) \in E : sig(k_a) = sig(k_b)$

   d.h. je zwei Knoten, die dieselbe benannte Klasse repräsentieren, sind die Wurzeln zweier deckungsgleicher Teilbäume (ausgenommen bezüglich "vererbter" Knoten, diese werden in Regel zwei eingeführt). Anders als in v oder a kann eine bestimmte Klasse im Baum ggf. durch mehr als einen Knoten repräsentiert werden (da ein Baumknoten nur eine zulaufende Kante haben kann), und jeder Knoten hat durch P einen der vier Knotentypen 0 bis 3. Um aber die Äquivalenz zum Klassensystem zu visualisieren, werden bei der Darstellung mit dem Systematics-Schemaeditor und -Webinterface die Kanten gemäß dem Typ des nachfolgenden Knotens unterschiedlich dargestellt (vgl. Abbildung 3), während die Knotensymbole selbst, vom Label abgesehen, äußerlich nicht unterschieden werden.

2. $\forall k_1, k_2, k_{2'} \in K, k_1 \rhd_+ k_2, k_a \in A \cup O \cup V', (k_1, k_a) \in E :$
   $label(k_{2'}) = label(k_2) \wedge k_{2'} \neq k_2$
   $\Rightarrow \exists k_b \in K : (k_{2'}, k_b) \in E \wedge sig(k_a) = sig(k_b)$

   Damit wird sichergestellt, dass ein Knoten Kindknoten zu geerbten Aspekten, Alternativen und Optionen besitzt ("erbt"), es sei denn, es handelt sich bei dem Knoten der erbenden Klasse um einen Ausprägungsknoten des Oberklasseknotens. Mit dieser Regel wird die Besonderheit von Ausprägungen und insbesondere der Unterschied zu Alternativen sichtbar, die sich, was die Selektions-Logik betrifft, gleich verhalten, siehe 2.2.1.

3. $\forall c_a, c_b \in C :$
   $\exists c_{11}, ..., c_{1n}, c_{21}, ..., c_{2m} \in C : c_a \rhd c_{11} \wedge c_{11} \rhd c_{12} \wedge ... \wedge c_{1n} \rhd c_b \wedge c_a \rhd c_{21} \wedge c_{21} \rhd c_{22} \wedge ... \wedge c_{2m} \rhd c_b \Rightarrow n = m \wedge c_{11} = c_{21} \wedge ... \wedge c_{1n} = c_{2m}$

   Hiermit wird ausgedrückt, dass es nicht mehrere unterschiedliche Vererbungs-Pfade zwischen denselben benannten Klassen geben kann.[16]

Ein Problem besteht hier darin, dass mit dem Be-Have-Baum keine benannten *abstrakten Klassen* (d.h. Klassen, die zwar Unterklassen besitzen und damit

[16] Die Einhaltung dieser Regel wird von der aktuellen Version des Schemaeditors nicht geprüft.

die Unterklassen mit beschreiben, selbst aber nicht in Selektionen vorkommen können) definiert werden können. Um dieses Problem zu umgehen, erlaubt der Systematics -Schemaeditor die Erstellung von Be-Have-Baum-*Wäldern*. Selektionen dürfen zwar nur über einem ausgezeichneten Baum im Wald erstellt werden, die anderen Bäume können aber als "Hilfsbäume" verwendet werden, um abstrakte Klassen zu definieren. Die genannten formalen Definitionen bleiben dazu sinngemäß gültig, mit dem Unterschied, dass T keinen zusammenhängenden Graphen, sondern eine Menge von Bäumen beschreibt. In der Praxis ist die Definition von abstrakten Klassen die Hauptaufgabe von Ausprägungen, während zur Modellierung von Entscheidungsalternativen meist Alternativen verwendet werden.

Und-Oder-Bäume können mittels folgender Entsprechungen als Be-Have-Bäume notiert werden:

- Knoten des Be-Have-Baums, die nur Typ 0-Kinder (Ausprägungen) *oder* nur Typ 1-Kinder (Alternativen) haben, heißen (exklusive) *Oder-Knoten*
- Knoten des Be-Have-Baums, die nur Typ 2-Kinder (Aspekte) haben, heißen *Und-Knoten*
- Knoten des Be-Have-Baums, die nur Typ 3-Kinder (Optionen) haben, heißen *nicht-exklusive Oder-Knoten*

Sonstige Knoten nennen wir *kombinierte Knoten*.

Analog zu Categories kennt Systematics neben Aspekt-, Alternativ-, Ausprägungs- und Optionsknoten auch *Textblätter* und *Tabellen*, die aber in der Theorie (trotz völlig verschiedener visueller Darstellung und Bedienung) wieder nur als praktische Vereinfachungen ensprechender Knoten-Konstellationen betrachtet werden (vgl. 2.1)[17].
Der Schemaeditor erlaubt dem Benutzer das freie Einfügen und Löschen von Knoten, sichert aber die Einhaltung der genannten Be-Have-Baum-Regeln, in dem er automatisch weitere Knoten hinzufügt oder löscht. Z.B. wird ein Knoten, der als Kind eines Knotens einer bestimmten benannten Klasse hinzugefügt wird, automatisch auch unter allen anderen Knoten dieser Klasse ergänzt.[18].

Abbildung 3 zeigt einen Be-Have-Baum und eine Selektion (vgl. 2.2.1) in der Darstellungsweise des Systematics-Webinterface.

[17] Im Schemaeditor werden Ausprägungs-Beziehungen (dort "Type extensions" genannt) als rote gestrichelte Kanten dargestellt und geerbte Kinder mit einem kleine 'i' (inheritance) an den zulaufenden Kanten, Kanten zu Alternativen dagegen als schwarze gestrichelte Kanten. Im Webinterface werden Ausprägungs- und Alternativbeziehungen gleich dargestellt, da die Selektionslogik identisch ist, d.h. für die Sprachbeschreiber und Abfrager ist der Unterschied beider Knotentypen transparent.

[18] Unendlich tiefe Äste, die durch Zyklen in a entstehen, werden dadurch dargestellt, dass alle Äste anfangs "eingeklappt" sind (also nur die Wurzel des Astes zu sehen ist), und erst auf Befehl um eine Ebene erweitert werden.

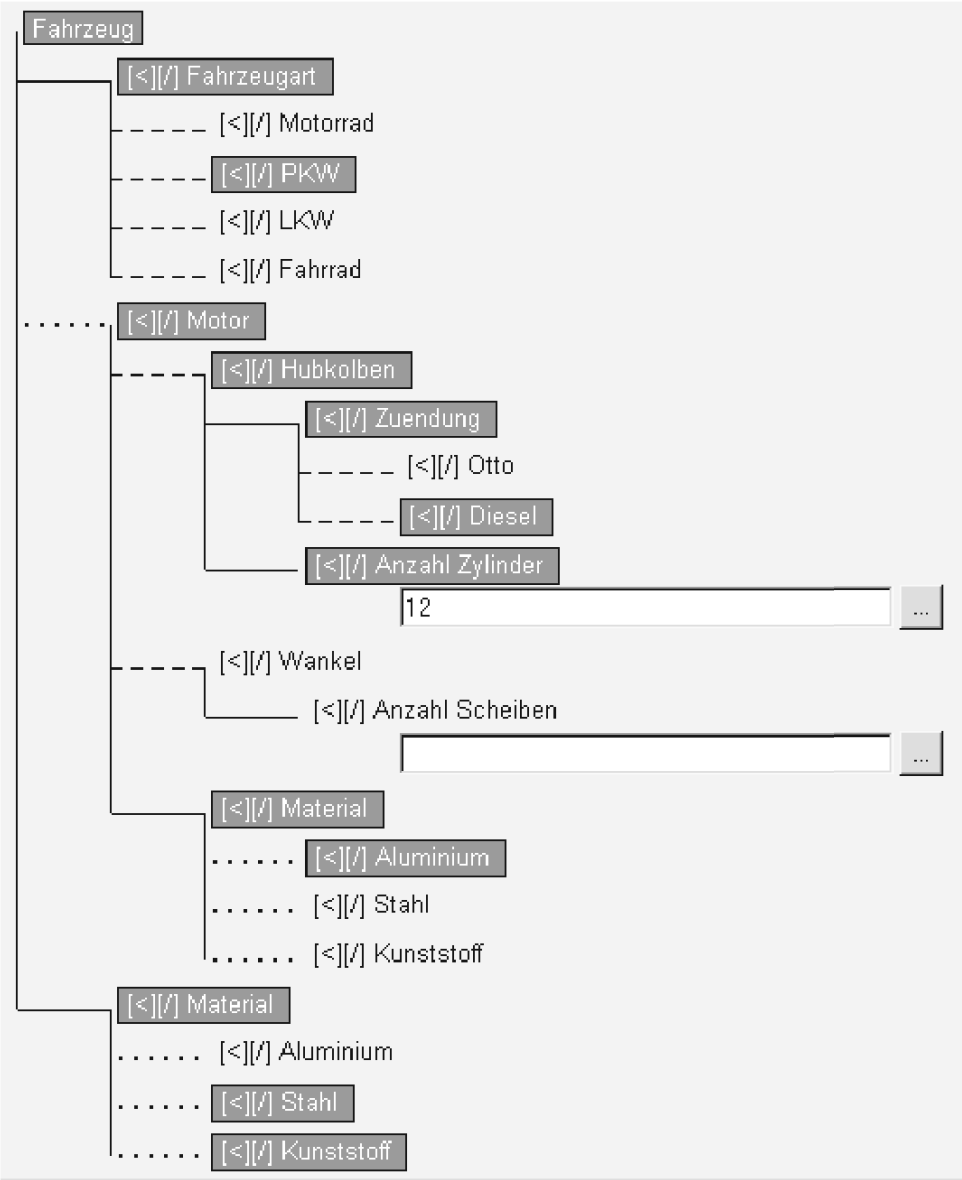

Abbildung 3: *Be-Have-Baum 'Fahrzeug' mit Selektion eines PKW mit Dieselmotor (selektierte Knoten sind mit inversem Label gekennzeichnet). Be-Kanten sind gestrichelt, Have-Kanten durchgezogen gezeichnet. Optionen (hier der 'Motor'-Knoten und die Kinder von 'Material') werden mit zulaufenden gepunkteten Kanten dargestellt. Der Knoten 'Fahrzeug' ist ein* kombinierter Knoten*, da er unterschiedliche Kind-Knotentypen mischt, was in beliebiger Weise möglich ist. Alle Vorkommnisse eines Unterbaums mit Wurzel 'Material' sind deckungsgleich (d.h. haben dieselben Kinder), da sie dieselbe Klasse 'Material' repräsentieren (die zu verschiedenen 'Material'-Knoten hinführenden Kanten könnten sich natürlich unterscheiden, was in diesem Beispiel aber nicht sinnvoll wäre. Auf die Selektionsmöglichkeiten der Kinder hat, wie das Beispiel zeigt, die Deckungsgleichheit ebenfalls keinen einschränkenden Einfluss.). Der 'Material'-Knoten taucht nur unter 'Motor' auf, nicht aber unter 'Hubkolbenmotor' und 'Wankel', da letztere Knoten mit be-Kanten mit 'Motor' verbunden sind. Trotzdem erben sie 'Material' als impliziten Aspekt und besäßen diesen Knoten als Kind, wenn sie außerhalb von 'Motor' auftreten würden.*

2.2.1 Selektionen über Be-Have-Bäumen

Eine Selektion über einem Be-Have-Baum wird analog zu einer Selektion über einem Und-Oder-Baum erstellt. Der Sprachbeschreiber oder Abfrager durchläuft den Baum von der Wurzel in Richtung zu den Blättern und fügt der Selektion die gewünschten Knoten hinzu. Jeder Aspekt muss dabei genau dann selektiert werden, falls sein Elternknoten bereits selektiert wurde. Von allen Ausprägungs- und Alternativknoten unter den Kindern eines selektierten Knotens kann dagegen nur höchstens einer selektiert werden, während von allen Optionsknoten eine beliebige Anzahl ≥ 0 selektiert werden kann. Formal definieren wir eine Selektion folgendermaßen:
Eine Selektion S über einem Be-Have-Baum $T = (K, E, r, (V, V', A, O))$ ist eine Teilmenge $S \subseteq K$ aller Knoten (die der *selektierten Knoten* des Be-Have-Baums), die folgenden Regeln gehorcht:

1. $r \in S$
2. $\forall k_1 \in S, k_2 \in K : (k_1, k_2) \in E \wedge k_2 \in A \Rightarrow k_2 \in S$
3. $\forall k_1 \in S : |\{(k_1, k_2) : (k_1, k_2) \in E \wedge k_2 \in V \cup V' \wedge k_2 \in S\}| \leq 1$[19]
4. $\forall k_1 \in K \backslash S, k_2 \in K : (k_1, k_2) \in E \Rightarrow k_2 \in K \backslash S$

Diese Regeln bewirken auch, dass T ohne die nicht in S enthaltenen Knoten zusammen mit einer entsprechenden Teilmenge von E weiterhin ein Baum ist (aber nicht mehr unbedingt dem Klassensystem entspricht), d.h. Selektionen sind selbst "baumförmig". Zusätzlich zu diesen Regeln sollten in einer zukünftigen Version von Systematics die von der Vorgänger-Software Categories [3] her bekannten *logischen Constraints* unterstützt werden, die dem Schema-Ersteller durch Regeln der Form $Teilselektion \Rightarrow Selektion$ eine weitere Verfeinerung der Menge gültiger (d.h. sinnvoller) Selektionen erlaubt.
Wieder analog zu Und-Oder-Bäumen heißt eine Selektion *vollständig* gdw. unter jedem selektierten Knoten, der Ausprägungen oder Alternativen als Kinder besitzt, mind. eines dieser Kinder selektiert ist, und *unvollständig* oder *partiell* andernfalls. Beide Arten von Selektionen dürfen mit Systematics jeweils sowohl vom Sprachbeschreiber in die Selektionen-Datenbank eingetragen als auch zur Datenbankabfrage benutzt werden, so dass der Begriff 'partiell' nur zur Veranschaulichung des Unterschiedes von Phänomenbeschreibungen (meist vollständige Selektionen) und Query-Selektionen (meist partiell) dient, aber keine formal relevante Bedeutung hat, wohingegen in Categories unvollständige Selektionen nicht in die Datenbank eingetragen werden sollten und nur als Patterns zur Query verwendet wurden. Der Systematics -Schemaeditor erlaubt außerdem die Angabe von *Default-Selektionen* für Textblätter, die im Systematics -Webinterface voreingestellt sind (vgl. 2.2.2 und 3.2). Die Angabe von allgemeinen Default-Selektionen (die beliebige Knoten-Selektionen beschreiben würden) ist aber in der Version 1.0 noch nicht möglich.
Für die Ergebnismenge $R(q)$ einer Query einer Selektionen-Datenbank mit der

[19] Selbstverständlich dürfen unterhalb verschiedener Knoten derselben Klasse unterschiedliche Kinder selektiert sein.

Menge D eingetragener Selektionen mittels einer (üblicherweise unvollständigen) *Query-Selektion* q gilt $R(q) := \{d \in D : q \subseteq d\}$. Die reflexive Halbordnung $\{(a, b) : a, b \in S, a \subseteq b\}$ der Selektionen eines Be-Have-Baums (zu der die minimale Selektion das Minimum ist) beschreibt also genau die Abfrage-Möglichkeiten einer Selektion und gleichbedeutend damit die Menge der linearen Vererbungshierarchien der Selektionen (die, wie am Anfang des Kapitels erläutert wurde, nicht identisch mit der Vererbungshierarchie der benannten Klassen sein müssen). Da jede Selektion den Wurzelknoten des Be-Have-Trees enthält, sind in der Antwortmenge zu einer Query-Selektion genau die Selektionen enthalten, die zur Query-Selektion eine Fortsetzung darstellen.
Bei Datenbank-Queries mit dem Systematics Webinterface (vgl. 3.2) dürfen Knoten in Selektionen auch *negiert* aufgenommen werden. In diesem Fall werden nur solche Einträge aus der Datenbank zurückgeliefert, in denen die negierten Knoten *nicht* enthalten sind. Um dies zu ermöglichen, erweitern wir die Definition von $R(q)$ zu $R(q) := \{d \in D : q \setminus qn \subseteq d \wedge qn \not\subset d\}$, wobei $qn, qn \subset q$ die Menge der negierten Selektionsknoten ist.
Aus der Darstellung von Be-Have-Bäumen und Selektionen im XML-Format ergeben sich später einige Konsequenzen, die über die genannte einfache Query-Semantik hinausführen und in 2.2.3 besprochen werden.

2.2.2 Be-Have-Baum- und Selektions-Darstellung in XML

Der Darstellung von Be-Have-Baum und Selektionen in XML liegt die Idee zugrunde, Selektionen als im Prinzip beliebige XML-Dokumente anzusehen, die Instanzen eines XML-Schemas sind (sogenannte in Bezug auf das Schema *gültige* oder *validierbare* XML-Dokumente), das zum Be-Have-Baum isomorph ist. Das XML-Schema kodieren wir selbst in XML, und zwar gemäß einer zur Verwendung mit Systematics vereinfachten Variante des *XML-Data*-Formats [4], die wir im folgenden allgemein als *XML-Schema* bezeichnen. Die textuelle Darstellung des Be-Have-Baums als XML-Schema kann zur Speicherung des Baums, dem Transfer im Internet und ggf. der Vearbeitung mit XML-Tools (z.B. einem Editor) benutzt werden, und liegt der Definition der XML-Repräsentation von Selektionen und Queries zugrunde. Die Systematics-Tools verwenden XML sowohl intern (als *DOM*-Datenstruktur) als auch als Dateiformat.
Im Gegensatz zum nachfolgenden *XML-Schema*-Standard [7] hat XML-Data keine Normung erfahren, sondern wurde lediglich als Diskussionsvorschlag beim W3C eingereicht, was zur Entwicklungszeit von Systematics nicht absehbar war. Allerdings sind die prinzipiellen Unterschiede zwischen XML-Data und diesem Standard marginal und eher syntaktischer als semantischer Natur (vgl. Tabelle 1). Auch eine Umsetzung nach *RDF* [10] oder *DCD* [5] sollte technisch relativ leicht möglich sein.
XML-Schema-Dokumente sind XML-Dokumente, die als Schemata andere XML-Dokumente (die Instanzen des jeweiligen Schemas) in ähnlicher Weise *grammatikalisch* beschreiben wie *DTDs* (Dokumententyp-Definitionen). Ein XML-Schema-Dokument entspricht in unserer Sichtweise einem Be-Have-Baum, die Schema-Instanzen entsprechen den Selektionen. XML-Schemata können DTDs immer ersetzen und erlauben dabei eine genauere Beschreibung der Instanzen

als letztere. Insbesondere kennen XML-Schemata etwas ähnliches wie eine objektorientierte Vererbung in Form von Elementtypen-Subclassing (was für uns von besonderem Interesse bei der Modellierung von Ausprägungen ist) und ermöglichen die exakte Spezifikation von Attributtypen. Der Hauptvorteil eines XML-Schemas liegt aber darin begründet, dass es sich dabei selbst um gewöhnliche XML-Dokumente handelt, die mit XML-fähigen Tools verarbeitet und dargestellt werden können.
Ein XML-Schema beschreibt XML-Dokumente durch sogenannte *Elementtyp*-Deklarationen. Eine solche Deklaration spezifiziert die Struktur und den Inhalt der XML-Elemente des jeweiligen Typs, indem sie festlegt, welche Elemente und Attribute in dem beschriebenen Element auftreten dürfen. Damit wird das sogenannte *content model* des Elements beschrieben. Elementtyp-Definitionen entsprechen in unserem Fall den Klassen des Klassensystems, wobei im Fall der Existenz mehrerer Knoten im Be-Have-Baum zur selben Klasse im XML-Schema nur eine einzige Elementtyp-Definition zu dieser Klasse existiert (Ausnahmen gelten für die Zuweisung von Default-Werten, siehe unten). XML-Schemata sind allerdings in einigen Aspekten leistungsfähiger als die objektorientierten Schemata bzw. Be-Have-Bäume, wie wir sie in 2.2 beschrieben haben. Ein wesentlicher Unterschied ist, dass XML-Schemata die Spezifikation einer variablen Anzahl von Feldern in einem Objekt bzw. von Kind-Elementen eines Elementes erlauben. Wäre dies mit Be-Have-Bäumen möglich, so könnten wir diese Bäume selbst durch ("Meta"-)Be-Have-Bäume beschreiben. Es wäre interessant, die Definitionen in diese Richtung zu erweitern.

Folgendes etwas vereinfachte und gekürzte XML-Dokumentenfragment repräsentiert einen Be-Have-Baum:

```
<?xml version=1.0 encoding=ISO-8859-1 ?>
    ...
    <elementType id='Fahrzeug' content='Closed'>
        <Description>Klasse aller Fahrzeuge</Description>
        <element occurs='REQUIRED' type='Fahrzeugart'/>
        <element occurs='OPTIONAL' type='Motor'/>
        <element occurs='REQUIRED' type='Material'/>
    </elementType>
    ...
    <elementType id='Motor' content='Closed'>
        <element occurs='REQUIRED' type='Material'/>
    </elementType>
    <elementType id='Hubkolbenmotor' content='Closed'>
        <element occurs='REQUIRED' type='Zuendung'/>
        <element occurs='REQUIRED' type='Zylinderzahl'/>
        <superType type='Motor'/>
    </elementType>
    <elementType id='Zuendung' content='Closed'>
        <group groupOrder='OR' occurs='OPTIONAL'>
            <element type='Otto'/>
```

```
        <element type='Diesel'/>
    </group>
</elementType>
<elementType id='Wankelmotor' content='Closed'>
    <element occurs='REQUIRED' type='Scheibenzahl'/>
    <superType type='Motor'/>
</elementType>
<elementType id='Scheibenzahl' gui='GUI_TextNode'>
        <Default>12</Default>
</elementType>
<elementType id='Material' content='Closed'>
    <element type='Aluminium' occurs='OPTIONAL'/>
    <element type='Stahl' occurs='OPTIONAL'/>
    <element type='Kunststoff' occurs='OPTIONAL'/>
</elementType>
...
```

Im Beispielcode sind die `<elementType>`-Elemente genau die Klassen des Klassensystems, das dem Be-Have-Baum unterliegt. Im Normalfall gehört zu jeder Klasse genau ein Elementtyp. Z.B. besitzt die Klasse 'Material' zwei Knoten, aber nur einen Elementtyp im XML-Schema (dies ist ein wesentlicher Unterschied zur XML-Darstellung von Selektionen, wo Mehrfachvorkommen gleichnamiger Knoten natürlich auch mehrfach im XML-Dokument repräsentiert werden müssen, da sich die Selektionszustände der zugehörigen Teilbäume ja unterscheiden können). Lediglich wenn wir (im Ausnahmefall) Feldern einer bestimmten Klasse mehrere *Default-Werte* für spätere Selektionen zuweisen wollen (für die gleichzeitige Verwendung an verschiedenen Positionen im Und-Oder-Baum) repräsentieren wir dieselbe Klasse im XML-Schema mehrfach. Klassen, die in einer Klasse als Klassen von Members enthalten sind (entsprechend den Aspekt-Knoten) werden durch `<element>`-Elemente mit Attribut `occurs='REQUIRED'` innerhalb des `<elementType>` der Elternklasse codiert, wobei das `type`-Attribut auf die Klasse mit dem `id`-Attribut gleichen Inhalts verweist (es handelt sich um Attribute des Typs ID/IDREF, also um Element-Querverweise bzw. Verweisziele). Dieser Elementtyp beschreibt dann das enthaltene "Aspekt-Element". Lautet das Attribut dagegen
`occurs='OPTIONAL'`, so ist das Kind ein Optionsknoten (im Beispiel gilt das für einen 'Motor' als Kind von 'Fahrzeug'). Die Ausprägungs-Beziehung wird durch `<superType>` hergestellt[20]. Das `type`-Attribut verweist auf die Oberklasse mit entsprechendem `id`-Attribut. Aspekte, die in einer Oberklasse vorkommen, werden nur im content model des Elementtyps der Oberklasse explizit aufgeführt, nicht aber in den Elementtypen der Unterklassen, die mit `superType` auf die Oberklasse Bezug nehmen. In Selektionen (bzw. den entsprechenden XML-Schemainstanzen), die einen Unterklassen-Knoten enthalten, tauchen (nur) un-

[20] Unser `superType` vereint die Eigenschaften des `superType` und des `genus` von XML-Data. Insbesondere dürfen mehrere `superType`-Elemente in einem `elementType` enthalten sein. Auf die Benennung als 'genus' wurde verzichtet, um keine Verwechslung mit dem linguistischen 'Genus' zu provozieren.

terhalb dieses Knotens die Aspekte, Optionen und Alternativen der Oberklassen aber auf. Jede Instanz (d.h. ein XML-Element mit einem Tag, der dem Wert des `id`-Attributs im `elementTyp` entspricht) eines Unterklassen-Elementtyps ist auch eine Instanz eines Oberklassen-Elementtyps, d.h. die Oberklassen-Instanz ist durch die Unterklassen-Instanz substituierbar, was eine Folge der strikten Vererbung benannter Be-Have-Baum-Klassen ist. Eine genaue Beschreibung der Schema-Semantik findet sich in [4].

Folgende Tabelle zeigt die verschiedenen Entsprechungen von Be-Have-Bäumen und anderen Datenmodellen im Überblick. Es handelt sich natürlich nur um informelle Entsprechungen, nicht um formale Übereinstimmungen:

Be-Have-Baum	Aspekte	Optionen	Ausprägungen	Alternativen
kontextfreie Grammatik, DTD	$V \rightarrow ab$	$V \rightarrow a\|ab\|b\|\epsilon$	$(O \rightarrow ab\|ac\|a$ $U_1 \rightarrow ab$ $U_2 \rightarrow ac)$	$V \rightarrow a\|b\|\epsilon$
XML-Schema (Systematics)	`<elementType> <element occurs= 'REQUIRED'>`	`<elementType> <element occurs= 'OPTIONAL'>`	`<superType>`	`<group groupOrder ='OR'>`
XML-Schema (W3C)	`<complexType> <element>`	`<complexType> <element minOccurs=0>`	`<extension base=...>`	`<choice>`
Klassensystem	Aggregation	-	Vererbung	(Vererbung)
Selektionslogik	$\wedge$	$\vee$	*xor*	*xor*

Tabelle 1: *Alternative Interpretationen zu Be-Have-Bäumen*

Alternativen werden in XML statt durch `<superType>` durch das XML-Schema-Element `<group>` mit Attribut `groupOrder=OR` ('OR-Gruppen') dargestellt (siehe im Beispiel im Elementtyp 'Zuendung'). OR-Gruppen haben etwa die Funktion des grammatikalischen Oder-Operators in DTDs und der BNF-Notation ('|'-Symbol). Die einzelnen Alternativen werden als Unterelementen der Gruppe aufgeführt (hier die Klassen 'Otto' und 'Diesel'). Andere Kinder von Knoten des Elementtyps, in dem die OR-Gruppe enthalten ist, werden im Be-Have-Baum und in Selektionen nicht als Kinder der Elementtypen repräsentiert, auf die in der OR-Gruppe verwiesen wird.
XML-Data kann außerdem den Typ skalarer Datentypen (z.B. String- und Integer-Daten) für XML-Elemente und Attribute festschreiben, was von Systematics aber nicht genutzt wird, da Systematics nur den Typ der visuellen *Darstellung* der Knoten des Be-Have-Baums im Webinterface (vgl. 3.2) festlegt und den Inhalt von Textblättern immer als uneingeschränkten String entsprechend den CDATA-Abschnitten von XML betrachtet. Textblätter werden im Webinterface als editierbares Formularfeld für Zeichenketten dargestellt (im Beispiel durch das Attribut `gui` (graphical user interface) des Elements mit `id`='Scheibenzahl')[21].

[21] Eine naheliegende Systematics-Erweiterung wäre die Typisierung entsprechend dem im

Neben den Schemata bzw. Und-Oder-Bäumen benötigen wir auch für die Selektionen eine XML-Darstellung, um diese als Phänomenbeschreibungen in einer XML-(fähigen)Datenbank ablegen und Query-Selektionen textuell darstellen zu können.
Wir legen für *XML-Selektionen* naheliegenderweise das Format der Instanzen des XML-Schemas zugrunde, d.h. gültige XML-Dokumente, die gegen das XML-Schema validierbar sind, wobei es als Einschränkung genau ein Wurzelelement in jeder XML-Selektion geben muss, das vom ersten `elementType` im Schema beschrieben wird ('Fahrzeug' in unserem Beispiel). Es ergibt sich dadurch eine textuelle Darstellung mit verschachtelten Elementen ähnlich der Prolog-Termdarstellung von Selektionen (vgl. 2.2). Die Schema-Instanzen enthalten keine Selektions-spezifischen Meta-Tags, sondern im Prinzip kann *jedes* XML-Dokument bei einem aus einem Be-Have-Baum generierten passenden Schema eine Selektion darstellen[22]. Jedem selektierten Knoten im Be-Have-Baum entspricht dann genau ein vom Wurzelelement ausgehender Pfad im XML-Dokument ("Pfad" im Sinn von *XPath* [13], wobei als wohl wichtigster Unterschied zu XPath Wildcards in unvollständigen Selektionen nur implizit am Ende eines Pfades auftreten können), und die Element-Tags entsprechen den Knoten-Labels, die auf die benannten Klassen verweisen[23]. Treten in einem Selektions-Teilpfad mehrere Ausprägungen mit aufsteigender Knotentiefe direkt hintereinander auf, so ist im entsprechenden Dokumenten-Teilpfad nur das gleichnamige Element zur letzten Ausprägung in dieser Ausprägungs-Sequenz enthalten (also zum "untersten" Elementtyp), was zu einer signifikanten Verminderung von Redundanz führen kann. Die Kindknoten vom Typ 1 oder 3 des/der Oberklasse(-n)-Knoten in dieser Sequenz selektierter Knoten werden in das XML-Element zur letzten Ausprägung übernommen. Wurde z.B. ein 'Motor'-Knoten und sein Kind 'Hubkolbenmotor' selektiert, resultiert daraus kein Element `<Motor><Hubkolbenmotor>...`, sondern nur `<Hubkolbenmotor>...`, wobei der Aspekt 'Material' der Oberklasse in der Selektion nun innerhalb von `<Hubkolbenmotor>` steht, also als `<Hubkolbenmotor> <Material> ...`[24]. Diese Regel gilt nicht für andere Knotentypen, und insbesondere nicht für Alternativen (die ja als OR-Gruppen notiert sind). Eine exakte Beschreibung dieser Regeln für die Bildung von Schema-Instanzen finden sich in [4].

Das folgende Beispiel zeigt eine partielle Selektion als XML-Dokument. Die Selektion ist unvollständig, da keine der möglichen 'Motor'-Ausprägungen ('Hubkolbenmotor' oder 'Wankelmotor') angegeben ist und die näheren Spezifikatio-

XML-Schema angegebenen skalaren Datentyp, so dass zum Beispiel nur Zahlen in das Textblatt eingetragen werden können.

[22]Die theoretisch durchaus möglichen unendlichen Selektionen können so natürlich nicht als XML-Dokument dargestellt werden, was sich in der Praxis aber nicht als ernsthafte Einschränkung erweist.

[23]Wie spezifizieren die Reihenfolge von Geschwister-Elementen und Knoten o.B.d.A. nicht. Technisch, insbesondere bei gleichnamigen Geschwister-Knoten und für die Umsetzung mittels XML-QL (vgl. 3.2), ist sie aber relevant.

[24]Daher ist es u.U. möglich, dass das Wurzelelement einer XML-Selektion einen anderen Namen trägt als die Be-Have-Baum-Wurzel.

nen vom 'Material' von Motor und Karosserie fehlen.

```
<Fahrzeug>
    <Fahrzeugart>
        <PKW/>
    </Fahrzeugart>
    <Material/>
    <Motor>
        <Material/>
    </Motor>
</Fahrzeug>
```

2.2.3 XML-Queries

In Sektion 2.2.1 wurde die Antwortmenge einer Query als Menge der Selektionen, in der die (üblicherweise partielle) Query-Selektion enthalten ist, definiert. Zur Funktion $\theta : S \rightarrow D$, die Selektionen auf Schema-konforme XML-Dokumente abbildet, gibt es eine Umkehrfunktion θ^{-1}, die zu einem solchen XML-Dokument die zugehörige Selektion liefert[25]. Damit ist $R_x(q_x) := \{\theta(r) : r \in R(\theta^{-1}(q_x))\}$ die Menge der XML-Dokumente, die auf eine Datenbank-Abfrage mit einer XML-Query-Selektion q_x zurückgeliefert wird (Definition von R siehe 2.2.1).
Ausgehend von dieser formalen Definition kann eine Query mit einer XML-Queryselektion q_x intuitiv folgendermaßen ausgewertet werden: Alle Blatt- Elemente (d.h. Elemente, die keine Elemente enthalten), deren Tag im XML-Schema eine Oberklasse bezeichnet (d.h. deren content model mit einer Elementtyp-Deklaration definiert ist, auf die irgendwo mit einem `superType`-Attribut verwiesen wird), fungieren als Platzhalter für gleichnamige Elemente oder ein Element einer Unterklasse mit beliebigem (auch textuellem) Inhalt (ein Blatt `<Motor>` in der Query würde also mit z.B. `<Hubkolbenmotor>...</Hubkolbenmotor>` matchen, oder mit z.B. `<Motor><Material><Stahl/><Aluminium/></Material></Motor>`). Alle anderen Blatt-Elemente sind Platzhalter für gleichnamige Elemente beliebigen Inhalts. Die Query fungiert also als ein Muster bestehend aus einem festen *Präfix* (dem XML-Query-Dokument) und einem nachfolgenden Wildcard, das auf beliebige XML-Fortsetzungen des Präfix "passt". Ist $L(q_x)$ die Menge aller XML-Dokumente, die mit diesem Muster matchen, und S_x die Menge aller Schema-Instanzen, so gilt $R_x(q_x) = L(q_x) \cap S_x$. Geht man von genau einem Be-Have-Baum als Beschreibungsschema aus, d.h. davon, dass nur Schema-konforme XML-Selektionen in die Datenbank eingetragen wurden (eine in der Praxis nicht unbedingt zutreffende Annahme, wie wir gleich sehen werden!), so gilt einfach $R_x(q_x) = L(q_x)$. Dies ist die Semantik, die der praktischen Umsetzung mit XML-QL zugrundeliegt (vgl. 3.2). Das Schema wird also datenbankseitig nicht benötigt, sondern dient lediglich als Hilfe bei der Selektions-Erstellung durch Sprachbeschreiber und Abfrager.

Beispiel:

[25] ohne Beweis

Enthält die Datenbank den Eintrag

```
<Fahrzeug>
    <PKW/>
    <Material>
        <Aluminium>
        <Kunststoff>
    </Material>
    <Wankelmotor>
        <Material>
            <Stahl>
        </Material>
        <Scheibenzahl>
            4
        </Scheibenzahl>
    </Wankelmotor>
</Fahrzeug>
```

so würde die in 2.2.2 genannte partielle Selektion mit diesem Eintrag matchen, aufgrund des nicht übereinstimmenden Fahrzeugtyps nicht aber mit z.B.

```
<Fahrzeug>
    <LKW/>
    <Wankelmotor>
    ...
    </Wankelmotor>
</Fahrzeug>
```

Das Beschreibungsschema hat die primäre Aufgabe, linguistisch sinnvolle und einem einheitlichen Konzept folgende Selektionen durch die Sprachbeschreiber zu erzwingen und es den Abfragern zu ermöglichen, die gespeicherten Selektionen in der Datenbank aufzufinden. Datenbank-technisch notwendig ist das Schema aber, wie gezeigt, nicht. Es ist möglich, auch nicht zum vom AVG-Projekt vorgegebenem Schema passende XML-Dokumente (*nichtkonforme Selektionen*) in der Datenbank abzulegen und mit geeigneten Query-Selektionen abzufragen. Aufgrund der Natur der auf die *semistrukturierten* (d.h. nicht unbedingt schematischen) XML-Dokumente zugeschnittenen Abfragesprache (vgl. 3.2) kommt es dabei nicht zu Interferenzen zwischen Selektionen unterschiedlicher Schemata, d.h. man kann problemlos mit verschiedenen Schemata auf die selbe Datenbank zugreifen. Der Eintrag von Selektionen über unterschiedlichen Schemata in derselben Datenbank kann notwendig sein, falls sich das AVG-Schema verändert hat, man aber schon gespeicherte Selektionen nicht verlieren oder aktualisieren möchte, oder falls für eine bestimmte Phänomenbeschreibung das vorgegebene Schema unpassend ist. Im letzteren Fall muss das Schema für den Sprachbeschreiber Client-seitig veränderbar sein. Categories besitzt diese Möglichkeit, da dieses Programm den Schemaeditor mit dem

Zugriffstool kombiniert, Systematics z.Zt. nicht - eine entsprechende Erweiterung wäre aber technisch leicht zu bewerkstelligen. Categories beschränkt allerdings solche Veränderungsmöglichkeiten auf Fälle, die im Original-Schema vorgesehen und speziell markiert sind (sogenannte "other-Knoten", zu denen vom Sprachbeschreiber und Abfrager Kinder hinzugefügt werden dürfen). Dies erscheint sinnvoll, um eine unkontrollierte Schema-Vielfalt (die zu linguistisch und konzeptuell-logisch unsinnigen Sprachbeschreibungen führen kann) zu vermeiden.
Das Problem liegt hier nicht bei der Speicherung solcher Selektionen und der technischen Abfrage auf Datenbankseite (dies ist mit Systematics bereits möglich), sondern (von inhaltlich-linguistischen Problemen, die durch parallel gültige, womöglich konkurrierende Schema-Versionen entstehen, einmal abgesehen) bei der Erstellung von Query-Selektionen durch die Abfrager. Diesen muss, da aus naheliegenden Gründen eine freie, textuelle Eingabe von Queries in Form von XML-Code nicht in Frage kommt, ein Schema zur Verfügung gestellt werden, mit dem sie die Datenbank abfragen können, ohne dass ihnen etwaige nichtkonforme Selektionen verborgen bleiben. Folgende Lösungsansätze, die auch kombinierbar sind, wären denkbar:

- Der AVG-Server besitzt eine Versionsverwaltung für verschiedene vom AVG-Projekt erstellte Schemata und erlaubt den Anwendern die Auswahl eines geeigneten Schemas. Dieser Ansatz ist naturgemäß relativ starr. Phänomenbeschreibungen müssen entweder schemakonform sein oder redaktionell in ein Schema eingearbeitet werden.

- Das Schema ist Client-seitig veränderbar (beliebig oder eingeschränkt), so dass ein Abfrager eine größere Freiheit bei der Query-Erstellung erhält. Hier besteht die Gefahr, dass Phänomenbeschreibungen durch eine *zu große* führungslose Freiheit bei der Query-Erstellung nicht mehr gefunden werden. Allerdings ist eine (eingeschränkte) Client-seitige Schema-Veränderbarkeit eine Vorraussetzung für die zukünftige Implementation von sogenannten $Partitur - Beispieltabellen$. In solchen Tabellen, deren Zellen-Zahl vom Schema- Ersteller nicht statisch festgelegt werden kann (daher die Notwendigkeit der Schema-Veränderung) wird vom Sprachbeschreiber zu jeder gewählten sprachlichen Kategorie in einer variablen linguistischen Notation (*Partitur*) ein konkretes Beispiel angegeben.

- Alle Schema-Änderungen werden in einem einheitlichen Schema zusammengeführt. Alte Selektionen werden automatisch (oder nach einer redaktionellen Beurteilung) auf das neue Schema umgestellt. Dies erfordert eine möglicherweise technisch aufwendige Umsetzung und ist in vielen Fällen nicht oder nur mit manueller Beihilfe überhaupt möglich.

- Das Schema (oder ein Teil davon) wird aus den vorhanden Selektionen automatisch generiert. Dieses Verfahren ähnelt dem der Erzeugung von *Dataguides* [14] aus schwach strukturierten Datenmengen. Ein Problem könnte hier darin bestehen, dass zu XML-Selektionen nicht eindeutig ein zugrundeliegendes Schema festgestellt werden kann, da die Knotentypen

nicht aus den XML-Elementen hervorgehen. Eine mögliche Lösung hierfür wäre, nur 'Optionen' als den flexibelsten Knotentyp zu generieren, die Kotentypen zu speichern, oder einen Vergleich mit dem Schema durchzuführen (was wohl nicht immer zu einem eindeutigen Resultat führt).

- Selektionen werden mit dem "falschen" Schema erfragt. Dies ist möglich, falls die gesuchten Phänomenbeschreibungen in der oben beschriebenen Menge $L(q)$ enthalten sind.

Jeder dieser Ansätze besitzt also Vor- und Nachteile, so dass für zukünftige Versionen von Systematics vermutlich eine Kombination die beste Lösung bietet.

3 Anwendungsarchitektur

Abbildung 4 zeigt die Systematics-Architektur im Überblick, bestehend aus den beide Komplexen Schemaeditor und Datenbank mit Webinterface. Der *Schemaeditor* dient zur Erstellung des AVG-Klassensystems in Form eines BeHave-Baums. Da der Schemaeditor als Client-/Server-Anwendung ausgeführt ist, können via Internet mehrere Linguisten an verschiedenen Orten gleichzeitig am selben zentralen Schema arbeiten. Dieses Schema wird auf einem Server der AVG als XML-Schema gespeichert und durch einen XSLT-Prozessor gemäß einem für Systematics entwickelten Stylesheet (*Schema-Styler*) in eine DHTML-Webseite (*Webinterface*) umgewandelt, die das Beschreibungs-Schema als BeHave-Baum graphisch darstellt und die interaktive Erstellung von Selektionen erlaubt. Diese DHTML-Seite verwenden die Sprachbeschreiber und Abfrager zum Speichern bzw. Abfragen der Daten der zentralen AVG-Selektionendatenbank, die auf einem Server der AVG eingerichtet ist. Abgesehen vom Webbrowser ist auf der Seite der Sprachbeschreiber und Abfrager keine weitere technischen Infrastruktur notwendig, d.h. es müssen weder Applikationen installiert noch Applets oder Plugins geladen werden. Die Verbindung des Browsers mit der Datenbank stellt ein Java-Servlet her, das auf dem AVG-Server ausgeführt wird, und via HTTP Selektionen von der DHTML-Seite entgegennimmt (zum Eintrag in die Datenbank oder zur Verwendung als Muster für eine Datenbankquery) und zur DHTML-Seite sendet (als Antwort auf eine Query). Bei der eigentlichen Datenbank handelt es sich bei dem implementierten Prototypen um ein einziges XML-Dokument, zu dem Selektionen durch einfaches Anhängen hinzugefügt werden. Die Abfrage der Datenbank übernimmt ein Prozessor der XML-Abfragesprachen XML-QL [6], der vom Servlet angesteuert wird. XML-QL, das wiederum auf dem *Strudel*-Ansatz zur Webseiten-Strukturierung aufsetzt [15], extrahiert auf sehr effiziente Weise die Antwort-Selektionen, die mit der Query-Selektion matchen, so dass trotz der nicht mit einem Index versehenen Datenbank bis zu einer Datenbankgröße von ca. 5 MB auf einem aktuellen PC eine für Demonstrationszwecke ausreichende Geschwindigkeit bei der Abfrage erreicht wird, falls die gespeicherten Selektionen nicht allzu komplex sind.

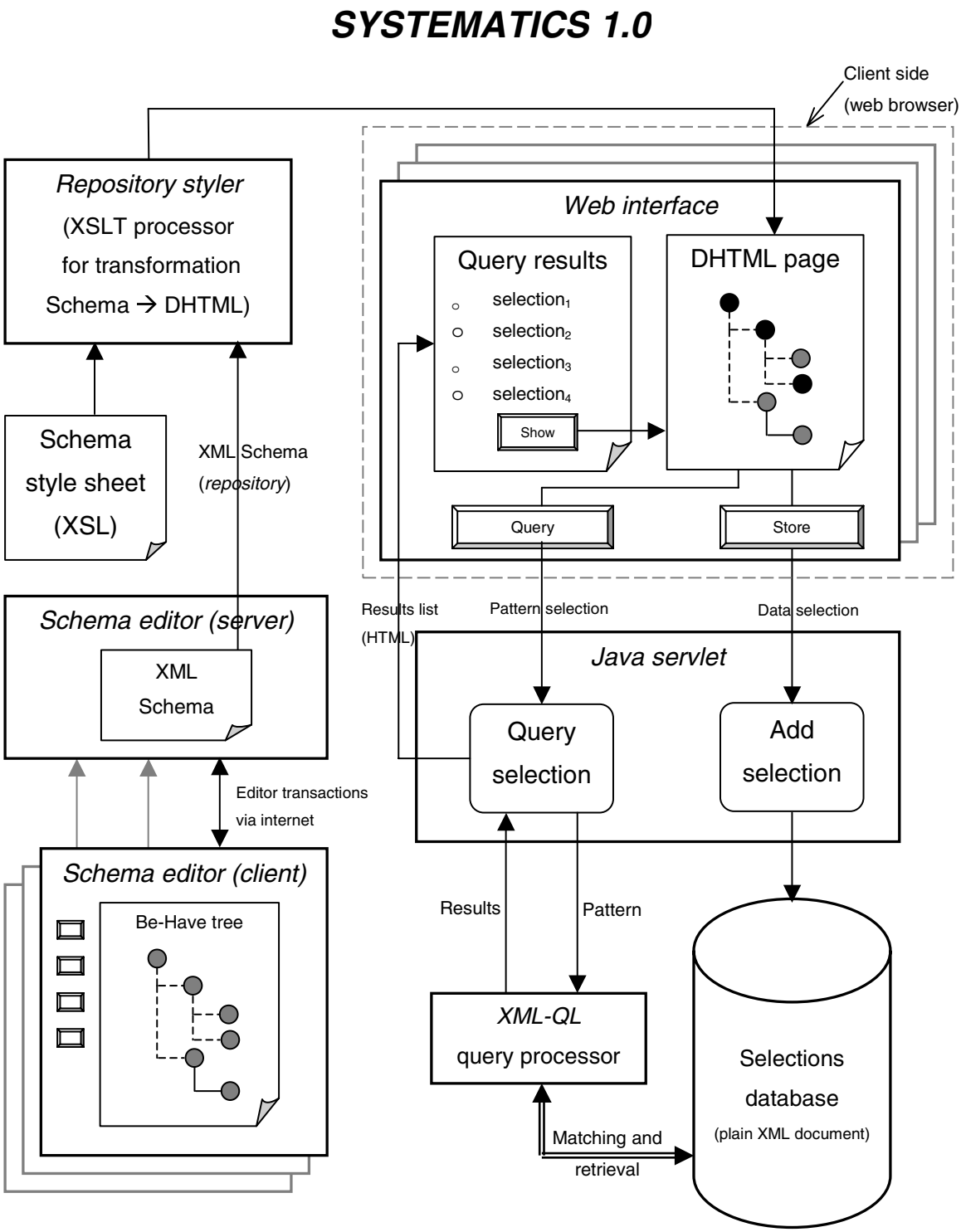

Abbildung 4: *Überblick*

3.1 Schemaeditor

Der Systematics Schemaeditor wurde als verteilte Anwendung ausgeführt, bei der mehrere Schemaeditor-Clients mit einem Schemaeditor-Server via Internet kommunizieren. Bei Client und Server handelt es sich um die selbe, in Java 1.1 implementierte Anwendung, die in zwei Modi ablaufen kann. Der Client-Modus dient der interaktiven Erstellung und Bearbeitung des Beschreibungs-Schemas in Gestalt eines Be-Have-Baums, während im Server-Modus die von den Clients eingehenden Änderungen am Schema in einem einzigen Schema zusammengeführt und synchronisiert werden. Zu diesem Zweck verwaltet der Server das Schema in einer Schemaeditor-internen zentralen Mehrbenutzer-Datenbank[26], dem sogenannten *Repository*. In dieser in XML codierten Datenbank werden neben dem eigentlichen Schema auch Zusatzinformationen gespeichert, die durch den Mehrbenutzerbetrieb notwendig sind und den Dateiattributen von Dateisystemen ähneln (insbesondere Eigentümerinformationen, Zeitangaben für Generierung und Änderungen, sowie Zugriffsrechte). Zu diesem Zweck wurde das Format zur Repräsentation des Schemas um entsprechende Elemente und Attribute erweitert. Die Clients halten lokale Kopien des Repository, die im Falle einer Änderung synchronisiert werden. Die Kommunikation der Clients mit dem Server verläuft über TCP/IP-Socket-Verbindungen, über die Transaktionen übermittelt werden, die Änderungen am Schema beschreiben. In Richtung von Client zum Server handelt es sich dabei neben Verwaltungstransaktionen (z.B. zur Anmeldung des Benutzers am Server) um die Editier-Befehle des Client-Anwenders (1:1-Verbindung), während in umgekehrte Richtung die Editier-Befehle der anderen Clients gebroadcastet werden, um das Client-lokale Schema zu aktualisieren (1:n-Verbindung). Neben dieser online-Arbeitsweise können die Clients auch offline betrieben werden, d.h. ohne Internet-Verbindung zum Server. In diesem Fall werden die anfallenden Transaktionen gesammelt und zum Beginn der nächsten online-Sitzung zum Server übermittelt.
Durch die Implementation des Schemaeditors in Java kann das Programm ohne Anpassungen unter jedem gängigen Betriebssystem eingesetzt werden. Der Schemaeditor stellt dem Benutzer über eine in Java-Swing ausgeführte MDI-Benutzeroberfläche (multi-document interface) neben den Funktionen eines Graph-Editors (Erstellen, Löschen, Einfügen, Kopieren von Ästen, Drucken, Suchen mit regulären Ausdrücken etc.) auch Funktionen zur Verfügung, die sich aus dem Mehrbenutzerbetrieb ergeben (Rechteverwaltung). Wie unter 2.2 geschildert wurde, stellt der Schemaeditor bei der Bearbeitung sicher, dass die Baumdarstellung äquivalent zum repräsentierten Klassensystem ist, dass also z.B. alle Knoten, die eine bestimmte Klasse darstellen, als Wurzelknoten übereinstimmende Teilbäume besitzen. Neben der Be-Have-Baum-Bearbeitung wird aber auch die Möglichkeit geboten, den dem Be-Have-Baum zugrundeliegende XML-Code auf Quelltextebene sowie mit einem speziellen Strukturediteor direkt zu bearbeiten. Im Prinzip kann der Schemaeditor außerdem auch genutzt werden, um mehrere Be-Have-Bäume gleichzeitig zu erstellen (vgl. 2.2).
Abbildung 5 zeigt den Schemaeditor im Client-Modus.

[26] nicht zu verwechseln mit der Selektionen-Datenbank, vgl. 3.2.

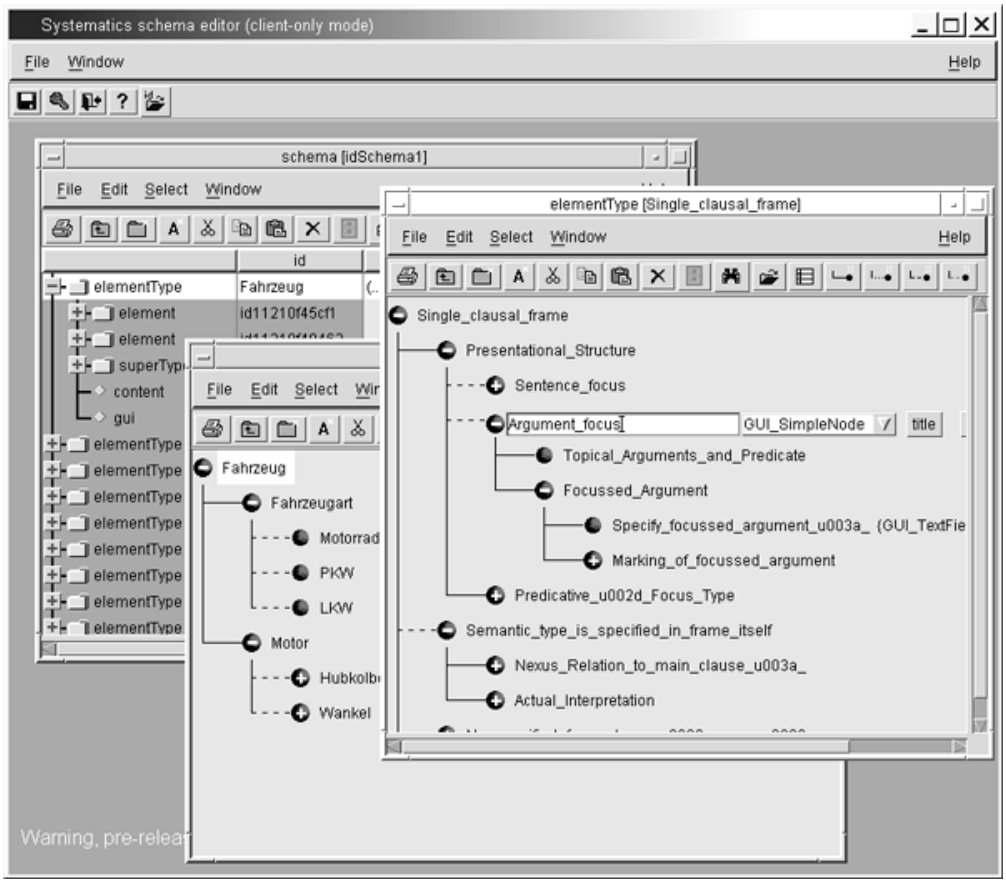

Abbildung 5: *Systematics Schemaeditor*

3.2 Webinterface und Datenbankzugriff

Der Teil von Systematics, der für die Gruppe der Sprachbeschreiber und Abfrager gedacht ist, untergliedert sich in vier Module:

- der *Schema-Styler* generiert aus dem mit dem Schemaeditor erstellten XML-Schema das sogenannte *Webinterface* zur interaktiven Erzeugung und Darstellung von Selektionen.
- das *Webinterface* ist eine in DHTML (dynamisches HTML) codierte Webseite, d.h. eine HTML-Seite, die nach der Darstellung mit dem Webbrowser vom Benutzer interaktiv verändert werden kann, ohne auf den Webserver zugreifen zu müssen. Sie stellt das Schema als Be-Have-Baum dar und erlaubt die interaktive Erstellung von Selektionen.
- ein Java 2-*Servlet* kommuniziert mit dem Webinterface, trägt Selektionen in die Selektionen-Datenbank ein und steuert den *XML-QL-Interpreter* an, der die Datenbankabfrage übernimmt.
- ein von AT&T stammender *XML-QL-Interpreter* durchsucht die Datenbank nach zur Query passenden Selektionen.

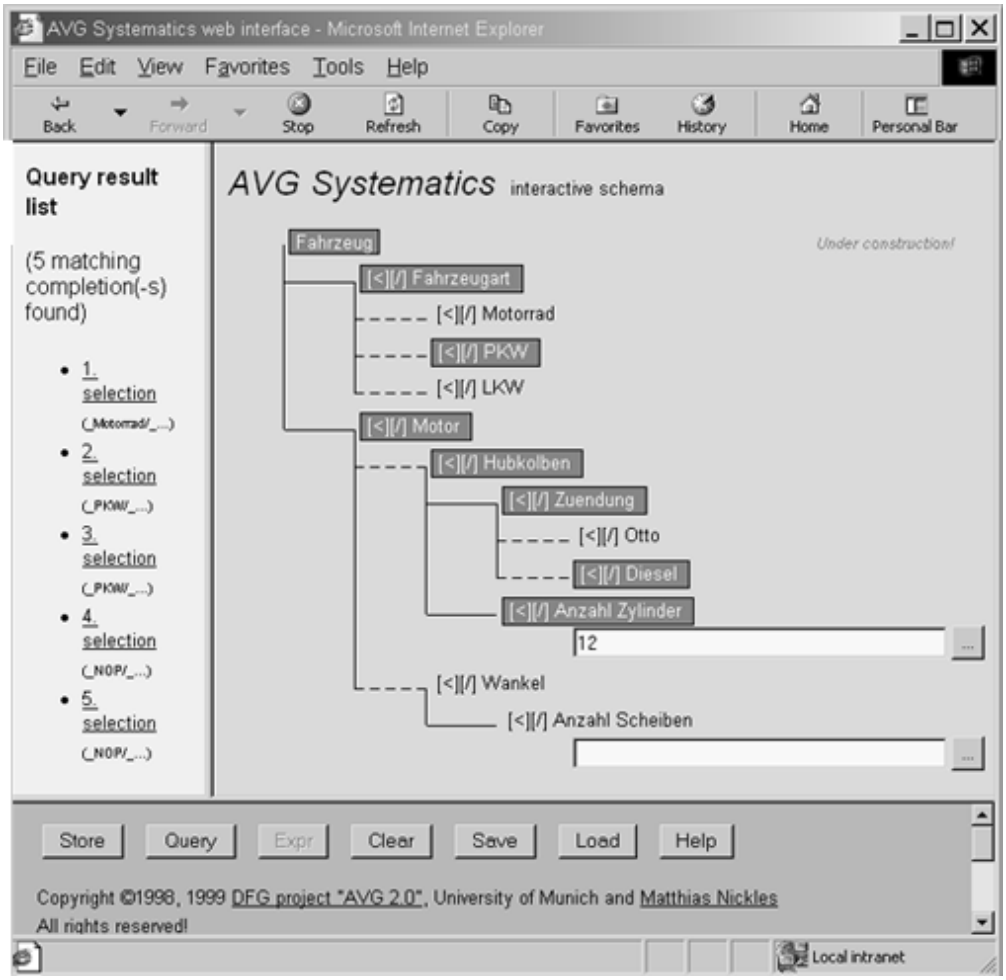

Abbildung 6: *Systematics Webinterface im Internet Explorer*

Der Schema-Styler besteht aus einem komplexen *XSLT-Stylesheet* (XSL = *Extensible Stylesheet Language* zur Darstellung eines XML-Dokuments und seiner Umwandlung in ein anderes, wobei XSLT die für letztere Aufgabe vorgesehene Teilsprache ist [8]), das die Regeln für die Transformation des XML-Schemas in das DHTML-codierte Webinterface übernimmt, sowie einem von IBM stammenden XSLT-Prozessor (*LotusXSL* [17]), der diese Transformation ausführt[27]. Das Webinterface stellt das Schema als Be-Have-Baum in etwa der selben Form dar wie der Schemaeditor, mit dem wesentlichen Unterschied, dass hier durch einfaches Mausklicken auf die gewünschten Knoten, ggf. der Negation von selektierten Knoten (vgl. 2.2.1) und durch Eintragung von Text in Textblätter und Tabellen Selektionen erstellt werden können, während die Baumstruktur nicht veränderlich ist. Dazu ist nur ein Webbrowser nötig, der das *DOM* (docu-

[27] Da zum Zeitpunkt der Systematics-Entwicklung der XSL-Standard noch nicht endgültig vom W3C festgeschrieben war, kann unser Stylesheet nur mit der damals aktuellen Version von LotusXSL verwendet werden. Eine Anpassung an die aktuelle XSLT-Version sollte aber leicht möglich sein.

ment object model) für die automatische Verarbeitung von HTML-Seiten und das DHTML-Scripting nach dem *ECMAScript*-Standard beherrscht (der standardisierten Variante von JavaScript) - selbst das Servlet wird nur gebraucht, wenn ein Zugang zur Selektionen-Datenbank benötigt wird. Das Webinterface stellt zu den Knoten auch mit dem Schemaeditor vorgegebene Hilfetexte dar und trägt in Textblätter Default-Selektionen ein (dieser Mechanismus sollte in einer zukünftigen Version auf allgemeine Default-Selektionen erweitert werden). Außerdem ist das lokale (also Client-seitige) Speichern und Laden von Selektionen möglich.
Nachdem der Sprachbeschreiber oder Abfrager die Daten- bzw. Query-Selektion erstellt hat, sendet er sie an das auf dem AVG-Server installierte Servlet, das im ersteren Fall die Selektion an die Datenbank-Datei anhängt. Im letzteren Fall generiert das Servlet aus der (üblicherweise partiellen) Query-Selektion einen XML-QL-Ausdruck[28] und ruft den XML-QL-Interpreter auf, der die Selektionen-Datenbank nach mit der Query-Selektion matchenden Einträgen durchsucht. Die Ergebnisliste wird dann vom Servlet in eine HTML-Seite umgewandelt und als Frame im Webinterface dargestellt (ohne die eigentlichen Ergebnis-Selektionen schon zum Browser zu übertragen). Der Abfrager kann dann Resultate aus der Ergebnisliste durch Anklicken auswählen, worauf das Servlet die Antwort-Selektion zum Client überträgt und das Webinterface sie über dem Be-Have-Baum graphisch darstellt. Danach kann die Selektion editiert und z.B. für eine erneute (verfeinerte) Query verwendet werden. Diese Technik des kombinierten Datenbank-Browsens und Query-Erstellens ähnelt der *Query-In-Place*-Technik des interaktiven Datenbank-Interfaces *PESTO* [16].

Abbildung 6 zeigt das Webinterface, wie es mit dem MS Internet Explorer 5.5 dargestellt wird. Im linken Frame erscheint nach einer Query die Liste der resultierenden Antworten. Jede dieser Antworten kann im rechten Frame als Selektion dargestellt werden. Über dem Be-Have-Baum rechts werden nicht nur die abgefragten Daten dargestellt, sondern auch neue Daten und Queries durch Knotenselektion (durch Anklicken) und ggf. Ausfüllen von Textblättern und Tabellen erstellt - ob die über dem Baum gerade angezeigte Selektion zur Abfrage verwendet oder der Datenbank hinzugefügt wird, hängt allein vom abschließenden Befehl ab ("Query" bzw. "Store"). Durch Klicken auf das Symbol "/" rechts neben den Knotennamen können Knoten *negiert* werden (vgl. 2.2.1).

XML-QL ist eine der frühesten XML-spezifischen Abfragesprachen für semistrukturierte Daten und orientiert sich anders als z.B. XQL [9] stärker an den Besonderheiten von XML, anstatt SQL nachzubilden, und ist am ehesten mit XPath [13] -basierten Sprachen wie dem W3C-Standard *XQuery* [19] vergleichbar. Wie bei den meisten Abfragesprachen für XML besteht hier ein Query-Ausdruck (vergleichbar mit dem `SELECT...FROM...WHERE...`-Ausdruck von SQL) aus einem *Pattern* und einem *Konstruktor*. Das Pattern ist ein wohl-

[28] Der Schema-Styler kann optional auch Queries in Form von Prolog-Termen (mit Platzhaltern) generieren, damit wie mit dem Systematics-Vorgänger Categories statt einer XML-Datenbank eine Klausel-Datenbank verwendet werden kann. Dieses Feature ist aber als experimentell einzustufen.

geformtes XML-Fragment, das existenzquantifizierte Variablen enthalten kann. Der Konstruktor ist ebenfalls ein XML-Fragment, in dem die Variablen aus dem Pattern vorkommen dürfen. Der Query-Prozessor versucht das Pattern mit den XML-Dokumenten in der Selektionen-Datenbank zu matchen, wobei die Variablen als Wildcards fungieren (generiert als neue Kinder von Blatt-Elementen in den XML-Selektionen, vgl. 2.2.3). Matcht ein Element des XML-Dokuments mit dem Pattern, werden die Variablen im Konstruktor entsprechen instanziert und der so vervollständigte Konstruktor zum Ergebnis-Dokument hinzugefügt. Im Fall von Systematics wird die Konstruktion des Resultats allerdings aus Effizienzgründen vom Servlet statt von XML-QL übernommen, da in unserem Fall die Struktur von Query-Selektion und Antwortselektion ja gleich ist (vom Umfang abgesehen) und eine komplette Neukonstruktion der Resultate daher überflüssig ist.

Folgendes Beispiel zeigt die zu einer partiellen Selektion gehörige XML-QL-Abfrage (die Originalabfrage wurde etwas vereinfacht). Der Konstruktor befindet sich im Element `<qresult>`, das Pattern hinter `WHERE`, die Datenbank-Datei ist nach `IN` angegeben:

```
function query () { \\ automatically generated by Systematics
    ExperimentalDBMS CONSTRUCT
    <qresult>
        <v1>$v1</v1><v2>$v2</v2>
    </qresult> WHERE
    <selections>
        <selection>
            <Fahrzeug>
                <Fahrzeugart>
                $v1
                </Fahrzeugart>
                <Motor>
                $v2
                </Motor>
            </Fahrzeug>
        </selection>
    </selections> IN
    file://localhost/D:/Programme/Systematics/.../selections.xml
}
```

4 Zusammenfassung und Ausblick

Die Entwicklung und Anwendung von Systematics hat gezeigt, dass Kategoriensysteme zur Beschreibung natürlicher Sprachen als objektorientierte Klassensysteme gefasst und als XML-Schemata repräsentiert werden können. Außerdem wurde die prinzipielle Umsetzbarkeit in Form eines XML-basierten Datenbanksystems gezeigt, das sich interaktiv auf sehr einfache Weise bedienen lässt. Für den Eintrag und die Abfrage von Daten haben sich Selektionen über Be-Have-Bäumen bewährt, die zur graphischen Darstellung von Klassensystemen für die Verwendung mit Systematics entwickelt wurden. Allerdings handelt es sich bei den hier beschriebenen Modulen (Systematics 1.0) um Prototypen, die für einen nicht-experimentellen, nicht-demonstrativen Einsatz nicht oder nur eingeschränkt tauglich sind. Als wesentliche Einschränkung ist die mangelnde Eignung für große Datenmengen zu nennen, was insbesondere das Webinterface und XML-QL betrifft. Daneben fehlen noch einige von der Seite der Linguisten gewünschte Funktionen, wie z.B. die statistische Auswertung von Query-Resultaten (u.a. zum quantitativen Vergleich mehrerer Sprachen), Client-seitig veränderbare Schemata (vgl. 2.2.3, darunter insbesondere die Implementierung dynamischer Partitur-Tabellen), die Löschbarkeit bereits in die Datenbank eingetragener Selektionen (was eine Prüfung der Autorisation der Sprachbeschreiber erforderlich macht, z.B. durch Passwort-Vergabe), sowie logische Selektions-Constraints. Außerdem sollte die Definition von Be-Have-Bäumen in Hinblick auf die Unterstützung weiterer Fähigkeiten von XML-Schemata erweitert werden, insbesondere um die Deklaration von Elementsequenzen unspezifizierter Länge (Kleene-Abschluss) zu ermöglichen. Vor einer Umsetzung derartiger Funktionen müssen allerdings zuerst die genannten grundsätzlichen Beschränkungen aufgehoben werden. Dies wurde und wird unter Beibehaltung der Basisarchitektur von Systematics nach dem hier geschilderten Entwicklungsabschnitt im wesentlichen durch die Einbindung eines kommerziellen objektrelationalen DBMS geleistet.

Danksagung. Die Entwicklung von Systematics wurde im Rahmen des Projekts 'Allgemein-vergleichende Grammatik' (AVG 2.0) von der Deutschen Forschungsgemeinschaft (DFG) gefördert. Der Autor dankt außerdem allen zur Entstehungszeit von Systematics am AVG-Projekt Beteiligten am Institut für Deutsche Philologie und am Institut für Informatik der Universität München für die gute Zusammenarbeit und die hilfreiche Unterstützung, namentlich Dietmar Zaefferer, Ellen Brandner, Heribert Schütz, Gerhard Hingerl, John M. Peterson, Stefan Gering und Irfan Bilgili.

Literatur

[1] H. Schütz, D. Zaefferer. Eine linguistische Wissensbank. Forschungsbericht PMS-FB-1996-17. Institut für Informatik der Universität München, 1996.
http://www.pms.informatik.uni-muenchen.de/publikationen
/PMS-FB/PMS-FB-1996-17.ps.gz

[2] G. Hingerl. Eine deduktive Datenbank für die Linguistik. Diplomarbeit. Institut für Informatik der Universität München, 1996.
http://www.pms.informatik.uni-muenchen.de/
publikationen#DA:Gerhard.Hingerl

[3] M. Nickles. Ein Werkzeug zur interaktiven Bearbeitung von Und-Oder-Bäumen. Projektbericht. Institut für Informatik der Universität München, 1997.
http://www.pms.informatik.uni-muenchen.de/
publikationen#Fopra:Matthias.Nickles

[4] A. Layman et. al. XML Data. W3C Note, 1998.
http://www.w3.org/TR/1998/NOTE-XML-data-0105/

[5] T. Bray, C. Frankston, A. Malhotra. Document Content Description for XML (DCD). W3C Note, 1998.

[6] A. Deutsch et. al. XML-QL: A Query Language for XML. W3C Note, 1998.

[7] XML Schema. http://www.w3.org/XML/Schema

[8] J. Clark, S. Deach. Extensible Stylesheet Language (XSL). W3C Submission, 1998.

[9] J. Robie, J. Lapp, D. Schach. XML Query Language (XQL). http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[10] D. Brickley, R.V. Guha, A. Layman. Resource Description Framework (RDF) Schemas. W3C Submission, 1998.

[11] AVG-Homepage. http://www.cross-ref-grammar.uni-muenchen.de

[12] S. Abiteboul et. al. Tools for Data Translation and Integration. IEEE Data Engineering Bulletin 22(1): 3-8, 1999.

[13] XPath. http://www.w3.org/TR/xpath

[14] R. Goldman, J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Twenty-Third International Conference on Very Large Data Bases, 1997.

[15] M. Fernandez, D. Florescu et. al. Catching the boat with Strudel: experiences with a web-site management system. In SIGMOD Conference, 1998.

[16] M. J. Carey et. al. PESTO: An Integrated Query/Browser for Object Databases. VLDB 1996: 203-214, 1996.

[17] LotusXSL. http://www.alphaworks.ibm.com/tech/LotusXSL

[18] http://www.cross-ref-grammar.uni-muenchen.de/aim.htm

[19] XQuery. http://www.w3.org/TR/xquery/