FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualisierung von Metadaten in Suchergebnissen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Dominik Berchtold
Matrikelnummer 9325753

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber, Univ.Doz.

Wien, 21.09.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Dominik Berchtold
Sobieskiplatz 5, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

An dieser Stelle möchte ich allen danken die es mir ermöglicht haben, diese Diplomarbeit am Ende eines durchaus langen Studiums endlich fertigzustellen. Als erstes möchte ich meinem Betreuer Andreas Rauber danken, der mich auch nach längeren Pausen, die ich während der Erstellung dieser Arbeit immer wieder eingelegt habe, weiterhin unterstützte und wichtige Ideen zur Entwicklung des metaViewers beisteuerte. Besonders erwähnen möchte ich meine Freundin Patrizia Eisele, die es während dieser Zeit sicher nicht leicht mit mir hatte und mir immer mit Rat und Tat zur Seite stand, speziell was universitäre Gepflogenheiten betraf. Außerdem möchte ich meinen Eltern danken. Sie haben viel Geduld bewiesen beim Hoffen auf meinen fast schon nicht mehr zu erwartenden Abschluss. Meine Oma hat immer daran geglaubt dass ich irgendwann doch noch abschließen werde. Ihr möchte ich hier ganz besonders danken, dafür dass sie immer für uns Enkel da ist. Mein Dank gilt auch der *mobilkom austria* und speziell meinem Freund und Vorgesetzten Klaus Hilbe. Er hat es mir ermöglicht meinen Arbeitsplatz zu nutzen zur Erstellung des metaViewers und dieses Dokuments und gewährte mir die dafür benötigte Auszeit. Abschließend möchte ich gerne noch meine Cousine Simone Nachbaur erwähnen. Sie hat mich auf die Idee mit dem Studienabschlussstipendium gebracht, ohne das ich die notwendige Motivation wohl nicht aufgebracht hätte.

Zusammenfassung

Bei Recherchen im Internet werden Suchergebnisse oft als Text in sehr langen und dadurch unübersichtlichen Listen dargestellt (z.B. bei Google, Amazon, Online-Katalogen und -Bibliotheken, ...). Die Aufbereitung mit Hilfe einer graphischen Darstellung wäre in vielen Fällen wünschenswert und könnte den Suchprozess unterstützen. Sollen vor allem die in den Treffern enthaltenen Daten, sogenannte *Metadaten*, sichtbar gemacht werden, bietet sich zur Visualisierung die Verwendung *graphischer Metaphern* an. Damit ist hier die ikonisierte Darstellung von Objekten oder Abbildungen gemeint, deren Aussehen durch den Inhalt von Datenfeldern beeinflusst wird. Die Informationen jedes Treffers definieren jeweils das Aussehen einer Ausprägung der gewählten Metapher. Diese Methode bietet eine Vielfalt an Möglichkeiten, strukturierte Informationen zu den einzelnen Treffern sichtbar zu machen, um eine schnellere Bewertung der Ergebnisse zu erlauben.

Das Finden einer passenden Darstellung, die den Anwender bei seiner Suche auch tatsächlich unterstützt, ist oft schwierig und aufwändig. Deshalb beschränkten sich bisherige Anwendungen meistens auf die Auswahl einer bestimmten Metapher und optimierten die Darstellung, um Daten aus einer konkreten Datenquelle zu visualisieren. Wir wollen diesen Prozess beschleunigen und suchen nach einer Methode, die es ermöglicht, auf einfache Weise verschiedene Metaphern zur Visualisierung unterschiedlichster Datenquellen auszuprobieren.

Die Herausforderung dabei besteht darin, allgemein gültige Gemeinsamkeiten sowohl in möglichen Datenquellen als auch bei der Darstellung verschiedenster Metaphern zu identifizieren. So sollte es möglich sein eine Software zu entwickeln mit der, im Gegensatz zu bisherigen Anwendungen, die direkte Abhängigkeit zwischen Metadaten und Metapher offen gehalten wird.

Aus diesen Überlegungen heraus wird der *metaViewer* entwickelt. Mit seiner Hilfe kann, fast schon spielerisch, eine Vielzahl an Darstellungsmöglichkeiten ausprobiert werden. Er ermöglicht es, die Zuordnung von Metadaten zu den Merkmalen einer graphischen Metapher in vielfältiger Weise zu gestalten. Außerdem ist er derart konzipiert, dass dieser Zuordnungsprozess auf Daten aus verschiedenen Quellen angewendet werden kann und dabei die Verwendung frei zu definierender Metaphern unterstützt wird.

Abstract

Most of today's search engines still provide search results as a list of textual entries. The available information on the internet is increasing rapidly. A growing number of result sets for a given search text can soon become unwieldy. Using some sort of visualization could help the user analyzing the returned results. To make information visible contained in the *metadata* of results, *graphical metaphors* can be used for visualization. For this purpose one icon of a well known object is used to represent each search result. The appearance of the icon is controlled by the metadata contained in the result. This kind of representation is one way of making structural information visible and easier to compare and evaluate.

Finding an appropriate visualization can be a rather challenging task as there are many possible metaphors. Previous publications concentrate on querying a specific data source and try to visualize it using a single graphical metaphor. We want to find an easy way to speed up the process of finding suitable metaphors by developing an application that provides a set of alternative visualizations.

The challenge will be to identify qualities that metadata of all data sources have in common and to find general properties for describing those metaphors. This should be the way to loosen the tight connection between metadata and graphical metaphors.

The *metaViewer* presented in this work shows a new approach to finding the right metaphor in an easy manner. It allows the assignment of metadata fields to properties of the chosen metaphor. The metaViewer can be used for all kinds of different data sources. For visualization some graphical metaphors are ready to use and new ones can easily be defined.

Inhaltsverzeichnis

1 Einführung

Visualisierung von Daten ist nicht erst seit Erfindung von Computern ein sehr lebendiges Feld. So nützlich eine gut gewählte graphische Darstellung zur Vermittlung von Informationen sein kann, so gewissenhaft muss bei der Wahl und dem Entwurf der graphischen Repräsentation vorgegangen werden, um das gewünschte Ziel zu erreichen. Einen richtungsweisenden und sehr lesenswerten Beitrag zu diesem Thema leistete Edward R. Tufte mit [22].

Es existieren etliche Studien (z.B. [2, 29]), die belegen, dass die graphische Aufbereitung und Darstellung von Daten unterschiedlichster Art die Vermittlung der in den Daten enthaltenen Information positiv beeinflussen kann. In vielen Bereichen werden solche Visualisierungen bereits erfolgreich eingesetzt. Gerade deshalb fällt auf, dass sich, entgegen des allgemeinen Trends, für manche Anwendungsfälle in der Darstellung in den letzten 10 Jahren kaum etwas geändert hat. Ein Beispiel dafür bilden Ergebnisse von Suchanfragen aller Art. Egal ob in den gängigsten Suchmaschinen, online Bibliotheken, Warenhäusern, Dateiverzeichnissen, E-Mail Archiven oder anderen Datenquellen gesucht wird, die Ergebnisse werden in den meisten Fällen als textbasierte Listen bereitgestellt. In manchen dieser Services werden einzelne Treffer durch Bilder ergänzt, damit hat sich die graphische Aufbereitung aber bereits wieder erschöpft. Einen kurzen Einblick bietet z.B. [21].

In den letzten Jahren wurden durchaus Versuche unternommen, textbasierte Ergebnislisten graphisch darzustellen. Einige Beispiele dazu werden im nächsten Kapitel vorgestellt. Eines davon ist der libViewer[1] (Abbildung 1.0.1) der die Idee zu dieser Arbeit lieferte und auch die Basis für die Umsetzung bildete. Der im Rahmen dieser Arbeit entwickelte metaViewer greift viele dieser Ideen auf, entwickelt sie weiter und setzt sie in neuer Form um.

Dieses Kapitel beschreibt, wie es zur Entwicklung des metaViewers kam und welche Erfahrungen mit ihm gemacht werden können. Gleich im folgenden Abschnitt wird angesprochen, wie sich aus den anfänglichen Anforderungen weitere Ideen entwickelten, wie diese aussehen und welche Motivation dahinter steckt. Welche Zielsetzungen sich daraus ergaben und was von der Umsetzung erwartet werden kann, wird daran anschließend behandelt. Der letzte Abschnitt dieses Kapitels gibt einen Überblick über die weiteren Kapitel dieser Arbeit.

[1] Im libViewer können Online-Bibliothekssysteme abgefragt und die Ergebnisse als Bücher in einem Bücherregal dargestellt werden. Details zum libViewer finden sich in [3].

(a) Überblicks-Ansicht

(b) Detail-Ansicht

Abbildung 1.0.1: Der libViewer stellt Treffer einer Anfrage an eine Online-Bibliothek als Bücher dar (aus [3]).

1.1 Motivation und Idee

Suchanfragen liefern als Ergebnis Listen von Einträgen. Diese Einträge, oder Treffer beinhalten sogenannte Metadaten (siehe Abschnitt 3.1), diese enthalten in mehreren Datenfeldern Beschreibungen von Eigenschaften des jeweiligen Objekts nach dem gesucht wurde (ein Beispiel zeigt Abbildung 3.1.1). Ein bestimmtes Datenfeld enthält dabei für jeden einzelnen Treffer dieselbe Art von Information, also z.B. den Preis einer Ware oder den Autor eines Buches usw. Diese Art strukturierter Daten kommt sehr häufig vor, wenn eine große Anzahl ähnlicher Objekte verwaltet werden soll. Wir behandeln hier solche Listen zwar weitgehend in digitaler Form, dasselbe Prinzip wurde aber schon aus Zeiten vor der digitalen Revolution angewendet. Karteikarten in einer Bibliothek z.B., bedienten sich bereits derselben Strukturen zur Verwaltung von Informationen zu den Werken in ihrem Bestand.

Dass Suchergebnisse in dieser Form präsentiert werden ist nicht verwunderlich, werden sie meistens aus Anfragen an Datenbanken generiert, welche genau diese Art von Informationen enthalten und bei Abfragen auch so zurück liefern. Aber nicht nur bei Suchanfragen oder in Datenbanken treffen wir auf solche Strukturen. Auch das Dateisystem von Computern kann als Liste von Einträgen aufgefasst werden, die in Datenfeldern weitere Informationen (z.B. Dateigröße oder Erstellungsdatum) enthalten. Die Anzahl und der Inhalt der verwendeten Datenfelder mögen sich von Anwendung zu Anwendung ändern, das Prinzip dahinter bleibt dasselbe.

Viele wissenschaftliche Arbeiten, aber auch unzählige Tools, zielen darauf ab, solche strukturierten Daten anschaulicher darzustellen und greifen dafür auf unterschiedlichste Methoden zur graphischen Darstellung zurück. Speziell Softwaretools, aber auch die meisten wissenschaftlichen Arbeiten, beziehen sich dabei auf eine ganz konkrete Anwendung und versuchen für diesen speziellen Fall eine optimale Darstellung der Informationen zu erreichen. Daraus entstanden sehr nützliche Anwendungen wie z.B. WinDirStat (siehe Abbildung 1.1.1), oder man erlangte wertvolle Erkenntnisse, z.B. darüber, wie gut eine gewählte Art der graphischen Repräsentation für ganz bestimmte Daten die Informationsaufnahme beim Betrachter unterstützt (etwa in [20]).

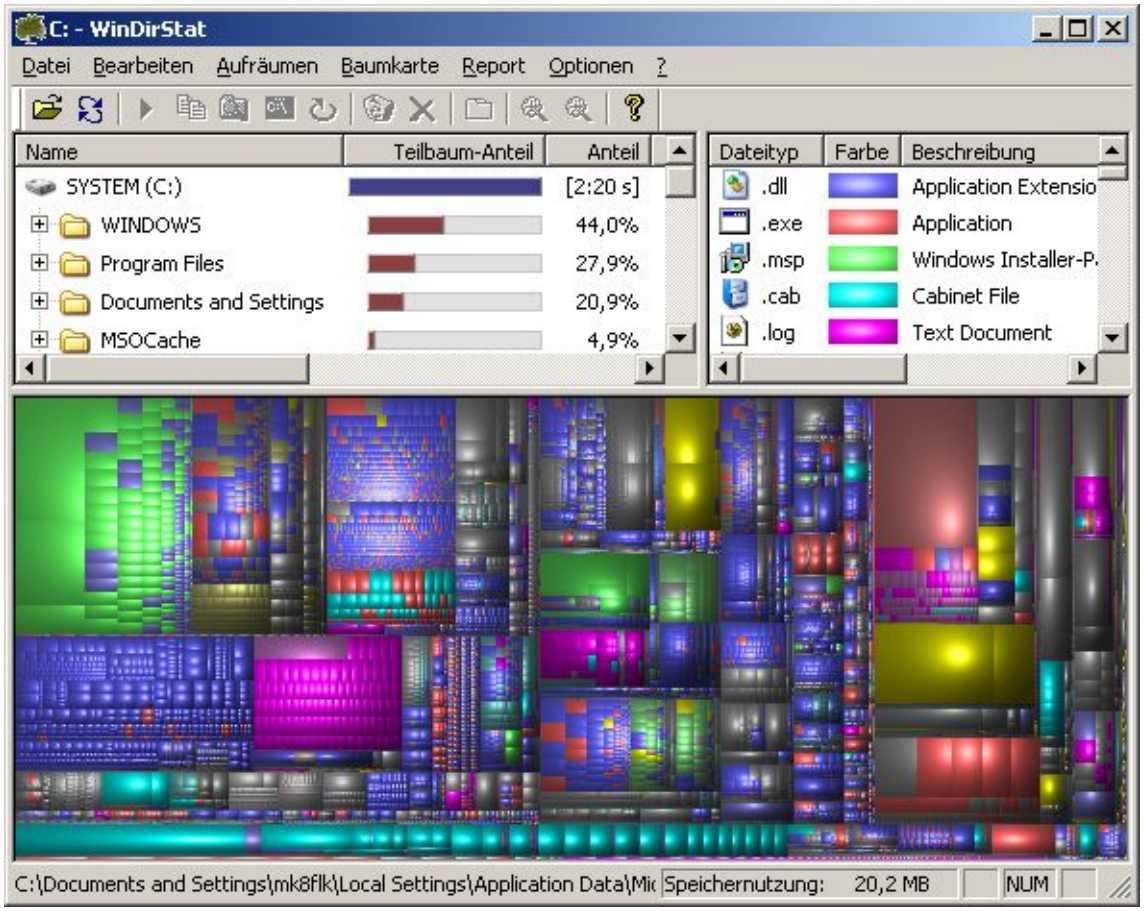


Abbildung 1.1.1: Das Tool WinDirStat gibt einen graphischen Überblick über Dateien auf einem Datenträger und vermittelt einen Eindruck ihrer Größe und ihrer Aufteilung auf Verzeichnisse (http://windirstat.info/).

Untersuchungen an konkreten Beispielen sind sehr wichtig, helfen sie uns zu verstehen, wie Menschen Daten aufnehmen und verarbeiten. Diese Vorgehensweise ist

jedoch aufwändig und behandelt stets nur einen eng begrenzten Anwendungsfall. Da es, wie oben erwähnt, eine Vielzahl an in Frage kommenden Datenquellen gibt, und den Möglichkeiten der graphischen Darstellung kaum Grenzen gesetzt sind, ergeben sich daraus eine nicht überschaubare Anzahl an Kombinationsmöglichkeiten. Diese alle genauer zu untersuchen und die jeweils beste, oder auch nur eine sehr gute auszuwählen, scheint kaum möglich. Das könnte auch der Grund dafür sein, dass z.B. bei Suchergebnissen eine graphische Aufbereitung so sparsam eingesetzt wird. Es gibt keine allgemein gültigen Richtlinien, welche Darstellung gewählt werden und wie die Zuordnung im Einzelnen erfolgen soll.

Die Idee, nach alternativen Möglichkeiten zu suchen, eine nützliche Darstellungsform zu finden, entwickelte sich aus der ursprünglichen Anforderung, den Weg, der bei der Entwicklung des libViewer [3] eingeschlagen wurde, weiter zu verfolgen. Im libViewer können Suchergebnisse aus Online-Bibliothekskatalogen als Bücher in einem Regal dargestellt werden (siehe Abbildung 1.0.1). Dabei werden Metainformationen aus den Ergebnissen dazu benutzt, die Gestalt der Bücher zu beeinflussen, um damit dem Anwender die Suche zu erleichtern. Aus der Motivation heraus, die Darstellung und Bedienung des libViewer technisch auf einen aktuelleren Stand zu bringen, wurde die Idee geboren, auch das Anwendungsgebiet zu erweitern. Im metaViewer sollen sowohl Datenquellen als auch Darstellungsformen unabhängig voneinander erweitert, ausgetauscht und miteinander kombiniert werden können.

1.2 Zielsetzung und Beitrag der Arbeit

Wenn im Weiteren von Online Suchen oder Ergebnislisten solcher Suchen die Rede ist, bezieht sich das nicht nur auf Suchmaschinen wie Google, Yahoo, Bing und dergleichen, es sollen dabei auch z.B. Seiten zur Suche nach Produkten wie Amazon oder eBay eingeschlossen werden, als auch alle Arten von Onlinebibliotheken, Literaturverzeichnissen oder Ähnlichem. Relevant für die vorliegende Arbeit ist nur, dass in den Ergebnissen Metadaten enthalten sind, in denen Attribute (siehe 3.2) identifiziert werden können, deren Ausprägungen Aussagen über die einzelnen Treffer zulassen.

Die Benutzeroberfläche des libViewer [3] entspricht nicht mehr dem aktuellen Stand der Technik. Die Ideen die zu seiner Entwicklung führten, haben jedoch nichts an Aktualität eingebüßt und sollen mit dieser Arbeit weiter verfolgt werden. Wir entschlossen uns, den metaViewer von Grund auf neu zu implementieren, um nicht nur die Oberfläche sondern auch die verwendete Programmiersprache und die eingesetzten Softwarepakete zu aktualisieren. Einige der im metaViewer umgesetzten Funktionen sind z.B.: eine beliebig zoom-bare graphische Darstellung, Verwaltung in mehreren Fenstern, ein Überblicksfenster zur leichteren Navigation innerhalb der dargestellten Ergebnisse, intuitive Steuerung der Applikation und Interaktionsmög-

lichkeiten mit den graphisch dargestellten Suchergebnissen. Näheres dazu findet sich in den Kapiteln 4 und 5.

Während der Definition und des Entwurfs der angestrebten Verbesserungen zur Bedienung und Darstellung, wurde die Idee geboren, auch die grundlegende Funktionalität gegenüber dem libViewer zu erweitern. So sollte auf der „Eingangsseite" die Wahl der Datenquelle nicht auf Suchergebnisse von Onlinebibliotheken beschränkt bleiben, sondern auf alle Arten von mit Metadaten behafteten Listen erweitert werden können. Außerdem sollte die Darstellung der Ergebnisse als Bücher nur eine mögliche Variante darstellen. Die Implementierung anderer graphischer Metaphern (siehe 3.3) als Darstellungsform sollte jedoch möglich sein, ohne allzu große Anpassungen an den restlichen Teilen der Applikation vornehmen zu müssen. Daraus ergab sich die Notwendigkeit, die Zuordnung von Attributen aus den Metadaten zu Metapher-Merkmalen (siehe dazu Kapitel 3) flexibel und allgemein gültig zu halten. Diese neuen Funktionen wurden bei der Entwicklung des metaViewers umgesetzt. Damit eröffnen sich völlig neue Möglichkeiten zur Verwendung und zur Verwirklichung weiterer Ideen!

Die Methode, graphische Metaphern zur Visualisierung strukturierter Daten zu verwenden, ist keinesfalls neu (siehe z.B. [6]). Alle von uns untersuchten Arbeiten haben jedoch gemein, dass die Zuordnung der in den Ergebnislisten identifizierten Attribute zu einzelnen Merkmalen von graphischen Metaphern relativ statisch erfolgt. Damit wird ein "Ausprobieren" der besten Zuordnung nicht oder nur in geringem Maße unterstützt. Auch das Hinzufügen weiterer Metaphern oder der Wechsel zu gänzlich anderen Darstellungsformen wird kaum ermöglicht. Bisherige Arbeiten untersuchten meistens die Möglichkeiten der graphischen Darstellung einer ganz bestimmten Quelle von Daten, z.B. Ergebnisse einer Suchmaschine oder von Referenzen aus Onlinebibliotheken und dergleichen. Uns interessierte es, die Schnittstelle zwischen Metadaten-Feldern der Suchergebnisse und den graphischen Metaphern so flexibel zu gestalten, dass unterschiedliche Suchquellen mit verschiedenen Visualisierungen kombiniert und deren jeweilige Aussagekraft überprüft werden kann (Abbildung 1.2.1).

Wie schon im vorigen Abschnitt erwähnt, scheint es noch etwas länger zu dauern, bis auf allgemein gültige Richtlinien für graphische Darstellungen von strukturierten Daten zurückgegriffen werden kann. Deshalb versuchen wir in dieser Arbeit einen anderen Weg zu gehen. Wenn der exakte, gewissenhafte Weg nicht zum gewünschten Ergebnis führt, kann evtl. simples Ausprobieren und "Herumspielen" mit unterschiedlichen Daten, Darstellungsformen und Zuordnungen zu brauchbaren Resultaten führen. Der metaViewer soll einen Rahmen bilden, um solche Versuche durchführen zu können.

Die Herausforderung bei herkömmlichen Untersuchungen war, für ganz spezielle Daten eine gewählte Art der Darstellung zu optimieren und zu testen. Mit dem

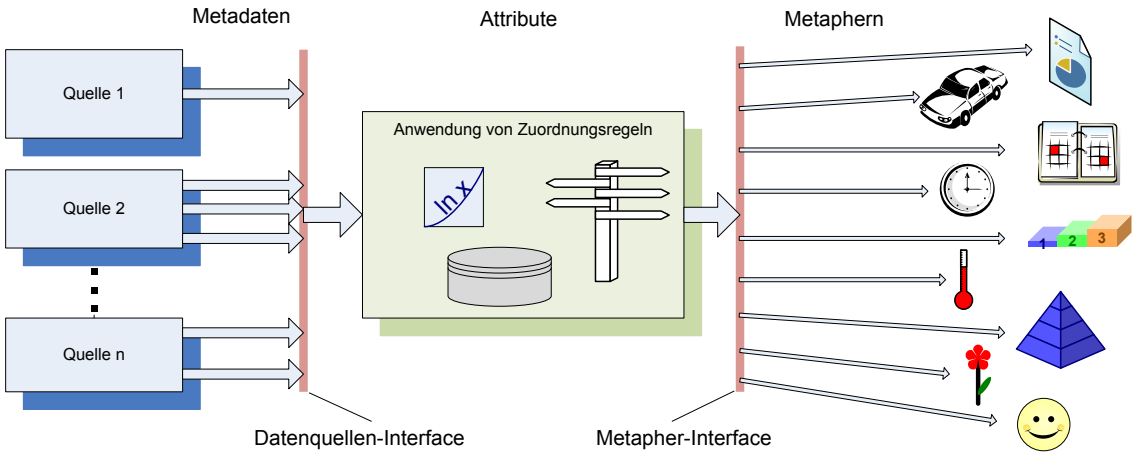


Abbildung 1.2.1: Der metaViewer ermöglicht die flexible Zuordnung von Metadaten aus verschiedenen Datenquellen zu den Merkmalen unterschiedlicher Metaphern. Die klare Definition der Schnittstellen ist dabei besonders wichtig.

metaViewer kann das Problem von der anderen Seite her angegangen werden. Mit seiner Hilfe können, für alle Arten von Daten[2] und möglichst unabhängig von deren Inhalt, unterschiedlichste graphische Darstellungen ausprobiert werden. Und nicht nur eine freiere Wahl der Darstellung wird ermöglicht, auch mit den Zuordnungen zwischen Daten und Graphik kann in einfacher Weise herum gespielt werden. Damit kann ein Gefühl dafür geschaffen werden, bei welchen Attributen eine graphische Darstellung von Nutzen ist und welche graphischen Metaphern dafür verwendet werden können.

1.3 Aufbau und Kapitelübersicht

Zunächst werden in Kapitel 2 Arbeiten vorgestellt, die sich mit der Aufbereitung und Präsentation von strukturierten Daten befassen. Die vorgestellten Verfahren werden dabei an Hand ihrer Herangehensweise unterteilt, mit der sie versuchen, die Darstellung von Suchergebnissen zu verbessern. Anschließend werden die unterschiedlichen Methoden verglichen und ihre Vor- und Nachteile identifiziert. Abschließend wird darauf eingegangen, welche dieser Verfahren im libViewer und im metaViewer eingesetzt werden.

In Kapitel 3 werden die Begriffe, *Metadaten, Attribut, graphische Metapher* und

[2]Voraussetzung ist allerdings, dass aussagekräftige Metadaten mit der beschriebenen Struktur darin enthalten sind!

Metapher-Merkmal behandelt, so wie sie in dieser Arbeit verwendet werden. Dann wird aufgezeigt, was bei der Identifizierung unterschiedlicher Typen von Attributen und Metaphern eine Rolle spielt. Anschließend wird betrachtet, was bei der Zuordnung von Attributen zu Metaphern beachtet werden muss.

Kapitel 4 beschreibt die Herangehensweise an die gestellten Aufgaben, aber auch wie sich durch die nähere Beschäftigung weitere Ideen ergeben haben. Dem Weg der Daten durch den metaViewer folgend, werden die einzelnen Aufgaben beschrieben, die bei der Erstellung bewältigt werden mussten. Angefangen mit der Einteilung der Quelldaten in Attributs-Typen, über die Zuordnung zu Metapher-Merkmalen bis zur graphischen Darstellung. Abschließend wird ein Überblick über die Bedienmöglichkeiten im metaViewer gegeben.

In Kapitel 5 wird ein detaillierterer Einblick in die Architektur des metaViewers gegeben. Nach einer kurzen Aufzählung der Programmierumgebung und der verwendeten Programmpakete, folgen eine Beschreibung der wichtigsten Teile des metaViewers sowie die Darstellung der Klassen-Struktur seiner zentralen Komponenten. Die abschließende Dokumentation der Programmierschnittstellen soll die Möglichkeiten zukünftiger Erweiterungen aufzeigen und Unterstützung bei deren Implementierung bieten.

Kapitel 6 fasst noch einmal zusammen, was die Ergebnisse dieser Arbeit sind. Dabei werden auch einige der Erfahrungen beschrieben, die während der Entwicklung und bei ersten Anwendungen des metaViewers gemacht wurden. Mögliche Ergänzungen und Erweiterungen die im Rahmen dieser Arbeit nicht mehr umgesetzt werden konnten und ein paar neue Ideen die evtl. in zukünftigen Arbeiten realisiert werden könnten beschließen dieses letzte Kapitel.

2 Stand der Technik

Dieses Kapitel soll einen Einblick in die Vielzahl der Verfahren geben, die bereits angewendet werden um Informationen die in Form von textbasierter Listen vorliegen, zur besseren Verwendbarkeit aufzubereiten. Wir beschränken uns im Folgenden auf die Präsentation von Ergebnissen aus Online-Suchanfragen, jedoch lassen sich viele Erkenntnisse die bei der visuellen Analyse von Daten in der Vergangenheit gewonnen werden konnten auch auf diesen spezielleren Fall anwenden.

Dabei werden wir zuerst versuchen, einen Überblick über Verfahren zu geben, die bereits zur Aufbereitung von Suchergebnissen zur Anwendung kommen (einen sehr guten Überblick bietet z.B. auch [10]). Danach folgen ein Vergleich der verschiedenen Herangehensweisen und ein Blick darauf, wie sich der libViewer und der metaViewer dabei einordnen lassen bzw. welche der Verfahren in dieser Arbeit zur Anwendung kommen.

2.1 Verfahren zur Aufbereitung von Suchergebnissen

Auch heute noch wird das Ergebnis bei den meisten Online Suchen in Form von textbasierten Listen bereitgestellt. Allerdings wurde in den letzten Jahren mit unterschiedlichen Herangehensweisen versucht, die in den Suchergebnissen enthaltenen Informationen effizienter zur Verfügung zu stellen. Eine ausführliche Behandlung von Strategien, die zur Verbesserung des Suchprozesses aktuell verfolgt und teilweise bereits zur Anwendung kommen, wird in [25] behandelt.

Betrachtet man diese Arbeiten genauer, kann man die Methoden, mit denen versucht wurde die Präsentation der Ergebnisse zu verbessern, in mehrere Klassen unterteilen (in ähnlicher Form behandelt auch in [21]).

2.1.1 Umgang mit (sehr) vielen Treffern

Ein Problem auf das man bei der Recherche im Internet oft stößt, ist die schiere Anzahl an Treffern die eine Suchanfrage liefert. Gängige Suchmaschinen ordnen die Ergebnisse und geben sie als Liste aus, bei Bedarf aufgeteilt auf mehrere Seiten. Obwohl die Anzahl der gefundenen Ergebnisse meist sehr groß ist, werden vom Anwender selten mehr als die ersten paar Seiten durchsucht. Er verlässt sich somit

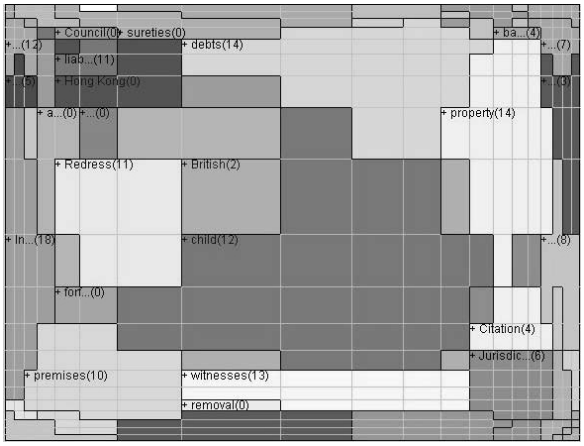
darauf, dass die angewandte Reihung ihren Zweck erfüllt und die relevanten Treffer zuoberst gereiht sind. Auch wenn so das Gesuchte in vielen Fällen gefunden werden kann, werden in anderen Fällen relevante Ergebnisse gar nicht erst in Betracht gezogen. Zusätzlich können Mehrdeutigkeiten der verwendeten Suchbegriffe zu vielen, weit vorne gereihten Treffern führen, die für das eigentlich gesuchte Thema völlig uninteressant sind. Um dem Abhilfe zu schaffen, werden z.B. Methoden eingesetzt, die dem Anwender eine größere Anzahl an Ergebnissen auf einen Blick präsentieren. Dies soll es ihm ermöglichen, Strukturen zu erkennen und diese für die weitere Suche zu nutzen. Die Liste der Treffer muss dafür erst in irgendeiner Form strukturiert und anschließend in anschaulicher Weise dargestellt werden. Die dabei eingesetzten Verfahren sind vielfältig, besonders die Suche nach einer geeigneten Darstellungsform bietet ein weites Betätigungsfeld.

Um in den Treffern einer Suche Strukturen zu bilden, wird ein Maß für die Ähnlichkeit von Dokumenten definiert und mit dessen Hilfe Cluster gebildet. Kriterien die dabei beachtet werden müssen sind z.B. die Definition eines aussagekräftigen Ähnlichkeitsmaßes sowie die Reduktion des multidimensionalen Suchraums auf wenige Dimensionen. Letzteres ist notwendig, um eine zufriedenstellende Repräsentation auf einem zweidimensionalen Bildschirm zu ermöglichen bzw. für den Menschen erfassbar zu machen, dessen kognitive Prozesse für die Betrachtung von maximal drei Dimensionen optimiert sind. In [27] werden z.B. *SOM*[1] verwendet, um die gefundenen Cluster in einer Landkarten-ähnlichen Form abzubilden. Bezüglich des gesuchten Themas als relevanter eingeschätzte Cluster werden dabei mehr in der Mitte, weniger relevante weiter am Rand der Anzeige platziert. Zusätzlich kommt auch eine für diesen Zweck häufig genutzte Methode genannt *Fisheye View* zum Einsatz, bei der die Darstellung, wie bei der Betrachtung durch eine Fischaugen-Linse, verzerrt wird, sodass Gebiete gestaucht werden und zwar umso mehr, je weiter sie vom Zentrum entfernt liegen (siehe Abbildung 2.1.1a). Andere Verfahren bilden die Cluster in hierarchischen Strukturen ab (z.B. in [28, 1, 20], Abbildungen 2.1.1c und 2.1.3a) oder legen spezielles Augenmerk auf das Verhältnis der Cluster zueinander ([18], Abbildung 2.1.1d). Methoden um sehr viele Datenpunkte (in der Größenordnung von Milliarden, z.B. bei Datenbankabfragen oder wie auch in Abbildung 1.1.1) darzustellen, werden etwa in [19] betrachtet (Abbildung 2.1.2).

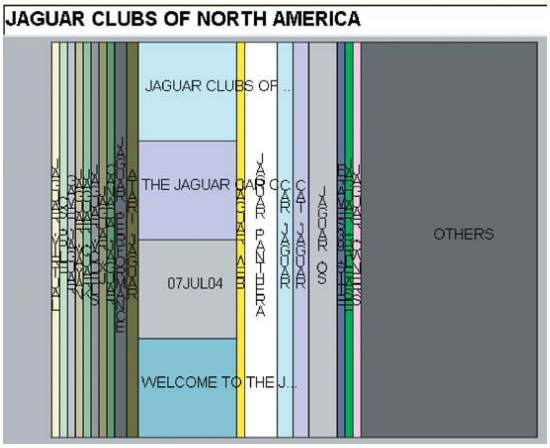
Viele der genannten Beispiele stellen die Daten nicht nur statisch dar, sondern bieten Interaktionsmöglichkeiten um es dem Anwender zu ermöglichen, sich in dem Suchraum zu bewegen und jene Bereiche hervorzuheben, die für ihn von Interesse sind (z.B. auch in [9, 20] und Abbildung 2.1.3).

Wenn mehrere Ergebnisse zu einem Cluster zusammengefasst werden, stellt sich die Frage, wie dieses Cluster am besten am Bildschirm repräsentiert wird. Da so

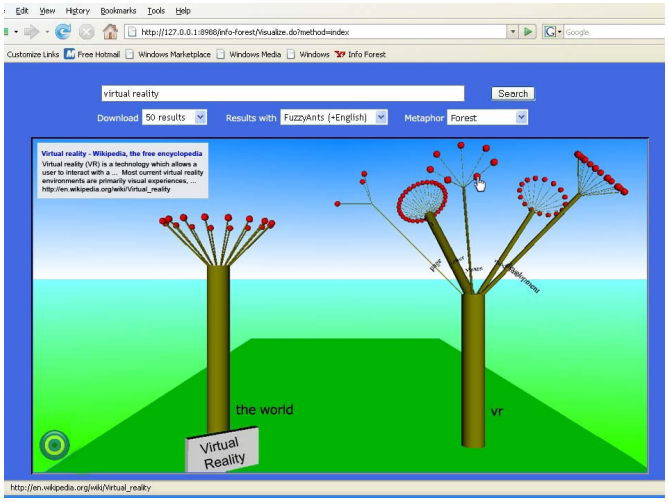
[1] Self-Organizing Maps (SOM), aus [12], verwenden neuronale Netzwerke um Cluster in einer Sammlung von Dokumenten zu bilden.



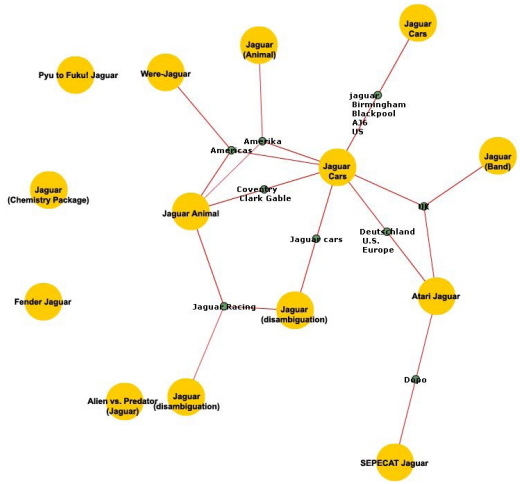
(a) Fisheye View von identifizierten Clustern aus [27]. Die Stauchung erfolgt relativ zur ermittelten Relevanz eines Clusters.



(b) Fisheye View aus [23] hebt einen gewählten Bereich hervor und bietet trotzdem einen Überblick über alle Ergebnisse. Der Grad der Stauchung hat hier keine weitere Aussagekraft.

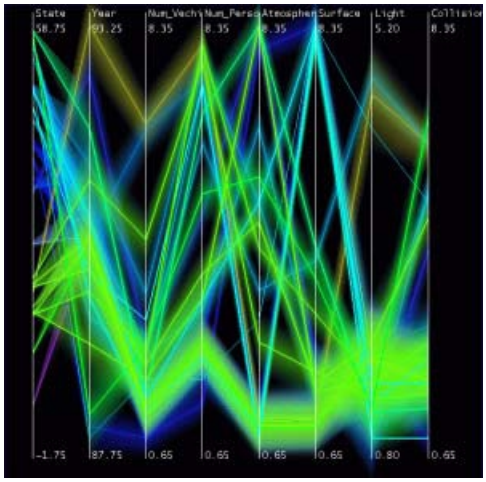


(c) Hierarchische Darstellung von Clustern mit Bäumen aus [1]

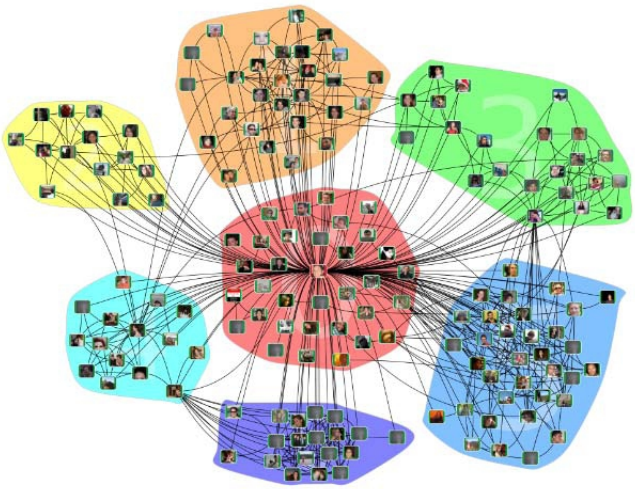


(d) Cluster mit eingezeichneten Beziehungen zueinander aus [18].

Abbildung 2.1.1: Unterschiedliche Darstellungen von Ergebnissen nach einer Cluster-Analyse

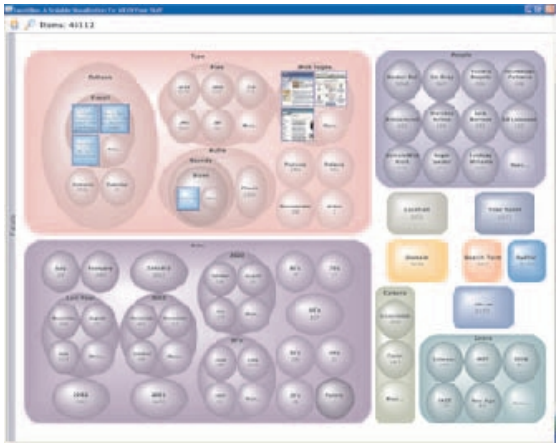


(a) Anzeige einer Datenbankabfrage mit Hilfe von Parallel Coordinates

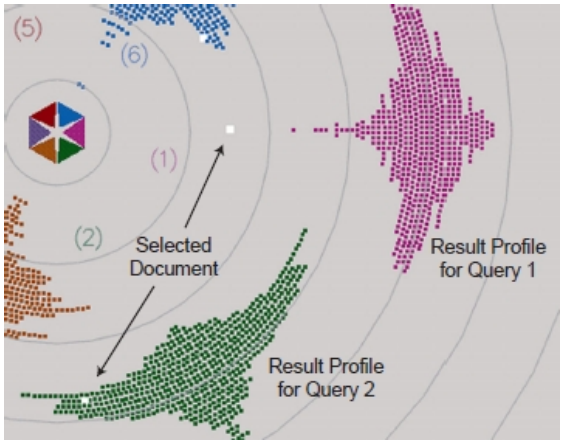


(b) Darstellung von Community-Strukturen eines Social Networks

Abbildung 2.1.2: Darstellung einer sehr großen Anzahl von Ergebnissen aus [19]



(a) FaceMap aus [20] stellt Cluster hierarchisch dar und kann interaktiv bedient werden.



(b) Darstellung von Dokumenten mehrerer Suchabfragen [9]. Der Abstand zum Zentrum zeigt die jeweilige Relevanz für die Abfrage.

Abbildung 2.1.3: Cluster-Darstellungen mit Interaktionsmöglichkeiten.

ein Cluster nun mehrere Ergebnisse repräsentiert, ist es nicht mehr so einfach, eine passende Darstellung oder Bezeichnung zu finden. In [28] wird z.B. untersucht, wie eine aussagekräftige Bezeichnung für einen Cluster aus mehreren Dokumenten gefunden werden kann. In [26] wird zusätzlich versucht, zu den Clustern, die auf Basis der Themenzugehörigkeit gebildet werden, ein passendes Bild aus den Ergebnissen darzustellen, um das schnelle Erfassen der einzelnen Themen zu erleichtern.

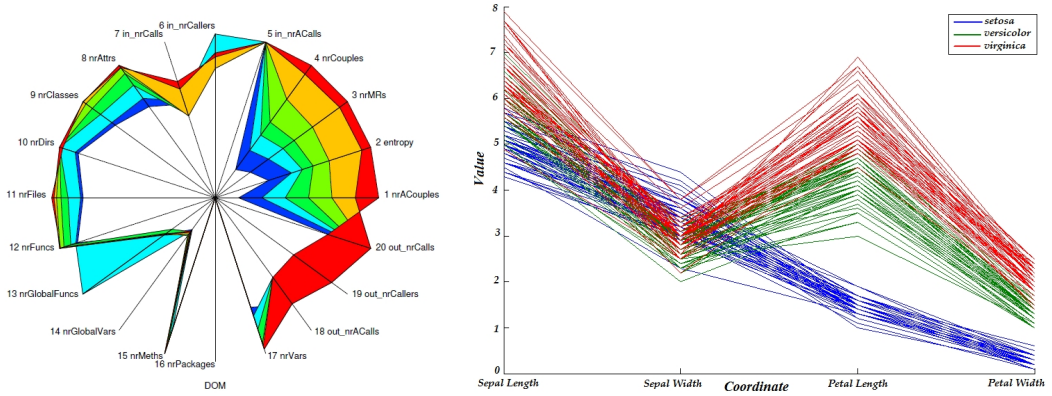
2.1.2 Verwendung und Darstellung von Metadaten

Nicht nur die Bildung von Clustern und damit die komprimierte Darstellung der Ergebnisse kann das Auffinden relevanter Informationen unterstützen, auch die Wahl einer geeigneten Repräsentation der einzelnen Treffer selbst kann die Suche entscheidend erleichtern.

Eine aussagekräftige Abbildung liefert schon wichtige Informationen, die auf einen Blick wahrgenommen werden können. Besonders bei der Auflistung von Produkten werden solche Bildinformationen bereits häufig beigefügt (z.B. Amazon[2] oder EBay[3]) und erleichtern besonders die intuitive Auswahl relevanter Treffer.

Zusätzlich zur Bildinformation wäre jedoch oft die Visualisierung einzelner Eigenschaften (z.B. eine Größe, ein Preis oder die Zugehörigkeit zu einer Kategorie) der Treffer wünschenswert. Dabei spielt die Auswahl der Informationen, welche in die Darstellung einfließen, eine entscheidende Rolle. Außer der Reihung nach *Relevanz*[4], die bereits durch den Suchprozess den Ergebnissen zugeordnet wird, sind für den Anwender oft andere Eigenschaften der Ergebnisse interessant, wie z.B. technische Spezifikationen, der Preis, ein Herstellungsdatum oder eine Bewertung anderer Nutzer. Gerade solche Bewertungssysteme werden häufig eingesetzt. Dabei können User die einzelnen Einträge auf einfache Art, z.B. mit einem Notenschlüssel oder Sternen, bewerten. Jedem Eintrag kann somit ein Durchschnitt dieser Bewertungen als ein subjektives Qualitätsmaß zugeordnet werden. Solche Systeme finden sich mittlerweile auf den unterschiedlichsten Seiten, so z.B. auf Produktsuchseiten bzw. Online-Versandhäusern (Geizhals[5], Amazon[6], ...), Suchseiten für Restaurants mit Zustellservice (Lieferando[7], Netkellner[8], Willessen[9], ...), Musik- Film- und Video-

[2] http://www.amazon.de/
[3] http://www.ebay.at/
[4] Die Relevanz kann dabei ein Maß für die Übereinstimmung mit den Suchbegriffen sein, oft fließt aber auch eine Beurteilung der Wichtigkeit insgesamt mit ein. Diese wird meistens mit Hilfe von Heuristiken selbst ermittelt, z.B. in [24].
[5] http://geizhals.at/
[6] http://www.amazon.de/
[7] http://www.lieferando.at/
[8] http://www.netkellner.at/
[9] http://www.willessen.at/



(a) Dieses Kiviat Diagramm stellt verschiedene Evolutionsstufen des Quellcodes eines Softwarepaketes dar [11].

(b) Dieses Parallel Coordinates Diagramm zeigt einen oft verwendeten Beispiel-Datensatz mit Messdaten von Iris Blumen.[10]

Abbildung 2.1.4: Koordinaten-Darstellungen

plattformen (iTunes[10], IMDB[11], YouTube[12], ...) u.v.a. Meistens wird die Bewertung mit Hilfe von Sternen oder Balken direkt bei den Treffern dargestellt, aber auch die Verwendung von Farben und Zahlen wurde schon untersucht [8].

Die Relevanz eines Suchergebnisses oder die Bewertung durch Anwender haben eines gemeinsam, sie sagen etwas über den einzelnen Treffer aus und das in strukturierter Form. Bei vielen Arten von Suchen enthalten die Ergebnisse, abgesehen von ihrer Bezeichnung in Textueller Form, mehrere solcher Felder strukturierter Informationen die manche Eigenschaften der einzelnen Treffer näher spezifizieren (z.B. Publikations- bzw. Erzeugungsjahr, Autor, Produzent, Abmessungen, Bildinformationen, ...). Diese Informationen, wir bezeichnen sie allgemein als *Metadaten*, können dem beschreibenden Text hinzugefügt werden, aber auch direkt als Basis zur graphischen Darstellung genutzt werden. Die Schwierigkeit besteht dabei darin, diese multidimensionalen Daten so darzustellen, dass möglichst viele Informationen aus ihnen abgelesen werden können, ohne die Darstellung zu überladen. Zwei Möglichkeiten wie an diese Aufgabe herangegangen werden kann, werden im Folgenden untersucht.

Einige Möglichkeiten um solche multidimensionalen Daten darzustellen stammen aus den Bereichen *Data Mining* bzw. allgemein aus dem Gebiet der *Analyse und Visualisierung* großer Datenmengen z.B. aus Datenbanken und Datenwürfeln. Eine

[10] http://www.apple.com/at/itunes/
[11] http://www.imdb.de/
[12] http://www.youtube.com/

häufig gewählte Methode sind Kiviat-Diagramme (oder laut Wikipedia[13] auch Netz-, Spinnennetz-, Radar-, Stern-Diagramme genannt). Dabei werden die Werte mehrerer ausgewählter Dimensionen auf kreisförmig angeordneten Achsen aufgetragen (siehe Abbildung 2.1.4a). Diese Methode eignet sich gut, um mehrere Datensätze (einzelne Treffer oder Kategorien) anhand der gewählten Dimensionen auf einen Blick zu vergleichen. Eine Behandlung dieser und ähnlicher radialer Methoden findet sich z.B. in [7, 14]. Eine konkrete Anwendung wird z.B. in [15] gezeigt. Hier werden die Veränderungen des Quellcodes unterschiedlicher Versionen einer Software mit Hilfe eines Kiviat Diagramms dargestellt. In [11] wird versucht, dasselbe Diagramm dreidimensional aufzufächern, um Regionen sichtbar zu machen, die in der 2D-Version verdeckte sind.

In einer anderen Variante von Diagrammen werden die Dimensionen nicht kreisförmig sondern mit gleichbleibenden Abständen auf einer Achse aufgetragen um so einen Vergleich mehrerer Datensätze zu ermöglichen. In solch einem *Parallel Coordinates* genannten Diagramm lassen sich besonders viele Datensätze gleichzeitig darstellen und sich somit ein Eindruck über die Verteilung der Datensätze vermitteln (siehe [13, 10, 19] sowie Abbildungen 2.1.2a und 2.1.4b).

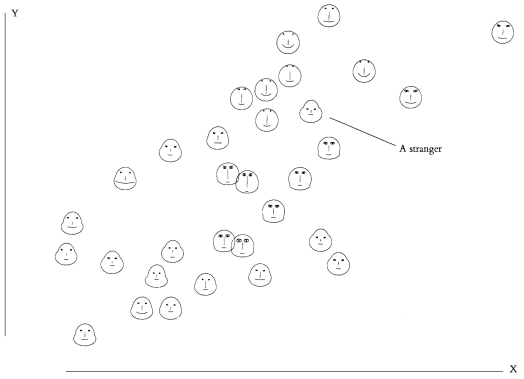
Die zuletzt genannten Darstellungsmethoden haben gemein, dass sie vom Betrachter verlangen, sich mit der Darstellung, im Speziellen mit der Wahl und Anordnung der Dimensionen, vertraut zu machen um sie effizient nutzen zu können. Deshalb werden diese Methoden auch oft eingesetzt, um wissenschaftliche Daten zu vermitteln oder um fachspezifische Auswertungen durchzuführen, meist von Personen, die mit der zugrunde liegenden Art an Information bereits vertraut sind. Bei der Suche im Internet kann jedoch meistens nicht davon ausgegangen werden, dass der Anwender sich intensiver mit der angebotenen Darstellung auseinandersetzt. Die Erfassung der dargebotenen Informationen sollte intuitiv und ohne längere Erklärungen erfolgen. Um dies zu erreichen, kann die Verwendung von Abbildungen alltäglicher Gegenständen bzw. für Menschen allgemein anschaulicher Repräsentationen entscheidend beitragen. Der Einsatz solcher *graphischer Metaphern* bietet vielfältige Möglichkeiten, einige davon sollen im nächsten Abschnitt behandelt werden.

2.1.3 Visualisierung mit graphischen Metaphern

Informationen aus Datenfeldern die in den einzelnen Einträgen enthalten sind, sogenannte Metadaten (siehe Abschnitt 3.1), können zur Aufbereitung der Suchergebnisse genutzt werden, so wie das auch die bereits genannten Verfahren machen.

Werden Daten (meist mehrdimensionale) mit Hilfe der bildlichen Darstellung von Objekten präsentiert und dabei teilweise oder ganz auf die textuelle Darstellung verzichtet, sprechen wir bei dem Objekt von einer *graphischen Metapher*. Dabei werden

[13] *http://de.wikipedia.org/wiki/Netzdiagramm*



(a) Chernoff faces in einem Diagramm aus [22]. Ein Ausreißer fällt durch seine Kopfform auf, die erkennbar von den Gesichtern in seiner Umgebung abweicht.

(b) Beispiel einer Gebäude-Metapher, Dargestellt in einer Stadt-Simulation aus [4].

(c) Eine Blumen-Metapher aus [5]. Links eine Erläuterung der einzelnen Merkmale, rechts zwei dargestellte Treffer als Beispiel.

Abbildung 2.1.5: Anwendungsbeispiele von Metapher-basierten Darstellungen

Informationen aus den Metadaten dazu benutzt, einzelne Merkmale der Metapher zu verändern bzw. erst zu definieren.

Diese Art der Repräsentation bietet eine Vielzahl an Variationsmöglichkeiten, einige davon wurden bereits in unterschiedlichen Anwendungen untersucht. Oft werden in diesem Zusammenhang *Chernoff faces* als richtungsweisendes Beispiel genannt [6]. Diese von *Herman Chernoff* vorgestellte Methode wurde so bekannt, da sie einen für die damalige Zeit relativ kontroversen Ansatz verfolgte. In seiner Darstellung beeinflussen unterschiedliche Variablen jeweils eine Eigenschaft eines skizzierten Gesichtes. So werden u. A. die Form des Gesichtes, die Länge der Nase oder die Krümmung des Mundes entsprechend variiert. Bei einer Anordnung der resultierenden Gesichter in einem zweidimensionalen Diagramm, lassen sich damit z.B. Ausreißer intuitiv erkennen, als Gesichter die „nicht zu ihrer Umgebung" passen (siehe Abbildung 2.1.5a).

Bereits in diesem frühen Beispiel lassen sich viele der grundlegenden Eigenschaften Metapher-basierter Darstellungen ablesen:

- Graphische Darstellung numerischer Daten
- Verwendung alltäglicher bzw. bekannter Objekte als Metapher
- Ikonisierte Darstellung
- Variation von intuitiv erfassbaren Merkmalen der Metapher durch Werte aus den Metadaten
- Informationen mehrerer Dimensionen werden in einer kompakten Darstellung untergebracht

Eine etwas aktuellere Anwendung, die graphischen Metaphern zur Visualisierung verwendet, wird z.B. in [4] beschrieben. Hier werden Webseiten, z.B. die Ergebnisse einer Online-Suche, als Fassaden von Gebäuden dargestellt. Die Gebäude wiederum werden in einer dreidimensionalen Simulation einer Stadt platziert. Die Einteilung der Gebäude in Häuserblöcke erfolgt auf Grund einer Klassifikation die mit Hilfe eines SOM Algorithmus ermittelt wurde. Die Höhe der Gebäude wird durch die Relevanz des Treffers bestimmt (siehe Abbildung 2.1.5b).

In [5] werden Blumen als Metapher verwendet, um rein textuelle Suchergebnisse mit bildlichen Informationen zu ergänzen. Dabei wird z.B. die Relevanz der einzelnen Suchbegriffe für einen Treffer mit unterschiedlich eingefärbten Blütenblättern repräsentiert, die Länge des Stängels ist ein Maß für die Dokumentlänge und in den Blättern und im Fuß der Blume wird die Anzahl von Links auf sowie Verweise zu der jeweiligen Seite dargestellt (Abbildung 2.1.5c).

Das Sprichwort „Ein Bild sagt mehr als tausend Worte (oder Zahlen)" soll auch hier strapaziert werden, schließlich dreht sich in dieser Arbeit alles um graphische Darstellung. Jede Anreicherung rein textueller Informationen mit Bildern kann den Prozess der Informationsaufnahme erleichtern. Nicht immer steht dafür jedoch eine passende Abbildung zur Verfügung. Die Verwendung graphischer Metaphern macht es möglich, auch solche Informationen graphisch aufzubereiten, für die sonst kein aussagekräftiges Bildmaterial gefunden werden kann.

Ein weiterer Vorteil Metapher-basierter Darstellungen gegenüber z.B. parallel Coordinates oder Kiviat-Diagrammen (siehe 2.1.2) liegt vor allem in der Anschaulichkeit der Darstellung. Durch die Wahl von Metaphern aus dem täglichen Leben, wird ein intuitives Erkennen relevanter Eigenschaften unterstützt. Einem Anwender müssen keine langwierigen Erklärungen gegeben werden, um ihm das Erfassen der präsentierten Daten zu ermöglichen.

Wie man aus den genannten Beispielen erkennen kann, bieten graphische Metaphern eine sehr vielfältige Methode. Der Phantasie sind keine Grenzen gesetzt, wenn es um die Wahl der richtigen Metapher geht. Allerdings bildet diese Vielfältigkeit

auch einen großen Nachteil, da es durch die Vielzahl der Möglichkeiten sehr schwierig sein kann, die richtige Metapher und eine passende Zuordnung der Daten zu den einzelnen Merkmalen zu finden. So aussagekräftig eine Metapher für einen Satz an Daten sein kann, so unpassend kann sie bei andersartigen Informationen sein, oder auch schon, wenn dieselbe Art von Daten nur anders verteilt sind.

Ein weiterer Nachteil für den Anwender ergibt sich aus der Schwierigkeit, aus den Graphiken auf die exakten Werte der dahinter liegenden Daten zu schließen. Diesen Nachteil müssen die meisten graphischen Methoden gegenüber einer textuellen Darstellung wettmachen, jedoch tritt die Ablesbarkeit eines konkreten Wertes bei Metaphern oft noch mehr in den Hintergrund, als dies z.B. bei der Darstellung in Diagrammen der Fall ist.

2.2 Vergleich und Bewertung der Verfahren

Ein Vergleich der im vorigen Abschnitt vorgestellten Verfahren ist nicht ganz einfach, da sie sich teilweise gegenseitig ergänzen bzw. völlig unterschiedliche Ziele verfolgen. Eine intelligente Vorselektion der Ergebnisse macht Sinn für sich alleine, aber auch bei zusätzlicher graphischer Aufbereitung und wird von vielen Suchseiten bereits erfolgreich angewendet. Auch das in dieser Arbeit vorgestellte Verfahren der graphischen Aufbereitung, könnte z.B. durch Clusterung noch aufgewertet werden. Im libViewer, dem Vorgänger des metaViewers, wurde ein Verfahren zu Clusterung angewendet (siehe auch 2.3) und auch für den metaViewer erscheint dies als interessante Erweiterung für die Zukunft. Die zusätzlichen Daten die durch Clusterung, Bewertungsverfahren und dergleichen generiert werden, stellen nichts anderes als Metadaten im oben definierten Sinne dar, d.h. sie liefern strukturierte Informationen zu den einzelnen Treffern (z.B. eine Kategorie oder die Relevanz u. Ä.). In diesem Fall werden diese Daten dazu benutzt, um Strukturen erkennbar zu machen bzw. die Anordnung der einzelnen Treffer zu beeinflussen.

Oft enthalten die Treffer selbst jedoch bereits Metadaten-Felder die wichtige Aussagen zu den Ergebnissen treffen und nicht erst durch aufwändige Verfahren generiert werden müssen. Diese Daten können manchmal die entscheidenden Informationen enthalten, die vom Anwender bei der Suche benötigt werden. Eine Darstellung die solche Daten übersichtlich und verständlich sichtbar macht, kann den Suchprozess sehr erleichtern. Folgende Schwierigkeiten treten dabei auf:

- Aus einer meist größeren Anzahl unterschiedlich interessanter Metadaten-Felder müssen jene mit relevanten und möglichst vollständigen Informationen zur Darstellung ausgewählt werden.

- Welches die interessantesten Felder sind, kann sich von Abfrage zu Abfrage auch aus derselben Quelle ändern. Auch die Qualität bzw. die Vollständigkeit

kann starken Schwankungen unterliegen.

- Metadaten aus verschiedenen Quellen unterscheiden sich stark in den vorhandenen Feldern und deren Ausprägungen.
- Welche der Informationen den Anwender am besten unterstützen hängt auch davon ab, nach was gerade gesucht wird und welche Präferenzen dieser bezüglich der Darstellung hat.
- Bei der Präsentation von mehreren (>3) Datenfeldern mit strukturierten Daten stellt man sich den Herausforderungen der Darstellung multidimensionaler Daten.

Auf Grund dieser Probleme die auftreten können, wenn versucht wird, Metadaten anschaulich zu präsentieren, werden diese in der Praxis meistens (wenn überhaupt) einfach in Textform den einzelnen Treffern hinzugefügt. Auch wenn in vielen Einzelfällen bereits die Nützlichkeit einer anschaulicheren Verwendung von Metadaten gezeigt wurde, fehlt ein allgemein gültiges Konzept das dabei angewendet werden könnte.

Eine verhältnismäßig einfache aber effektive Methode, den Anwender beim Vorselektieren der Ergebnisse zu unterstützen, ist die Anreicherung der Suchergebnisse mit Bildinformationen. Dies wird auch auf vielen aktuellen Suchseiten bereits praktiziert, besonders bei der Suche nach Produkten bietet es sich an, eine Abbildung des Artikels einzufügen. Die Verfügbarkeit eines brauchbaren Bildes hängt jedoch sehr von der Art der Datenquelle ab. Ist in den Metadaten kein aussagekräftiges Bildmaterial vorhanden[14], oder soll z.B. für ein gebildetes Cluster ein repräsentatives Bild ausgewählt werden (wie etwa in [26]), muss aus anderen Quellen ein Bild generiert werden. Bestehen die Ergebnisse aus Links zu verschiedenen Webseiten, kann z.B. auf diesen nach passendem Bildmaterial gesucht werden. Der Nachteil dieser Methode ist, dass sie sehr von der Qualität und der Aussagekraft des zur Verfügung stehenden Bildmaterials abhängt, deshalb sind ihre Einsatzmöglichkeiten begrenzt. Außerdem werden Informationen die in anderen, nicht bildorientierten Feldern der Metadaten enthalten sind, zur graphischen Darstellung nicht verwendet.

Ein Ausweg, wenn kein passendes Bildmaterial zur Verfügung steht, und trotzdem nicht auf eine anschauliche Darstellung verzichtet werden soll, bieten graphische Metaphern. Die Voraussetzung für die Verwendung von Metaphern in der hier vorgestellten Art, ist das Vorhandensein von Metadaten welche Eigenschaften der einzelnen Suchergebnisse näher beschreiben. Daraus können dann die Merkmale für die Darstellung abgeleitet werden. Man ist somit nicht auf vorhandenes Bildmaterial

[14]Wir zählen auch beigefügte Bilder zu den Metadaten. Wird z.B. ein Produkt durch ein Bild dargestellt, ist dieses in den Metadaten gewöhnlich als Link auf die Bilddatei vorhanden.

angewiesen sondern generiert sich selber eine graphische Darstellung aus den vorhandenen Informationen in den Metadaten. Dies wird bereits häufig angewandt, wenn z.B. Übersichtsdarstellungen aus Daten erstellt werden, die mit Hilfe von Cluster-Algorithmen ermittelt wurden. In solchen Fällen treten einige der oben genannten Probleme erst gar nicht auf, da die Metadaten selbst erzeugt wurden und man somit Kontrolle über deren Inhalt und Struktur hat. Allerdings stellt die Visualisierung auch mit diesen sehr begrenzten Daten eine Herausforderung dar, da kaum allgemein anerkannte Richtlinien zum Design der Darstellung existieren.

Verschärft wird dieses Problem noch, wenn die in den Treffern selbst vorhandenen Informationen zur Definition der Metapher genutzt werden sollen. Die vorstellbaren Metaphern erscheinen unendlich und die richtige Wahl dadurch umso schwieriger. Die große Zahl der Möglichkeiten hat jedoch den Vorteil, dass für alle Arten von Metadaten-behafteten Ergebnis-Listen und auch für unterschiedliche Anwendungen eine ansprechende Darstellung denkbar erscheint, wenn man Metaphern, je nach Art der Quelldaten, so wählt, dass der gewünschte Effekt beim Anwender erzielt wird. Wie auch schon im vorigen Abschnitt erwähnt, liegt ein großer Vorteil Metapher-basierter gegenüber Diagramm-basierter Visualisierungen in der Anschaulichkeit und Vielfalt der möglichen Darstellungen. Durch die vielen Variationsmöglichkeiten ist sozusagen für jeden etwas dabei, für den erfahrenen Datenanalysten oder den Internet-Gelegenheitsnutzer sowie den verspielten Visualisierer.

Zusammengefasst kann über Metapher-basierte Darstellung gesagt werden: Metadaten aller Art können damit visualisiert werden und dabei auch auf spezielle Anforderungen der konkreten Anwendung eingegangen werden kann. So mächtig diese Methode jedoch ist, so schwierig ist es, die richtige Metapher und die beste Zuordnung zu finden.

2.3 Verfahren genutzt im libViewer und im metaViewer

Wie erwähnt, bildet der libViewer [3] die Basis für die vorliegende Arbeit. Deshalb soll hier kurz auf die Verfahren eingegangen werden, die mit ihm realisiert wurden, besonders in Hinblick auf die eben getroffene Kategorisierung.

Der libViewer verwendet Metadaten, die in den Ergebnissen aus der Suche in Onlinebibliotheken enthalten sind, um die Ergebnisse als Bücher in einem Bücherregal darzustellen, verwendet also Methoden wie in den beiden vorangegangenen Abschnitten angeführt. Als graphische Metapher wurde das Buch gewählt, als Merkmale ergeben sich somit z.B. die Größe bzw. die Dicke des Buches, seine Farbe und Erscheinungsform (Ringbuch, Taschenbuch, ...) oder auch Grafiken und Texte die auf den Büchern abgebildet werden. Die Wahl des Buchs als Metapher war für diese

Art der Quellen sehr naheliegend. Auch viele Zuordnungen von Datenfeldern aus den Metadaten zu den Merkmalen der Bücher ergaben sich dadurch fast von selbst. Da die Ergebnisse und die gewählte Metapher sich in diesem Fall sehr ähnlich sind, konnten dadurch viele Schwierigkeiten vermieden werden, die bei der Wahl von Metaphern und Zuordnungen, auf Grund der vielen Möglichkeiten, auftreten können. So wird z.B. die Anzahl der Seiten des gefundenen Dokumentes als Basis für die Dicke des dargestellten Buches verwendet, und für erkannte Verlage wird ein entsprechendes Logo auf dem Buchrücken platziert. Da diese Logos vorab erstellt wurden, und die Bücher rein aus den Metadaten generiert werden, ist der libViewer nicht auf Bildmaterial in den Quelldaten angewiesen.

Zusätzlich zur graphischen Aufbereitung wurde eine der im Abschnitt 2.1.1 erwähnten Methoden zur Strukturierung der Ergebnisse mit dem libViewer angewendet. Es wurden Self-Organizing Maps verwendet, um eine inhaltsbasierte Clusterung der gefundenen Dokumente vorzunehmen [16, 17].

Der metaViewer folgt bei den verwendeten Methoden seinem Vorgänger grundsätzlich nach. Metadaten-behaftete Suchergebnisse werden durch graphische Metaphern visualisiert. Allerdings verlässt er dabei den „sicheren Pfad" der mit den Quelldaten gut übereinstimmenden Metaphern. Es wird im Gegenteil versucht, gerade die Probleme anzugehen, die bei der Visualisierung unterschiedlichster Daten durch beliebige Metaphern entstehen können. Abgesehen von der Möglichkeit weitere Quellen einzubinden und zusätzliche Metaphern zu implementieren, wurden speziell die vielfältigen Kombinationsmöglichkeiten von Metadaten-Feldern und Metapher-Merkmalen als würdiges Ziel eingehender Versuche angesehen (Abbildung 1.2.1). Es bleibt zu hoffen, dass mit seiner Hilfe und mit ihm als Basis für weitere Entwicklungen, einige der Nachteile bei der Verwendung Metapher-basierter Visualisierungen gemindert werden können.

Im Rahmen dieser Arbeit wurde auf die Verwendung von Cluster- und Ranking-Algorithmen, wie sie im Abschnitt 2.1.1 angeführt sind, verzichtet. Um alle Möglichkeiten dieses Ansatzes voll auszuschöpfen, wäre die Anwendung solcher Verfahren auch im metaViewer durchaus interessant. Auch die Anwendung hierarchischer Strukturen bei der Organisation der Ergebnisse, könnte die Untersuchung zusätzlicher spannender Aspekte ermöglichen. Gerade seine offene Architektur macht den metaViewer für weitere Experimente so interessant.

2.4 Zusammenfassung

Wir haben in diesem Kapitel unterschiedliche Methoden kennengelernt, wie Ergebnislisten von Suchanfragen aufbereitet werden können. Dabei wurden Aspekte wie Clusterung der Daten, die Rolle von Metadaten und die Wahl der Darstellung betrachtet und ihre Vor- und Nachteile beurteilt. Für weiterreichende Recherchen zu

diesem Thema bietet z.B. [25] einen guten Einblick in aktuelle und evtl. zukünftig angewendete Methoden.

Die im libViewer und im metaViewer angewandten Verfahren wurden gesondert angeführt. Auf das von ihnen verwendete Verfahren zur Darstellung von Metadaten-Informationen mit Hilfe graphischer Metaphern wurde speziell eingegangen. Dabei wurden Probleme und Möglichkeiten aufgezeigt, die mit dieser Art der Visualisierung einhergehen.

Um sich näher damit befassen zu können, werden im folgenden Kapitel einige Begrifflichkeiten zu diesem Thema genauer erläutert. Dies auch um klar zu definieren, was mit den einzelnen Begriffen im Kontext dieser Arbeit gemeint ist.

3 Grundlagen

Ergebnisse die man bei Suchanfragen erhält, sind sich vom Aufbau her meistens sehr ähnlich. Sie enthalten strukturierte Informationen zur gesuchten Entität (ein Artikel, ein Buch, ein Link zu einer Seite mit entsprechenden Informationen, ...). Damit verbundene Begriffe und Strukturen werden im Folgenden auch an Hand der Ergebnisliste einer Produktanfrage bei Amazon (Abbildung 3.1.1) näher erklärt. Dabei ist jedoch nicht die Wahl der Quelle entscheidend, um für unsere Zwecke geeignet zu sein, sondern die Form der darin enthaltenen Daten. Auch der durch diese Daten repräsentierte Inhalt spielt für die Betrachtungen vorerst noch keine Rolle. Trotzdem erleichtert es oft das Verständnis von zu allgemein gehaltenen Erklärungen, wenn auch auf konkrete Beispiele, und in diesem Fall speziell auch auf den Inhalt der Suchergebnisse verwiesen wird.

3.1 Metadaten

In [3, Seiten 14f] wird die Bedeutung des Begriffs Metadaten folgendermaßen erklärt:

> Metadaten sind strukturierte Daten um die Charakteristika einer Ressource zu beschreiben. Das Wort „meta", welches aus der griechischen Sprache kommt, beschreibt eine Natur von höherer Ordnung oder von größerer fundamentaler Art. Daher bezeichnet man Metadaten auch als übergeordnete Daten oder als „Daten über Daten".

Abbildung 3.1.1 zeigt einen Ausschnitt der Metadaten, wie sie in der Antwort auf eine Produktanfrage an Amazon enthalten sind. Wie man sieht, besteht jedes Element (Item) aus mehreren Datenfeldern. Diese Felder bestehen jeweils aus einer Bezeichnung (z.B. *„Titel"*) und dem dazugehörigen Inhalt (z.B. *„Harry Potter, Band 5: Harry Potter und der Orden des Phönix"*). Jedes Item enthält dasselbe Set an Datenfeldern. Dies gilt auch, wenn man in derselben Datenquelle nach einem anderen Begriff sucht. Wenn wir im Folgenden von *derselben Quelle* sprechen, ist damit gemeint, dass auch etwaige Parameter gleich gewählt werden, die Einfluss auf die Auswahl der Datenfelder in den Ergebnissen haben[1].

[1] Man kann bei Amazon in unterschiedlichen Katalogen suchen, damit ändern sich dann auch die enthaltenen Datenfelder. Außerdem kann das gewünschte Set der Felder durch Angabe von

Alle Arten von Datenquellen die Metadaten mit ähnlicher Struktur enthalten, können als Quelle für den metaViewer verwendet werden. Wichtig für eine sinnvolle Nutzung im metaViewer ist dabei, dass in den Metadaten der betrachteten Elemente Eigenschaften beschrieben werden, die bei der Suche für den Anwender von Interesse sind.

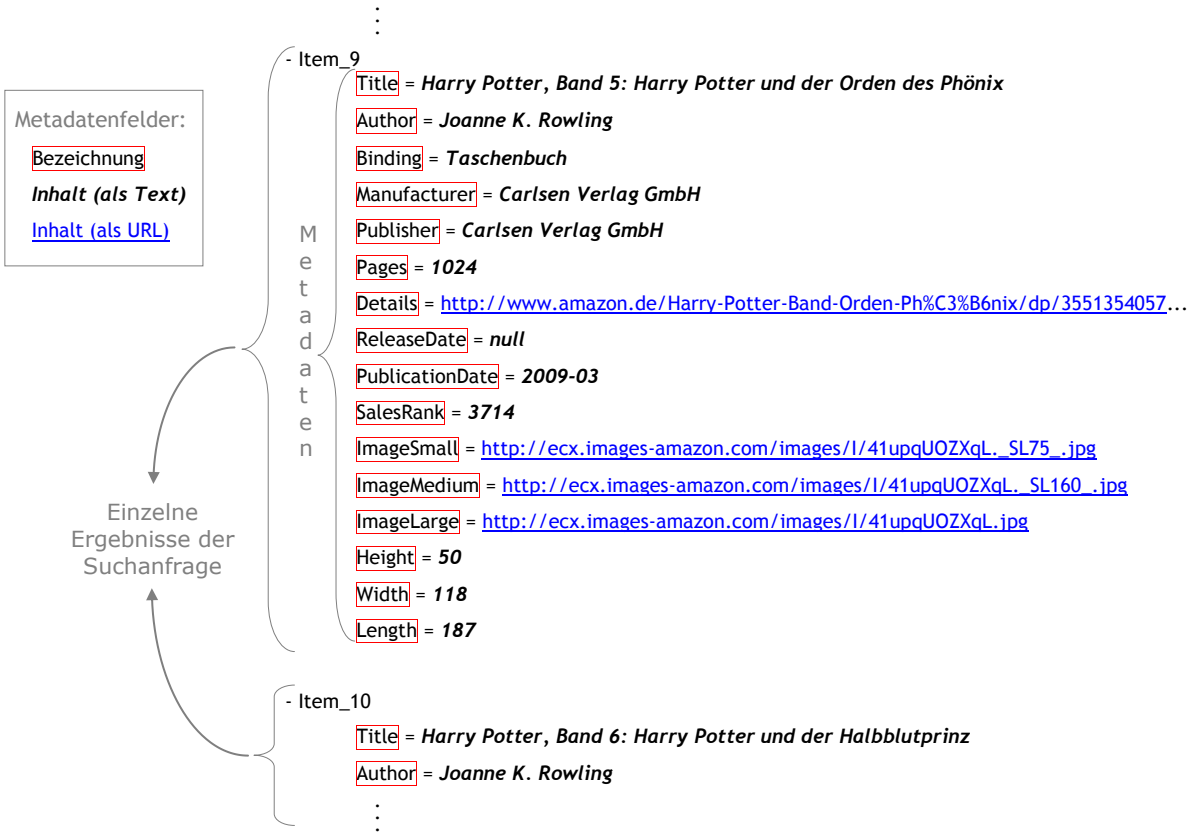


Abbildung 3.1.1: Auszug der Ergebnisse einer Produktabfrage bei Amazon. Gesucht wurde der Begriff „Potter" im Katalog „Bücher".

Parametern bei der Anfrage verändert werden. Für unsere Aufgabe wird der Katalog „*Books*" verwendet und die Auswahl (Parameter ResponseGroup) auf die Einstellung „*Small, ItemAttributes, SalesRank, BrowseNodes, Images*" fixiert. Damit wird sichergestellt, dass bei allen Anfragen dieselben Datenfelder zurückgeliefert werden.

3.2 Attribute

In einer Liste von Ergebnissen enthält jeder Eintrag (z.B. Item_9 in Abbildung 3.1.1) mehrere Datenfelder mit unterschiedlichen Bedeutungen, die durch ihre Bezeichnung (z.B."*Titel*") identifiziert werden. Alle Ergebniseinträge aus derselben Quelle besitzen dasselbe Set an Datenfeldern. Solch ein Feld, das für jeden Eintrag bestimmte Informationen enthält (z.B. Titel, Seitenanzahl, Größe, Preis, ...) wird bei der Verwendung im metaViewer allgemein als *Attribut* bezeichnet. Ein solches Attribut bezieht sich nicht nur auf eine Abfrage, sondern auf die Gesamtheit der Datenfelder mit derselben Bezeichnung in allen (möglichen) Abfragen aus dieser Quelle. Der Begriff *Datenfeld* wird im Weiteren nur noch verwendet, wenn speziell auf Eigenschaften der Datenquelle Bezug genommen werden soll.

Um im metaViewer verschiedene, auch erst in Zukunft erschlossene, Datenquellen verwenden zu können, müssen allgemein gültige Typen von Attributen identifiziert werden. Diese Attribut-Typen sollen nur auf Grund gemeinsamer Eigenschaften, jedoch unabhängig von der Bezeichnung eines Datenfeldes oder der Datenquelle aus der es stammt definiert werden. Als Beispiele in Abbildung 3.1.1 kann etwa zwischen Attributen die in irgendeiner Form eine Zahl repräsentieren (*Height*, *Width*, *Length*, ...) oder solchen, die nur als Text aufgefasst Sinn ergeben (*Titel*, *Author*, ...), unterschieden werden. Ein Text wiederum, enthält vielleicht als solches die entscheidende Information, oder muss als Link zu einem Bild (*ImageSmall*, *ImageMedium* und *ImageLarge*), zu einer Audiodatei oder zu einer Webseite mit weiterführenden Informationen (*Details*) aufgefasst werden.

Alleine aus der Betrachtung der Metadaten aus verschiedenen Quellen, lassen sich Attribute mit unterschiedlichen Eigenschaften und Informationen identifizieren. Im Folgenden soll ein Überblick über die Typen von Attributen gegeben werden, die wir in den untersuchten Quellen identifizieren konnten.

Textattribute: Das Textattribut bildet sozusagen die Basis für alle anderen Attribute, da in den meisten Quellen alle Felder als Text vorliegen bzw. als Text interpretiert werden können. Außerdem können alle weiteren Typen von Attributen leicht in Text umgewandelt werden. Die Ausnahme bilden hier Multimediainformationen wie Bilder, Audiodateien oder Videos, aber auch diese werden in allen von uns untersuchten Quellen als Link übermittelt und dieser kann wiederum als Text behandelt werden.

Ganzzahlige Attribute: Ganzzahlige (oder Integer) Attribute, bilden die häufigsten Vertreter von numerischen Feldern. Daten wie Seitenanzahl, Maße in mm, Gewichte in g oder auch jede Art von Nummerierungen (z.B. der *SalesRank* in3.1.1) werden als ganze Zahlen angegeben.

Fließkomma Attribute: Fließkommazahlen (oder Floating Point) fassen alle Arten von numerischen Werten zusammen, damit schließen sie auch den vorigen Typ der ganzen Zahlen mit ein. Typische Beispiele wären etwa Preise, Maße in Metern oder Gewichte in kg. Enthält das Attribut nur ganze Zahlen bzw. soll nur der ganzzahlige Anteil betrachtet werden, kann es von Vorteil sein, ein Integer Attribut zu verwenden. Eine Umwandlung kann sinnvoll sein, wenn z.B. spezielle Eigenschaften von ganzen Zahlen genutzt werden sollen, etwa eine gesonderte Behandlung des Wertes „0“ oder weil für die weitere Verwendung diskrete Werte vorausgesetzt werden.

Datum und Uhrzeit: Datenfelder die in irgendeiner Form ein Datum, eine Jahreszahl oder eine Uhrzeit enthalten, können in dieser Kategorie zusammengefasst werden. Für eine Liste von Büchern wäre z.B. das Datum der Publikation ein typischer Vertreter. Was hier besonders auffällt, sind die unterschiedlichen Formate, sogar innerhalb desselben Feldes der gleichen Quelle, in denen solche Daten auftreten können.

Multimediainformationen: Wie schon bei den Textattributen erwähnt, werden Bilder, Webseiten sowie Audio- und Videoinformationen in den Metadaten durch die Angabe eines Links repräsentiert. Obwohl es für die spätere Darstellung natürlich einen Unterschied macht, um welche Art von Information es sich jeweils handelt, kann an dieser Stelle der Link (URL) wie ein Textattribut behandelt werden.

Set-Attribute: Enthält ein Attribut nur eine begrenzte Anzahl an Ausprägungen, kann diese Eigenschaft von besonderem Interesse sein. Ein Beispiel hierfür wäre z.B. der Bucheinband (*Binding* in 3.1.1). Solche Attribute lassen sich später auf Metaphern abbilden, die bereits auf den ersten Blick eine grobe Einteilung der Ergebnisse ermöglichen sollen. So kann etwa ein Taschenbuch, ein gebundenes Buch, ein Ordner oder auch eine CD-Hülle für die Darstellung eines Ergebniseintrages verwendet werden, je nach Ausprägung eines solchen Set-Attributes. Bei Bedarf können auch numerische oder zeitliche Attribute in Sets umgewandelt werden, indem sie in Wertebereiche bzw. Zeiträume unterteilt werden. Wie das durchgeführt werden kann, wird in 4.3.1 näher erläutert.

Im metaViewer können neue Attribute aus bestehenden erzeugt werden (mehr dazu in 4.3.2). Solch ein erstelltes Attribut bezeichnen wir als *abgeleitetes Attribut*, im Gegensatz zu einem *Quell-Attribut* welches beim Erschließen einer Datenquelle direkt aus der Zuweisung eines Datenfeldes gebildet wird und während der Verwendung des metaViewers nicht verändert wird. Ein abgeleitetes Attribut kann erneut als Basis für ein weiteres abgeleitetes Attribut dienen. Zur Beschreibung einer solchen Ableitung wollen wir das Attribut von dem ausgegangen wird als *Original-Attribut* und das Ergebnis als *Ziel-Attribut* bezeichnen.

3.3 Metaphern und ihre Merkmale

Jedem Element aus der Ergebnisliste wird im metaViewer eine graphische Repräsentation zugeordnet. Wie diese auszusehen hat, soll prinzipiell offen gelassen werden, als Beispiel werden wir hier jedoch immer wieder auf die Repräsentation als Buch zurückgreifen.

Das Gegenstück zum Attribut aus unserer Datenquelle bildet ein *Metapher-Merkmal* als Teil unseres "Zieles", also der graphischen Darstellung. Wollen wir je einen Eintrag aus unserer Datenquelle z.B. als Buch (= Metapher) darstellen, wären etwa die Dicke des Buches, seine Farbe, seine Position oder ein Logo auf seinem Umschlag jeweils Merkmale dieser Metapher.

Um im metaViewer alle möglichen Arten von Metaphern zu unterstützen, muss auch hier wieder eine allgemeine Schnittstelle, diesmal für Metapher-Merkmale, geschaffen werden. Wie beim Versuch einer Verallgemeinerung von Attributen, ergibt sich auch bei Metaphern das Problem, dass die Anzahl denkbarer Variationen von graphischen Metaphern kaum überschaubar ist. Um aber die Eignung unterschiedlicher Darstellungen und den sich daraus ergebenden Metaphern und Merkmalen zu untersuchen, müssen auch diese auf irgendeine Art zusammengefasst werden. Eine rein praktische Unterteilung ergibt sich dadurch, dass untersucht wird, welche Arten von Attributen einem Metapher-Merkmal zugeordnet werden können. Z.B. kann ein Merkmal das die Dicke einer Buch-Metapher definiert durch eine Zahl festgelegt werden, die Zuweisung eines Textes erscheint in diesem Fall jedoch weniger sinnvoll.

Weiteren Einfluss auf die Wahl der Merkmals-Typen hatte der Umstand, dass damit konkrete Eigenschaften der graphischen Darstellung festgelegt werden sollen. Da diese Eigenschaften in der Umsetzung durch Variablen zur Verfügung stehender Datentypen repräsentiert werden, erschien eine Anlehnung an diese Datentypen sinnvoll zu sein. Dadurch ließen sich auch Probleme wie z.B. die Frage, ob sich einzelne Werte korrekt einem Merkmal zuordnen lassen oder auch die Umwandlung von Typen, leichter bewerkstelligen.

Die folgende Liste enthält die Kategorien von Metapher-Merkmalen, die von uns identifiziert werden konnten. Die Zugehörigkeit zu solch einem Merkmals-Typ legt bereits grundsätzlich fest, welche Attributs-Typen dem Merkmal zugeordnet werden können. Zusätzlich kann jedoch für ein Merkmal eine Einschränkung der erlaubten Ausprägungen sinnvoll sein. Das Festlegen einer minimalen und einer maximalen Dicke oder Höhe eines Buchs oder die Begrenzung eines Text-Merkmals auf eine maximale Länge wären Beispiele solcher Einschränkungen. Diese ergeben sich bei der Definition des Merkmals direkt aus dessen (graphischen) Eigenschaften. In der folgenden Beschreibung der Merkmals-Typen sind auch Beispiele möglicher Einschränkungen angeführt, die für Merkmale des jeweiligen Typs gelten könnten.

Text: Einem Merkmal das Buchstaben oder Zahlen darstellen soll, können diese als

Text übergeben werden. In der Regel ist keine weitere Beeinflussung der graphischen Darstellung mit dem Inhalt des Textes verknüpft, außer vielleicht, wenn sich die Metapher anders verhält, wenn der übergebene Text leer ist. Einschränkungen können sich z.B. durch die zulässige Länge des Textes ergeben. Weiters ist für den Anwender Interessant, wo und in welcher Größe und Form dieser Text dargestellt wird. Ein Beispiel wäre z.B. ein mehrzeiliges Text-Merkmal das auf der Vorderseite eines Buches dargestellt wird.

Ganze Zahl: Viele Eigenschaften einer Metapher lassen sich in Zahlen ausdrücken, die meisten davon auch als ganze Zahlen, ohne dabei einen entscheidenden Verlust an Genauigkeit eingehen zu müssen. Alle Arten von Ausdehnungen und Größen aber auch etwa die Häufigkeit des Auftretens von Metapher-Teilen oder irgendwelche Schwellwerte die das Aussehen der Metapher beeinflussen, lassen sich z.B. mit ganzen Zahlen festlegen. Als mögliche Einschränkung bietet sich z.B. die Angabe eines Gültigkeitsbereiches an.

Fließkomma: Mit Fließkommazahlen lassen sich ähnliche Eigenschaften wie mit ganzen Zahlen steuern. Wenn die Nachkommastellen einen entscheidenden Einfluss haben könnte die Verwendung eines Fließkomma-Merkmals vorteilhaft sein. Es gelten dieselben Möglichkeiten zur Einschränkung wie bei den ganzen Zahlen.

Datum / Zeit / Dauer: Sollen Informationen wie das Alter oder die Dauer eines Zeitraumes oder z.B. die Tageszeit zur Steuerung der Darstellung[2] genutzt werden, bietet sich ein Merkmal an, das mit Datums- und Zeitformaten umgehen kann. Z.B. könnte das Alter eines Buches Einfluss auf sein Aussehen oder seine Position im Bücherregal nehmen. Einschränkungen bezüglich des erlaubten Formates als auch des erwünschten Datums- oder Zeitbereiches müssen beachtet werden.

Link: Soll in irgendeiner Weise Information verwendet werden, die außerhalb der Metadaten gespeichert ist, kann dies durch einen Link erfolgen. Dieser kann z.B. ein URL oder ein Verzeichnispfad sein. Die Einschränkungen betreffend die Form sind dadurch bereits fix vorgegeben, die Frage, was sich am Ziel des Links befindet, ist jedoch entscheidend. Die Beschreibung des Merkmales muss dabei Auskunft geben, ob etwa ein Bild, eine Audiodatei, eine Web-Seite, ein Verzeichnis oder eine bestimmte Datei erwartet wird. Die Einschränkung besteht also darin, auf die richtige Art von Informationen (mit einem syntaktisch korrekten Link) zu verweisen.

[2]Soll das Datum oder die Zeit nur ausgegeben werden, ist die Verwendung eines Text-Merkmals vorzuziehen!

Set: Mit einem Set-Merkmal lassen sich Eigenschaften steuern, die nur eine eingeschränkte Anzahl an Ausprägungen zulassen. Die Darstellung eines Buches als gebundene Ausgabe oder als Taschenbuch bzw. als CD-Hülle wäre so eine Eigenschaft. Ein weiteres Beispiel wäre eine Anzahl von Piktogrammen die je nach gewähltem Wert auf der Metapher dargestellt werden.
Eine Einschränkung erfolgt durch Festlegung auf eine Liste von erlaubten Werten. Durch eine flexible Möglichkeit einer Zuordnung von Werten auf die erlaubten Ausprägungen, kann jedes Attribut als Basis zur Steuerung eines Set-Merkmales verwendet werden (siehe *CASE-Anweisung* in 4.3.1).

Boolean: Manche Merkmale beschränken sich auf die Möglichkeiten Ein/Aus oder True/False. Diese könnten ebenfalls durch ein Set-Merkmal repräsentiert werden, die Einschränkung auf einen Boolean-Wert macht die Verwendung aber oft einfacher.

Farbe: Farbe spielt bei der graphischen Darstellung eine große Rolle. Die Verwendung von Farben in einem bewusst offen gestalteten System kann auf unterschiedliche Arten gesteuert werden. Eine Möglichkeit ist, in einer dem Set-Merkmal ähnlichen Art, eine Auswahl an Farben anzubieten, aus denen dann, je nach Ausprägung des Attribut-Wertes, eine ausgewählt wird. Eine weitere und oft zielführendere Möglichkeit ist, einen Farbverlauf anzubieten, aus dem durch Zuweisung einer Zahl eine Farbe ausgewählt wird. Die Einschränkung des erlaubten Zahlenbereiches muss dabei beachtet werden.

Die dieser Unterteilung zugrunde liegenden Zuordnungsmöglichkeiten von Attributen beziehen sich ausschließlich auf strukturelle Eigenschaften. Ob die Wahl eines Metapher-Merkmals für ein gewähltes Attribut *inhaltlich* Sinn macht, wird dabei vorläufig außer Acht gelassen. Auch ob die Metapher zur Darstellung von Inhalten überhaupt geeignet ist, wird damit nicht geklärt. Solche Fragestellungen sollen dann erst mit konkreten Anwendungen untersucht werden!

3.4 Zuordnung von Attributen zu Metapher-Merkmalen

Die in den letzten beiden Punkten beschriebenen Einteilungen von Attributen und Metapher-Merkmalen erfolgt nach rein praktischen Gesichtspunkten jedoch bereits in Hinblick auf das eigentliche Ziel, die graphische Repräsentation der Listenelemente durch Metaphern. Man kann aus den Typen der Attribute und der Metapher-Merkmale bereits Rückschlüsse ziehen, ob diese, zumindest strukturell, einander zugeordnet werden können. Also wird dadurch z.B. verhindert, dass einem Metapher-

Merkmal das durch eine Zahl bestimmt wird, wie etwa Höhe oder Dicke eines Buches, ein Text-Attribut zugeordnet werden kann.

Wenn man die Unterteilungen der vorigen beiden Punkte vergleicht fällt auf, dass die Merkmale *Boolean* und *Farbe* bei den Attributen keine Entsprechung finden. Diese erste Auswahl an Attribut-Typen erfolgte auf Grund der in den untersuchten Quellen vorhandenen Datenfelder. Darin waren keine konkreten Repräsentationen von Farben enthalten. Trotzdem spielt die Auswahl von Farben für die Darstellung graphischer Metaphern natürlich eine wichtige Rolle und es fanden sich somit Merkmale die eine Farbinformation als Wert verlangen. Ein Attribut das solche Informationen liefern kann, erschien also sinnvoll.

Das *Boolean-Merkmal* hingegen ergibt sich aus rein praktischen Gründen, da sich damit gewisse Einstellungen bei der Darstellung leichter aktivieren oder deaktivieren lassen. Beide Typen sind zwar nicht direkt in den Quelldaten zu erwarten, sollten aber als *Ziel-Attribut* zur Verfügung stehen. Wie in Abschnitt 4.3.1 zu sehen ist, wurden deshalb beide, zusätzlich zu den oben identifizierte, im metaViewer umgesetzt. Eine genauere Beschreibung der Eigenschaften dieser beiden Attribut-Typen findet sich ebenfalls dort.

Zusätzlich zum *Typ* des Attributes bzw. des Metapher-Merkmals gibt es weitere Kriterien, die bei der Zuweisung eine Rolle spielen. Eigenschaften wie die Länge der Texte eines Attributs, der Wertebereich eines numerischen Attributs oder z.B. die Größe bzw. Auflösung von Bildern, auf die von einem Attribut über Links verwiesen wird, können für die Zuweisung entscheidend sein. Zusätzliche Informationen über die Daten der Attribute sowie über die Beschaffenheit der Metapher-Merkmale sind deshalb für den Anwender sehr wichtig und können ihn bei der Suche nach einer guten Zuordnung unterstützen.

Dasselbe Attribut kann mehreren Metapher-Merkmalen zugeordnet werden. So kann z.B. der Titel eines Buches auf dem Buchrücken und auf der Vorderseite angeführt sein. Umgekehrt muss einem Merkmal jedoch ein eindeutiger (oder gar kein) Wert zugewiesen werden. Sollen mehrere Attribute für die Ausprägung eines Merkmales verantwortlich sein, müssen diese bereits bei der Definition der Attribute in irgendeiner Form zusammengefasst werden (siehe auch 4.3.2). Um solche Verknüpfungen umzusetzen, bieten sich, je nach Typ, unterschiedliche Methoden an. Numerische Attribute können z.B. in mathematischen Ausdrücken gemeinsam verarbeitet werden. Texte kann man mit Hilfe von Konkatenierungsoperationen zusammenfügen und *Set-* oder *Boolean-Attribute* können mittels CASE-WHEN-ELSE-Anweisungen (siehe 4.3.1) Einfluss auf die Werte eines mit ihnen verknüpften Attributs nehmen. Um den sich daraus ergebenden Kombinationsmöglichkeiten gerecht zu werden, musste eine Methode entwickelt werden, mit der solche Verknüpfungen möglichst ohne Einschränkungen durchgeführt werden können. Ohne den Aufwand dafür allzu groß werden zu lassen, bietet sich eigentlich nur die Verwendung einer

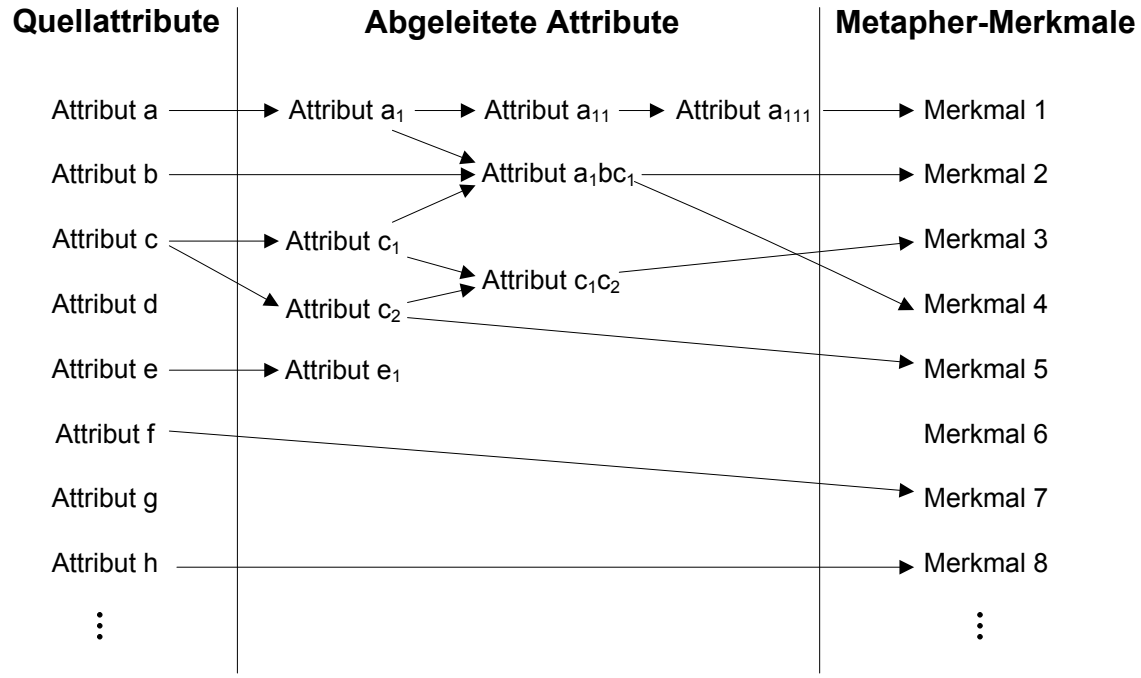


Abbildung 3.4.1: Einige Beispiele, welche Kombinationen von Ableitungen aus den Quellattributen durchgeführt werden können und welche Zuordnungen daraus möglich sind.

existierenden Programmiersprache an. Wie in Abschnitt 4.3.2 erläutert, verwendet der metaViewer zu diesem Zweck die Datenbank-Abfragesprache *SQL*, die all die geforderten Operationen zur Verfügung stellt.

Aus diesen Überlegungen heraus sind Kombinationen von Verknüpfungen und Ableitungen von Attributen sowie Zuordnungen zu Metapher-Merkmalen möglich, wie sie im folgenden Bild skizziert werden:

Wie man sieht, können mehrere Attribute zur Definition eines neuen abgeleiteten Attributs verwendet werden (z.B. *Attribut* a_1bc_1) oder Attribute mehrfach aus einem bestehenden abgeleitet werden, wobei nicht unbedingt nur das letzte und auch nicht nur eines der daraus entstehenden Attribute einem Merkmal zugewiesen werden kann (*Attribut* a bis *Attribut* a_{111} oder *Attribut* c_2). Dasselbe Attribut kann mehreren Merkmalen zugeordnet werden (*Attribut* a_1bc_1), jedem Merkmal allerdings, kann maximal ein Attribut zugewiesen werden. Weder alle Quell-, noch alle abgeleiteten Attribute und auch nicht alle Metapher-Merkmale müssen für die Zuordnung verwendet werden (*Attribut* g, *Attribut* e_1 bzw. *Merkmal* 6). Ungenutzte Merkmale werden bei der Darstellung dann mit einem Defaultwert belegt. Über die

Typen der Attribute und der Merkmale werden in Abbildung 3.4.1 keine Aussagen getroffen. Aus der Tatsache, dass nur Attribute entsprechenden Typs einem Merkmal zugeordnet werden können, kann man z.B. darauf schließen, dass *Merkmal* 2 und *Merkmal* 6 vom gleichen Typ sein müssen, da ihnen dasselbe Attribut zugeordnet wird.

Erfüllt eine Attribut–Merkmal Paarung alle strukturellen Voraussetzungen, sagt dies noch wenig über die tatsächliche Aussagekraft der damit erzeugten Darstellung aus. Wie gut eine solche Paarung zur Vermittlung von Informationen geeignet ist, sollte sich dann bei konkreten Versuchen herausfinden lassen.

3.5 Zusammenfassung

In diesem Kapitel wurden Grundlagen vermittelt und Definitionen vorgenommen, die im weiteren Verlauf dieser Arbeit benötigt werden. Im ersten Abschnitt wurde erklärt, was wir als *Metadaten* bezeichnen und in welcher Form sie in den Suchergebnissen vorliegen. Dann wurde auf die hier oft verwendeten Begriffe *Attribut, Metapher* und *Metapher-Merkmal* näher eingegangen. Dabei konnte man schon einige der Aufgaben erkennen, die bei der Entwicklung des metaViewers gelöst werden mussten. Speziell wie wichtig eine klare Definition der Schnittstellen für Attribute und Metapher-Merkmale ist, wurde deutlich. Am Schluss wurde noch kurz aufgeführt, was die grundlegenden Überlegungen sind, wenn Attribute den Metapher-Merkmalen zugeordnet werden.

Im nächsten Kapitel sehen wir, wie die gesammelten Ideen und Vorgaben im metaViewer umgesetzt wurden. Dabei tauchen auch viele der bisher eher theoretisch behandelten Probleme wieder auf und man erfährt, wie im metaViewer die konkreten Lösungen dafür aussehen.

4 Umsetzung

Ganz grundsätzlich bietet der metaViewer Unterstützung dabei, bei der Visualisierung von strukturierten Daten unterschiedliche Darstellungen und Zuordnungen auszuprobieren. Der Anwender hat dabei die Möglichkeit, Daten aus den erschlossenen Quellen durch eine der implementierten graphischen Ausgabemöglichkeiten darzustellen. Dabei können die zur Verfügung gestellten Attribut-Typen verwendet, mit den angebotenen Funktionen manipuliert und den Metapher-Merkmalen der gewählten Graphik zugeordnet werden. Bei der Entwicklung des metaViewers wurde speziell darauf geachtet, diese Zuordnung so flexibel wie möglich zu gestalten. Um diese Möglichkeiten des metaViewers aufzuzeigen, wurden beispielhaft einige Datenquellen erschlossen und graphische Darstellungen implementiert. Die eigentlichen Fähigkeiten des metaViewers erschließen sich allerdings erst, wenn man bereit ist, in den Programmcode des metaViewers einzugreifen und, je nach Bedarf, Erweiterungen zu implementieren. Nicht nur bei den Bedienmöglichkeiten des metaViewers wurde auf größtmögliche Flexibilität Wert gelegt, auch bei den Programmschnittstellen wurden darauf geachtet, das Erweitern um zusätzliche Komponenten möglichst zu unterstützen. Speziell folgende Erweiterungsmöglichkeiten wurden bereits bei der Entwicklung vorgesehen:

- Erschließen neuer Datenquellen
- Entwurf neuer Attribut-Typen
- Erweiterung der Bearbeitungsmöglichkeiten von Attributen
- Definition neuer Typen von Metapher-Merkmalen
- Implementierung weiterer Darstellungsmöglichkeiten durch die Entwicklung neuer Metaphern
- Definition weiterer Restriktionen (siehe 4.4.3) die bei Merkmalen angewendet werden können

In diesem Kapitel werden vorrangig die tatsächlich im metaViewer umgesetzten Bedienungsmöglichkeiten beschrieben, sowie die Überlegungen die bei deren Umsetzung angestellt werden mussten. Auf die Erweiterungsmöglichkeiten des Programmcodes und die dafür benötigten Schnittstellen wird dann in Kapitel 5 näher eingegangen.

4.1 Datenfluss, der Weg durch den metaViewer

Um an die Umsetzung der vielen Ideen heranzugehen, die einem im Laufe der Themenfindung und auch später während der Umsetzung kommen, ist es hilfreich diese zu ordnen. Eine mögliche Ordnung dafür bieten z.B. die einzelnen Stationen des Datenflusses durch die Applikation. Angefangen beim Abfragen der Daten von einer Datenquellen bis zur graphischen Darstellung lassen sich Schnittstellen identifizieren, die als Grenzen auch für die Ordnung von Ideen und umzusetzender Funktionalitäten dienen können. Eine Beschreibung der in Angriff genommenen Aufgaben in der Anordnung entlang dieses Datenflusses bieten die folgenden Abschnitte dieses Kapitels. Das Flussdiagramm in 4.1.1 skizziert den Ablauf der Arbeitsschritte die beim Finden einer geeigneten graphischen Darstellung durchlaufen werden.

In den Abschnitten 3.2 und 3.3 wurde beschrieben, wie die Schnittstellen zum einen an der Eingangsseite (in 4.1.1 zwischen *Datenquelle* und *Attribut-Bereich*) und zum anderen auf der Ausgabeseite (zwischen *Attribut-Bereich* und *Metapher-Bereich*) definiert wurden. Durch diese Trennung von Ein- und Ausgabe kann bei den späteren Versuchen die Darstellungsform unabhängig von der Datenquelle gewählt werden. Diese Offenheit des Systems war die Grundidee für die vorliegende Arbeit und beanspruchte am meisten Aufmerksamkeit bei der Planung und Implementierung der dabei entstandenen Software. Leider konnten nicht alle sich anbietenden Datenquellen erschlossen, alle interessanten Darstellungsformen implementiert und alle aufkeimenden Ideen hier verfolgt werden, da dies den Rahmen dieser Arbeit bei weitem gesprengt hätte. Das Ergebnis der vorliegenden Arbeit bietet jedoch eine gute Basis, dies alles in zukünftigen Projekten weiter zu verfolgen und gibt einen Startpunkt vor, der bereits Lösungen für viele auftretende Probleme enthält.

4.2 Quellen für Suchergebnisse

Die Auswahl einer speziellen Suchquelle spielte für das vorliegende Problem nur eine untergeordnete Rolle. Vielmehr kam es darauf an, einen Überblick über die verschiedenen Arten von Quellen und die daraus zu erhaltenden Daten zu gewinnen. Die Erschließung von Datenquellen, d.h. die Methode wie der metaViewer Daten aus einer speziellen Quelle auslesen kann, ist vorrangig ein technisches Problem. Obwohl auch dieses gelöst werden musste und muss, interessieren uns vielmehr die Gemeinsamkeiten die in unterschiedlichen Quellen gefunden werden können, um daraus die Spezifikation für die erste wichtige Schnittstelle in unserem Datenfluss zu erstellen. Dafür wurden die in den Metadaten der Ergebnisse vorkommenden Datenfelder untersucht, um daraus auf eine Auswahl an Attribut-Typen zu schließen, mit Hilfe derer jede Art von Datenfeld möglichst komfortabel weiterverarbeitet werden kann. Die Einteilung der Attribute in Typen mit denen so etwas wie eine Hülle über alle

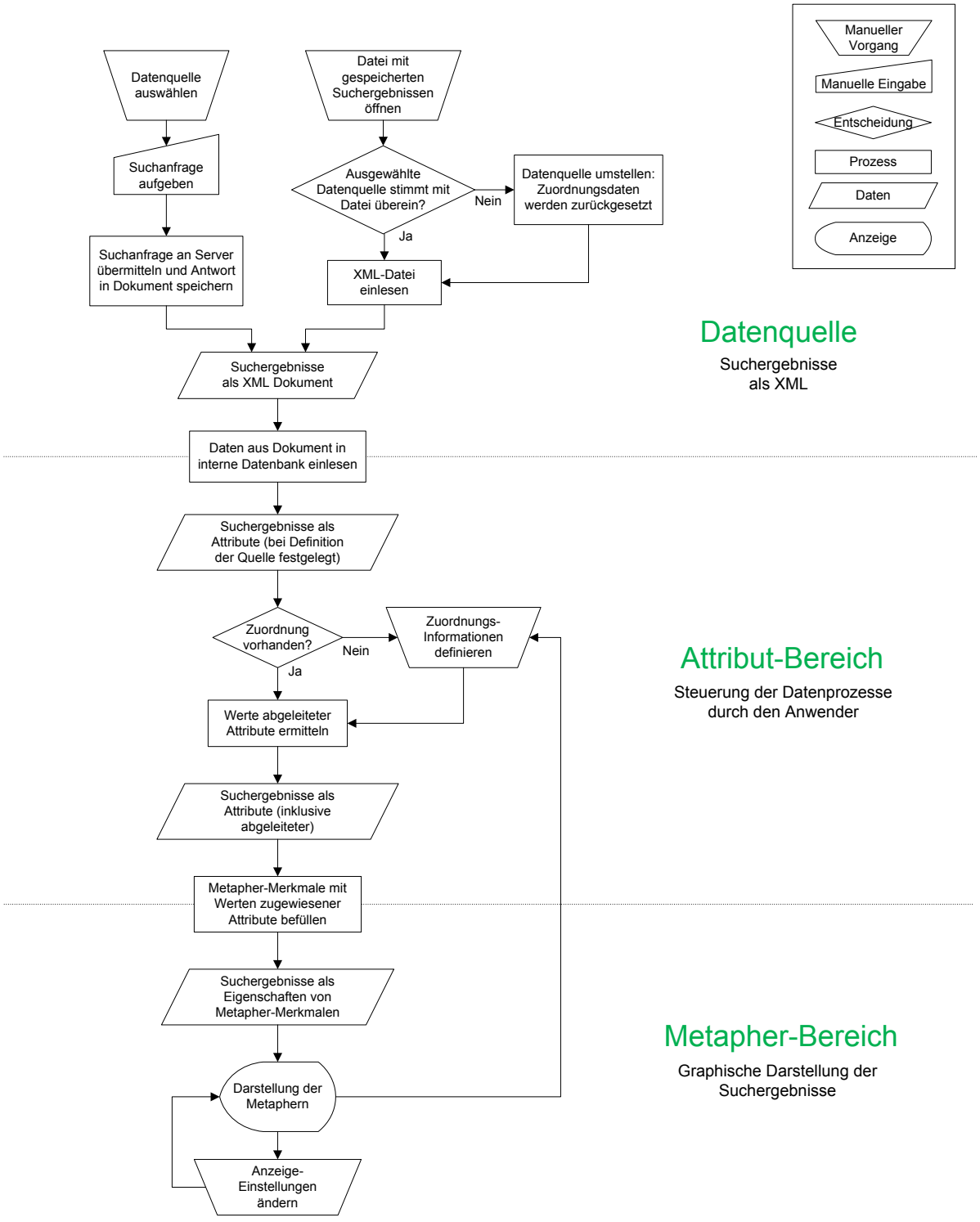


Abbildung 4.1.1: Dieses Flussdiagramm des metaViewers zeigt die Schritte die beim Definieren einer Visualisierung durchlaufen werden.

möglichen Datenfelder gebildet werden soll führte auch zur Definition der Attribut-Schnittstelle wie in 5.5.2 beschrieben.

Für die Entwicklung des metaViewers, im Speziellen für die Entscheidung welche Typen von Attributen er unterstützen sollte, spielte die Wahl der tatsächlich abgefragten Quellen nur bedingt eine Rolle. Viel wichtiger war die Frage, welche Arten von Metadatenfelder insgesamt zu erwarten sind. Dafür war es nicht unbedingt notwendig die Quelle direkt abzufragen, sondern es reichte aus Informationen über die Datenfelder in den Ergebnissen zu analysieren. Trotzdem war die Einbindung einer möglichst vielfältigen Quelle während der Entwicklung notwendig, um die implementierten Funktionen zu testen. Die wichtigsten Datenquellen die für diese Zwecke untersucht wurden, werden hier nun kurz vorgestellt.

Amazon (*http://www.amazon.de/*) Bietet eine sehr gut zu verwendende Schnittstelle zu all seinen Produkten die online angeboten werden. Dabei ist der Produktkatalog in mehrere Kategorien unterteilt (z.B. Bücher, Bekleidung, Musik, ...). Auf Grund der vielen unterschiedlichen Arten von Produkten sowie der großen Anzahl an Einträgen stellt Amazon einen idealen Einstiegspunkt dar, um einen möglichst umfangreichen Eindruck von zu erwartenden Datenfeldern zu erhalten. Von allen betrachteten Quellen sind in diesen Abfragen die meisten und vielfältigsten Metadatenfelder vorhanden, deshalb wurde die Produktabfrage an Amazon als die zentrale Datenquelle während der Entwicklung des metaViewers verwendet. Durch die unterschiedlichen Kataloge und die große Anzahl an aussagekräftigen Datenfeldern bot sie auch die ideale Basis, um eine Einteilung der Attribute in Typen zu treffen.

Dateisystem Während der Entwicklung des metaViewers wurde die Abfrage des Inhaltsverzeichnisses von Dateisystemen immer als mögliche weitere Datenquelle angesehen. Auch ohne wirkliche Umsetzung war klar, dass die daraus ablesbaren Metadaten für die Verarbeitung im metaViewer durchaus interessant wären. Die tatsächliche Abfrage von Verzeichnisdaten wurde jedoch erst gegen Ende dieser Arbeit eingebaut, als die zentralen Komponenten bereits fertig entwickelt waren. Dadurch konnte eindrucksvoll gezeigt werden, wie gut die Einbindung einer neuen Datenquelle in den metaViewer durch die entwickelten Schnittstellen unterstützt wird. Wie zu erwarten war, enthalten die so ausgelesenen Informationen eine Vielzahl aussagekräftiger Metadatenfelder, deren graphische Darstellung mit den Methoden des metaViewers sich durchaus lohnen.

Google (http://www.google.at/) Als zurzeit sicherlich meist genutzte Suchmaschine, wurde die Google Suchseite natürlich als Quelle in Betracht gezogen. Die Datenfelder die bei einer Suche von Google geliefert werden sind jedoch sehr spärlich. Wollte man weitere interessante Informationen, also Metadaten, aus

den Suchergebnissen gewinnen, müssten die Webseiten auf die in den Treffern verwiesen wird inhaltlich analysiert werden. Außerdem stellte Google, zu der Zeit als wir infrage kommende Datenquellen untersuchten, keine frei verfügbare Programmierschnittstelle zur Verfügung, um maschinell Suchanfragen durchführen zu können[1]. Aus diesen Gründen wurde in der ersten Version des metaViewers auf das Erschließen dieser Quelle verzichtet. Dies stellte für die Entwicklung des metaViewers keine Einschränkung dar, da mit der Abfrage von Amazon Produkten eine viel Metadaten-reichere Quelle zur Verfügung stand, deshalb wurde auf diesen zusätzlichen Aufwand im Rahmen dieser Arbeit verzichtet, um sich auf die wesentlichen Aufgaben konzentrieren zu können.

Yahoo (http://at.search.yahoo.com/) Für Abfragen an die Suchseite von Yahoo gelten im Prinzip dieselben Einschränkungen wie für Google. Allerdings war eine funktionierende Programmierschnittstelle verfügbar, deshalb wurde mit der Einbindung dieser Quelle begonnen. Da jedoch auch hier die interessanten Metadaten selbst aus den Webseiten der Ergebnisse generiert werden hätten müssen, wurde auf die Weiterentwicklung dieser Anbindung verzichtet. In Verbindung mit Prozessen zur Extraktion von Metadaten aus Webseiten, würde die Einbindung reiner Suchmaschinen durchaus interessant werden. Diese Art der Aufbereitung lag jedoch nicht im Fokus dieser Arbeit, deshalb wurde auf Quellen zurückgegriffen, die bereits eine erschöpfende Auswahl an Metadatenfeldern bereitstellen.

4.3 Attribute

In Abschnitt 3.2 wurde erklärt, was wir unter Attributen verstehen und wie diese unterteilt werden können, auf Basis der Repräsentation der Datenfelder in den Metadaten die mit ihnen verarbeitet werden sollen. Hier wollen wir nun näher darauf eingehen, wie diese Unterteilung die weitere Verwendung beeinflusst und welche Möglichkeiten sich für die Anpassung unterschiedlicher Attribute ergeben. Im Diagramm in Abbildung 4.1.1 haben wir damit die erste Schnittstelle überschritten und bewegen uns nun im Attribut-Bereich.

Wenn eine neue Datenquelle für den metaViewer erschlossen wird, müssen die in den Metadaten enthaltenen Datenfelder identifiziert und einem Attributs-Typ zugeteilt werden. Da die Metadaten prinzipiell in textueller Form vorliegen[2], könnte,

[1] Dies hat sich mittlerweile geändert, wie man hier sehen kann: http://code.google.com/apis/customsearch/v1/overview.html.

[2] Die Ausnahme bilden Multimediadaten, die streng genommen selbst auch zu den Metadaten gehören. In diesem ersten Schritt der Einteilung von Attributen werden wir allerdings aus-

ohne weitere Überlegungen anstellen zu müssen, jedes der dabei entdeckten Attribute als Textattribut interpretiert werden. Um die spätere Verwendung zu erleichtern, bietet es sich allerdings an, bereits hier eine Voreinteilung der Attribute vorzunehmen, wenn dabei keine wichtigen Informationen verloren gehen. So kann z.B. ein Feld das einen Link (URL) zu einer Detailseite enthält auch gleich als Link gekennzeichnet werden. Sollte später doch nicht der Inhalt, auf den verwiesen wird verwendet werden, sondern die URL als Text ausgegeben werden, kann diese sehr leicht wieder in Text umgewandelt werden (siehe auch Abschnitte 4.1 und 4.3.2). Oder enthält ein anderes Feld z.B. eine Größenangabe (Breite, Höhe, Dateigröße, ...), kann das dafür verwendete Attribut gleich als Zahlenwert festgelegt werden, was eine spätere Umwandlung erspart.

4.3.1 Typen von Attributen

Bei Abfrage einer Datenquelle werden zunächst die Attribute in ihrer Rohform geliefert, d.h. als jene Attribut-Typen in die sie bei der Erschließung der Quelle eingeteilt wurden. Für die weitere Verwendung kann es sinnvoll sein, den Typ eines Attributes zu ändern. So ist z.B. in einem Textattribut von allen Attribut-Typen am wenigsten strukturelle Information enthalten. D.h. außer des Textes selbst, der ausgegeben werden kann, ist damit keine weitere Steuerung der Ausgabe vorgesehen. Alle anderen Typen ermöglichen, außer der Ausgabe ihres Inhalts, auch die Beeinflussung struktureller Eigenschaften wie z.B. die Größe, Form oder Farbe graphischer Elemente. Damit stellt sich die Frage, wie aus Textattributen andere, mit mehr steuernden Informationen „angereicherte" Typen gemacht werden können bzw. welche Umwandlungen von einem Attributs-Typ in einen anderen sinnvoll und technisch umsetzbar sind. Tabelle 4.1 zeigt, welche Umwandlungen eines Original- zu einem Ziel-Typ sich für uns daraus ergeben haben.

Wie man sieht, können alle unterstützten Attribute uneingeschränkt in ein Textattribut gewandelt werden. Andere Umwandlungen, wie z.B. die eines Links in eine Zahl, machen ganz offensichtlich keinen Sinn und sind deshalb nicht möglich. Eine Umwandlung in die Typen *Bit* und *Set* ist dagegen immer möglich, wenn dazu eine *CASE-Anweisung* verwendet wird. Dies bedeutet, dass eine Einteilung der Ausprägungen des Attributes folgender Form verwendet wird:

```
WENN Attributwert = Wert_11 ODER Wert_12 ODER ... ODER Wert_1A
    DANN Listenwert_1
WENN Attributwert = Wert_21 ODER ... ODER Wert_2B
```

schließlich den Link betrachten, der auf diese Daten verweist und dieser ist wiederum als URL in Textform angegeben.

Original-Typ	*Ziel-Typ* **Text**	**Integer**	**Float**	**Bit**	**Color**	**Set**	**Link**
Text	X	möglich wenn Zahl enthalten	möglich wenn Zahl enthalten	mit CASE	mit CASE	mit CASE	möglich wenn URL enthalten
Integer	immer möglich	X	immer möglich	mit CASE	Farbverlauf oder mit CASE	mit CASE	nicht möglich
Float	immer möglich	immer möglich (ganz-zahliger Anteil)	X	mit CASE	mit CASE	mit CASE	nicht möglich
Bit	immer möglich (FALSE oder TRUE)	möglich (0 oder 1)	möglich	X	mit CASE	mit CASE	nicht möglich
Color	möglich, kaum sinnvoll	kaum sinnvoll	kaum sinnvoll	mit CASE	X	mit CASE	nicht möglich
Set	immer möglich	nicht möglich	nicht möglich	mit CASE	mit CASE	X	nicht möglich
Link	immer möglich	nicht möglich	nicht möglich	mit CASE	mit CASE	mit CASE	X

Tabelle 4.1: Übersicht aller Kombinationen von Typ-Umwandlungen und möglichen Einschränkungen. Mit dem Eintrag *„mit CASE"* ist gemeint, dass mit Hilfe einer *CASE-WHEN-ELSE-Anweisung* in *SQL* alle Ausprägungen des Original-Typs der eingeschränkten Anzahl der gewünschten Zustände des Ziel-Typs zugeteilt werden.

```
    DANN Listenwert_2
:
WENN Attributwert = Wert_Y1 ODER ... ODER Wert_YX
    DANN Listenwert_Y
SONST
    Listenwert_Z
```

Umgesetzt wird diese Funktion mit Hilfe einer CASE-Anweisung wie sie von der Datenbankabfragesprache *SQL* zur Verfügung gestellt wird (für Details siehe http://www.hsqldb.org/doc/1.8/guide/). Mit dieser Methode kann jedes Attribut auf eine begrenzte Zahl von Ausprägungen abgebildet werden. Damit können dann Metapher-Merkmale gesteuert werden, die nur eine (von der graphischen Darstellung) vorgegebene Anzahl von Zuständen annehmen können, wie in unserem Beispiel der Typ eines Buches, also Taschenbuch, gebundene Ausgabe, CD, ... (mehr zum Set-Merkmal findet sich in 3.3 und 4.4.2).

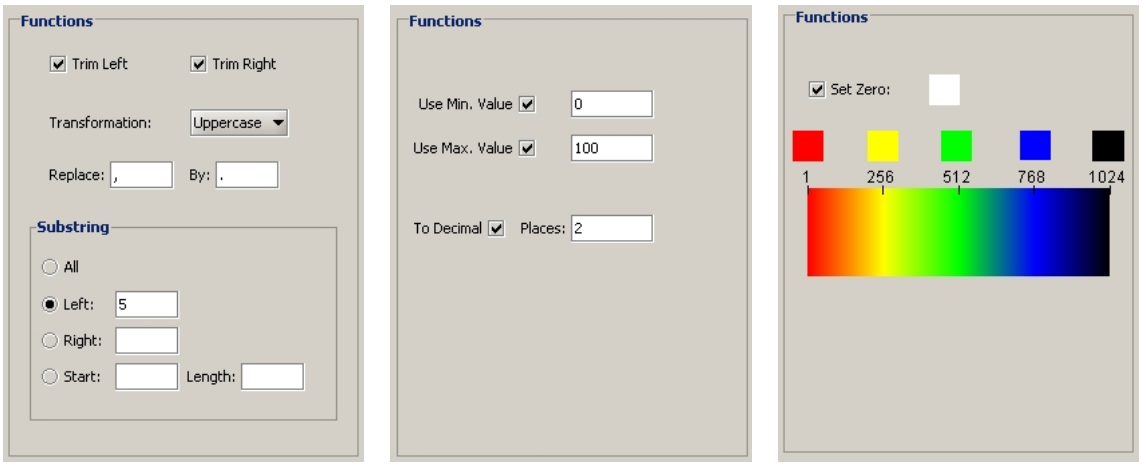
Der Typ *Bit* repräsentiert eine List mit nur zwei möglichen Ausprägungen. So können vereinfacht Wahrheitswerte wie *TRUE* und *FALSE* an die weiteren Verarbeitungsschritte übergeben werden. Die Umwandlung in ein solches Attribut hat damit folgende Form:

```
Wenn Attributwert = Wert_1 oder Wert_2 oder ... Wert_X dann TRUE/FALSE
Sonst FALSE/TRUE
```

Ein Set-Attribut mit zwei Ausprägungen kann diesen Typ natürlich vollständig ersetzen, aber der Umgang mit einem binären Wert ist einfacher und kann speziell zur manuellen Steuerung bestimmter Anzeigeeigenschaften sehr praktisch sein (siehe dazu auch 3.4 und 4.4.2).

Bei der Umwandlung in ein *Color-Attribut* können mit einer CASE-Anweisung einzelne Farben zugeordnet werden. Die Form entspricht dabei jener die auch für das Set-Attribut verwendet wird. Als resultierenden Listenwerte werden hier jedoch Farbcodes angegeben. Der metaViewer bietet Unterstützung bei der Auswahl einer Farbe durch einen *Color-Chooser* der den Farbcode der gewählten Farbe ausgibt. Ist das Original-Attribut vom Typ *Integer*, kann der Wertebereich zwischen 0 und 1024 einem Farbverlauf zugeordnet werden. Dieser kann in der Eingabemaske (siehe Abbildung 4.3.1c) durch Auswahl der Farben für die Werte 0 (bzw. 1), 256, 512, 768 und 1024 angepasst werden.

Der metaViewer unterstützt die mehrmalige Umwandlung des Typs eines Attributes. Dies kann genutzt werden, um nacheinander Operationen unterschiedlicher Attributs-Typen anzuwenden. Ganz allgemein können Operationen beliebig



(a) Textattribut (b) Fließkomma-Attribut (c) Color-Attribut

Abbildung 4.3.1: Beispiele von Eingabemasken für Funktionen die auf Attribute angewendet werden können

geschachtelt und hintereinander ausgeführt werden. Näheres dazu und wie diese Schachtelung von Operationen auf Attributen realisiert wurde, findet sich im folgenden Unterabschnitt.

4.3.2 Attribute anpassen – Abgeleitete Attribute

Wie schon im vorigen Teil beschrieben, kann der Typ eines Attributes geändert werden, wenn dies hilfreich erscheint. Zusätzlich können aber auch, je nach Attribut-Typ, unterschiedlichste Operationen auf die Inhalte des Attributes angewendet werden. Manche dieser Operationen können direkt über Eingabemasken ausgewählt und eingestellt werden. Diese Eingabemasken enthalten für jeden Attribut-Typ eigene Elemente, je nach den Operationen die auf diesen Typ angewendet werden können. Wie einige dieser Bedienelemente aussehen, zeigt Abbildung 4.3.1.

Werden mehrere Operationen in einer Maske ausgewählt, werden diese nacheinander, von oben nach unten, auf die jeweiligen Inhalte des Attributes angewendet. Entfernt man z.B. bei der Anpassung eines Textattributes (siehe 4.3.1a) durch Auswahl der entsprechenden *Trim*-Operation alle führenden Leerzeichen und verwendet vom Text jeweils nur die ersten fünf Zeichen (*Left*-Operation), werden zuerst die Leerzeichen entfernt und dann die führenden fünf Zeichen ermittelt. Bei der Anordnung der Elemente wurde darauf geachtet, dass die dadurch vorgegebene Reihenfolge für die meisten Anwendungen sinnvoll sein sollte. Die folgende Aufzählung gibt einen Überblick über alle Operationen die für jeden Attribut-Typ mittels graphischer Bedienelemente angewendet werden können:

- Text-Attribut
  - *Trim Left* und *Trim Right*: führende bzw. schließende Leerzeichen entfernen
  - *Transformation*: Umwandlung in Groß- oder Kleinbuchstaben
  - *Replace*: Ersetzung einer Zeichenfolge durch eine andere
  - *Substring*: Herausschneiden bestimmter Teile des Textes bestehend aus:
    - *Left*: entfernt die angegebene Anzahl an Zeichen beginnend von links
    - *Right*: entfernt die angegebene Anzahl an Zeichen beginnend von rechts
    - *Start* und *Length*: Auswahl des Textausschnitts der Länge *Length* beginnend mit dem Zeichen an der Position *Start*
- Integer-Attribut
  - *Use Min. Value*: alle Werte die kleiner als der angegebene sind, werden auf diesen gesetzt
  - *Use Max. Value*: alle Werte die größer als der angegebene sind, werden auf diesen gesetzt
- Float-Attribut
  - *Use Min. Value*: wie bei Integer-Attribut
  - *Use Max. Value*: wie bei Integer-Attribut
  - *To Decimal*: wandelt die Fließkommazahl in eine Zahl mit der angegebenen fixen Anzahl an Nachkommastellen und rundet auf die letzte Stelle nach dem Komma
- Set-Attribut
  - *Select Property*: Die Auswahl des Metapher-Merkmales mit dem das Attribut verknüpft werden soll ermöglicht es, dem User eine Vorlage einer CASE-Anweisung (siehe oben) im Feld *Resulting Statement* mit passenden Ergebnis-Ausprägungen zur Verfügung zu stellen. Weitere Anpassungen müssen dann durch Kopie und Bearbeitung dieses Statements im Feld *User Defined Statement* vorgenommen werden.
- Binary-Attribut

  - *If - Then - TRUE/FALSE*: Der im Feld angegebene Wert oder Text (je nach Attribut-Typ) wird mit den einzelnen Ausprägungen des Attributes verglichen. Stimmen sie überein, wird der im Dropdown-Feld gewählte Wahrheitswert ausgegeben, sonst der jeweils andere.

- Link-Attribut
  - keine Anpassung vorgesehen

- Color-Attribut mit Original-Attribut *nicht* vom Typ *Integer*: Auswahl einer Farbe durch eine CASE-Anweisung
  - *Color Chooser*: Mit Hilfe eines Farbauswahl-Fensters kann der zugehörige Farbcode ermittelt werden. Dieser kann dann direkt in der bereitgestellten Vorlage einer CASE-Anweisung als Ergebnis-Ausprägung verwendet werden.

- Color-Attribut mit Original-Attribut vom Typ *Integer*: Zuordnung eines Farbverlaufes (siehe 4.3.1c)
  - *Set Zero*: Bei Auswahl kann die Farbe für den Wert 0 getrennt gewählt werden, andernfalls wird sie durch den Farbverlauf festgelegt
  - *Farbwerte*: Durch Auswahl eines der farbigen Quadrate wird ein Farbauswahl-Fenster aufgerufen mit dem dessen Farbe verändert werden kann. Damit lassen sich die Farben des Verlaufes an den Stellen 0 bzw. 1, 256, 512, 768 und 1024 wählen. Der Farbverlauf wird jeweils an die gewählten Farben angepasst.

Werden diese vorgegebenen Elemente zur Bedienung benutzt, wird aus den getroffenen Einstellungen von der Software automatisch eine entsprechende Befehlskette für die SQL-Abfrage erstellt, die dann auf die Werte des Attributes angewendet wird. Sollten komplexere, durch die Bedienelementen nicht unterstützt Operationen benötigt werden, müssen diese in textueller Form, als Funktionen die auf die Spalten einer SQL-Abfrage angewendet werden, eingegeben werden. Dadurch verlässt man natürlich den Bereich der Bedienung, in dem man „nichts falsch machen" kann und muss sich somit selber z.B. um die syntaktische Korrektheit der Befehle kümmern. Um den Anwender bei der Entwicklung solcher Befehlsketten zu unterstützen, werden auch die Ergebnisse der automatisch generierten Befehle im Feld „*Resulting Statement*" ausgegeben. Diese können als Vorlage dienen oder direkt in das Feld „*User Defined Statement*" kopiert werden, um daraus komplexere Operationen zu entwickeln.

Um die Mächtigkeit der angebotenen Operationen und der Typ-Wechsel zu maximieren, musste ein Konzept entwickelt werden das es erlaubt, Manipulationen an

Attributen wiederholt und in beliebiger Reihenfolge durchführen zu können. Außerdem musste ein Weg gefunden werden, möglichst vielfältige Operationen auf einer großen Anzahl von Einträgen auszuführen. Weiters musste eine Datenstruktur gefunden werden, die eine möglichst flexible Speicherung der Attributs-Daten erlaubt. Obwohl der Typ eines Attributes auch während der Laufzeit geändert werden kann, sowie beliebig viele Attribute unterschiedlichen Typs hinzugefügt werden können, sollte trotzdem der Datentyp in der Datenstruktur festgelegt sein. Eine interne Datenrepräsentation mit solchen Eigenschaften wird von der gewählten Programmiersprache[3] nicht zur Verfügung gestellt und hätte mit großem Aufwand entwickelt werden müssen. Aus dieser Suche nach einer geeigneten Datenstruktur, wurde die Idee entwickelt, zur internen Repräsentation der Attribut-Daten eine Datenbank zu verwenden. Eine Datenbankabfrage mittels *SQL* bietet in der Spalten-Auswahlliste des *SELECT*-Statements die geforderten Möglichkeiten und zusätzliche Vorteile:

1. Eine Vielzahl an Funktionen kann auf alle Elemente (Zeilen in der Tabelle) effizient angewendet werden.

2. Die Funktionen können beliebig geschachtelt werden.

3. Eine Datenbanktabelle bietet die Möglichkeit, typisierte Informationen zu speichern und der Datentyp kann dabei auch erst während der Laufzeit festgelegt werden.

4. Die Befehlskette mit der die Funktionen auf der entsprechenden Spalte ausgeführt werden, bildet eine vollständige Beschreibung der durchgeführten Operationen und kann direkt zur Speicherung und Verwaltung der ausgeführten Manipulationen verwendet werden.

5. Die Abfrage und Ausführung der Funktionen ist so schnell, dass niemals die manipulierten Attribut-Werte gespeichert werden müssen, sondern die Operationen jederzeit erneut, auch auf einer sehr große Anzahl von Elementen, angewendet werden können.

6. Mit der Verwendung von *SQL* steht eine wohl definierte und allgemein zugängliche Programmiersprache zur Verfügung, um auf die Daten zuzugreifen.

Es sollte auch möglich sein, jedes von der Datenquelle gelieferte Attribut mehrfach zu verwenden, also in Varianten mit unterschiedlichen Anpassungen zur Verfügung zu stellen. Das Original-Attribut sollte dabei ebenfalls jederzeit verfügbar bleiben, um es unverändert zu verwenden oder als Basis für weitere angepasste Attribute. Um dies zu ermöglichen, wurde das Konzept des *abgeleiteten Attributes* entwickelt. Dabei

[3] Sun Java 1.6

wird unterschieden zwischen den unveränderten Attributen (*Quell-Attribute*), so wie sie von der Datenquelle geliefert werden und zusätzliche erstellten, sogenannten *abgeleiteten Attribute*. Diese abgeleiteten Attribute werden direkt im metaViewer aus bestehenden Attributen (Quell- oder abgeleitete!) durch Anwendung von Funktionen oder Typ-Umwandlungen generiert. Jedes so erzeugte abgeleitete Attribut kann somit wiederum als Ausgangspunkt für ein weiteres abgeleitetes Attribut dienen, als auch dann schließlich einem Metapher-Merkmal zugeordnet werden. Wenn bisher also von einer Anpassung eines Attributes die Rede war, so war damit eigentlich die Erstellung eines abgeleiteten Attributes gemeint, da das ursprüngliche Attribut unverändert bleibt.

Die Eingabe-Elemente zur Bearbeitung eines abgeleiteten Attributes (siehe Abbildung 4.6.2) können jeweils nur auf das ausgewählte Original-Attribut angewendet werden. Sollten zur Definition eines Metapher-Merkmales zwei oder mehr Attribute notwendig sein[4], kann dies alleine mit den graphischen Bedienelementen nicht durchgeführt werden. Abhilfe schafft die Verwendung des Feldes „*User Defined Statement*". Hier kann jederzeit auf alle vorhandenen Attribute über ihren Namen zugegriffen werden. Daraus können Terme erstellt werden, deren Ergebnis gleichzeitig von mehreren Attributen beeinflusst wird.

4.4 Metaphern

Im Gegensatz zu den Attributen, deren Form und Eigenschaften mehr oder weniger gut angepasst werden können, setzt bei den Merkmalen von Metaphern meist die graphische Darstellung bereits enge Vorgaben. Deshalb wurden die ganzen Manipulationsmöglichkeiten für Attribute geschaffen, um sie an die jeweiligen Bedürfnisse der einzelnen Metapher-Merkmale anpassen zu können. Die interessanten Fragen die sich im Zusammenhang mit den Metaphern stellen, sind zuerst die Wahl der Metapher und anschließend welche Information aus den Quelldaten die einzelnen graphischen Merkmale in welcher Form beeinflussen soll. Dies hat dann wiederum Einfluss auf die zu erstellenden abgeleiteten Attribute.

Wie man aus dieser Überlegung heraus leicht sehen kann, ergibt sich für die Anwendung des metaViewers kein strikt lineares Vorgehen beim Entwurf einer neuen Visualisierung, etwa entlang des erwähnten Datenflusses. Vielmehr wird ein Zyklus von der Erstellung abgeleiteter Attribute über die Zuweisung zu Metapher-Merkmalen bis zu daraus resultierenden neuen Anforderungen an weitere Attribute mehrfach durchlaufen werden müssen, um zum gewünschten Resultat zu gelangen[5].

[4] Dies kann z.B. sinnvoll sein, wenn Texte aus zwei Attributen in einem Metapher-Merkmal ausgegeben werden sollen, oder wenn zwei Attribute Bedingungen für die Darstellung eines Merkmals vorgeben.

[5] Sollte so kein befriedigendes Resultat erzielt werden, könnten auch Anpassungen an den Program-

Dabei spielen neben den logischen auch kreative Überlegungen eine entscheidende Rolle. Die Möglichkeiten des metaViewers sollen diesen kreativen Prozess möglichst gut unterstützen.

Im Diagramm von Abbildung 4.1.1 befinden wir uns nun an der unteren Schnittstelle und dringen in den Metapher-Bereich vor.

4.4.1 Die Wahl der Metapher

Die richtige Wahl einer graphischen Metapher zur Visualisierung von Daten ist ein offenes und noch weitgehend ungelöstes Problem. Auch wenn im metaViewer die Schnittstelle zur graphischen Darstellung möglichst klar definiert wurde, bleibt die Erstellung weiterer graphischer Darstellungsmöglichkeiten mit neuen Metaphern, trotz der Unterstützung die der metaViewer dabei bieten kann, noch relativ aufwändig. Deshalb konzentrierten wir uns in dieser Arbeit vorrangig auf die Darstellung der Suchergebnisse als Bücher, da bei der Entwicklung des libViewers bereits gute Erfahrungen damit gemacht wurden. Diese Darstellung wurde optimiert und durch Animationen noch aufgewertet, um die Möglichkeiten des metaViewers aufzuzeigen.

Eine weitere Variante der Darstellung die eine Darstellung von Planeten in einem Planetensystem als Metaphern verwendet wurde zusätzlich implementiert, jedoch wurde hierbei viel weniger Aufwand betrieben und nur die grundlegenden Merkmale definiert. Diese graphische Variante wurde erste nachträglich erstellt und zeigt somit mehr die Vorgehensweise bei der Erweiterung, als die graphischen Fähigkeiten. Trotz des geringen Aufwands der dabei getrieben wurde, lassen sich die vielfältigen Möglichkeiten des metaViewers auch bei dieser Darstellung bereits erkennen (siehe Abbildung 4.6.5b).

4.4.2 Arten von Metapher-Merkmalen

Durch die Wahl der Metapher werden auch die Merkmale vorgegeben mit denen das Aussehen einer Ausprägung dieser Metapher beeinflusst werden kann. Bei unserer Parade-Metapher, dem Buch, erhält man dabei ganz offensichtliche Merkmale wie Größe (in allen Dimensionen und auch einzelner Teile), Position und Lage (im Raum bzw. auf dem Bücherregal), Farbe (Aufdrucke, Texturen, Oberfläche, ...) oder Beschriftung (aufgedruckte Texte). Diese Merkmale finden sich, mehr oder weniger, bei allen Metaphern, die ein Objekt im 3-dimensionalen Raum repräsentieren. Weitere denkbare Merkmale wären z.B. Links zu Web-Seiten oder Multimediadateien. Diese müssen zwar nicht direkt sichtbar sein, könnten aber durch Anklicken weitere Informationen liefern.

---

mierschnittstellen des metaViewers notwendig werden. Z.B. durch Erweiterung bzw. Anpassung der Schnittstelle zur Datenquelle und somit der Quell-Attribute oder durch Änderungen an Eigenschaften der graphischen Darstellung. Dazu mehr in Kapitel 5.

Die Darstellung unterschiedlicher Typen einer Metapher wäre auch denkbar. In unserem Beispiel könnte man z.B. zwischen Taschenbüchern, gebundenen Ausgaben oder Ringmappen unterscheiden, oder wenn man die Bezeichnung „Buch" etwas weiter fasst, CD-Hüllen und Videokassetten als Darstellung wählen. Solch ein Metapher-Merkmal mit dem Namen *Binding* wurde im metaViewer bei der Buch-Metapher umgesetzt.

Die Identifikation der Merkmale die bei der Darstellung variiert werden können, muss für jede neu gewählte Metapher gesondert erfolgen. Jede der gefundenen Merkmale wird dabei einer der Merkmals-Typen aus Abschnitt 3.3 zugeordnet.

Die meisten dieser Kategorien konnten wie geplant realisiert werden. Für die hier umgesetzten Visualisierungen wurden keine Felder mit vollständigen Datums- bzw. Zeitangaben benötigt. Die in Frage kommenden Merkmale liegen entweder als Text (z.B. verwendet für die Ausgabe des Erscheinungsjahres eines Buches) oder als Zahl (z.B. für die Zuweisung des Alters einer Datei in Tagen) vor. Mit den Methoden die zum Editieren von Text- und Zahlen-Attributen zur Verfügung stehen, lassen sich die gewünschten Informationen in das geforderte Format bringen und dann dem entsprechenden Merkmal zuordnen. Ein Attribut das speziell zur Verarbeitung von Zeit- und Datums-Formaten dient könnte durchaus hilfreich sein. Auf Grund der Komplexität die sich durch unterschiedliche Formate und der Vielzahl an möglichen Operationen ergibt, wurde in dieser ersten Version des metaViewers auf die Umsetzung solch eines Attributs bzw. Merkmals verzichtet.

Mit *Link-Merkmalen* können alle Arten von verlinkten Informationen die als URL angegeben sind verarbeitet werden. Der metaViewer bietet dabei keine Unterstützung, um die Zuweisung auf Links zu beschränken, die auf den gewünschten Typ von Inhalt verweisen. Wird also z.B. einem Merkmal das ein Bild erwartet ein Link zu einer Webseite oder einem Audio-File zugewiesen, führt das im metaViewer zu unerwartetem Verhalten. Es wird somit dem Anwender überlassen, bei Links auf eine passende Zuordnung zu achten. Hier könnte in Zukunft eine bessere Unterstützung für die verschiedenen Arten von verlinktem Inhalt angeboten werden.

Die Möglichkeiten eines Boolean-Merkmals zeigt die *autoSize-Einstellung* bei der Bücherregal-Umsetzung im metaViewer. Hier handelt es sich streng genommen nicht um ein Merkmal der Metapher sondern um Steuerinformationen, allerdings kann diese Einstellung auf die gleiche Weise wie jedes andere Merkmal angesprochen werden. Setzt man *autoSize* auf *TRUE*, wird bei Vorhandensein eines Bildes auf dem Buchumschlag die Höhe und die Breite des Buches automatisch an das Seitenverhältnis des Bildes angepasst, um eine etwaige Verzerrung der Abbildung zu vermeiden. Wird *FALSE* angegeben, werden die Werte der den entsprechenden Merkmalen zugewiesenen Attribute unverändert übernommen. Durch die Umsetzung dieses Parameters als „normales" Merkmal, kann diese Einstellung für jedes Buch getrennt erfolgen und könnte somit auch von der Ausprägung eines weiteren Attributes abhängig gemacht

werden. Diese Methode bietet vielfältige Unterstützung, um bei der Entwicklung von Metaphern unterschiedliche Einstellmöglichkeiten vorzusehen.

Bisher wurden die Kategorien die für Metapher-Merkmale definiert wurden eher abstrakt behandelt. Im Gegensatz zu Attributen, deren Typ auch die Auswahl der zur Verfügung stehenden Bearbeitungsfunktionen bestimmt, legt der Merkmals-Typ nur die Art der Werte fest, die zugewiesen werden können. Im metaViewer wurden Merkmale durch konkrete Java-Typen umgesetzt. Die Frage der möglichen Werte die einem Merkmals-Typ zugeordnet werden können, reduziert sich also auf den Gültigkeitsbereich des verwendeten Variablen-Typs. Deshalb wird bei der Definition eines Merkmales direkt der gewünschte Java-Typ angegeben und die Kategorie des Merkmales wird daraus ermittelt (siehe auch 5.5.5). Daraus ergeben sich die bisher implementierten Merkmals-Kategorien mit den zugehörigen Java-Typen bzw. -Klassen wie folgt:

Text-Merkmal: Klasse *String*

Integer-Merkmal: Typ *int* oder Klasse *Integer*

Float-Merkmal: Typ *float* oder Klasse *Float*

Bit-Merkmal: Typ *boolean* oder Klasse *Boolean*

Link-Merkmal: Klasse *URL*

Farb-Merkmal: Klasse *Color*

List-Merkmal: Klasse *Enum*

Der Enum-Typ des List-Merkmals muss mit den gewünschten Ausprägungen bei der Implementierung des Merkmals definiert werden. Attribut-Werte werden bei der Zuweisung nicht direkt in diesen Typ gewandelt sondern als String übergeben und erst dann daraus auf das entsprechende Element der Enum-Liste geschlossen.

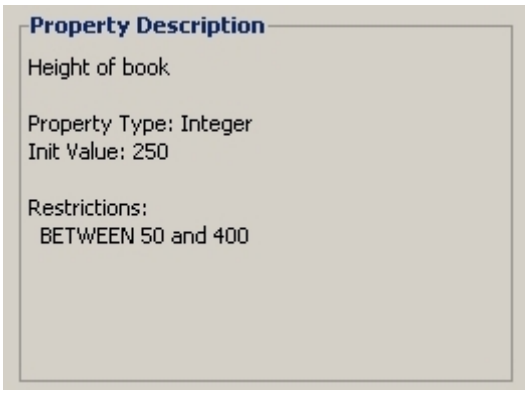
4.4.3 Zuordnungsmöglichkeiten von Attributen zu Metapher-Merkmalen

In Abschnitt 3.4 wurde behandelt, welche Bedingungen bei der Zuordnung von Attributen zu Metapher-Merkmalen gelten müssen. Der metaViewer muss sicherstellen, dass diese Bedingungen eingehalten werden. Wie dies im metaViewer umgesetzt wurde ist in Abschnitt 4.6.4 genauer beschrieben.

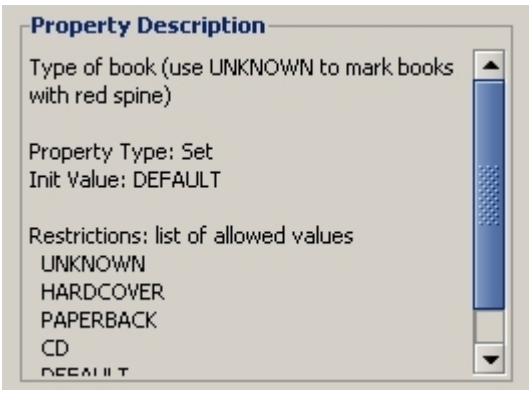
Durch die Beschränkung der möglichen Zuordnung von Attributen zu Metaphern auf den jeweils gleichen Typ, wird schon auf viele Einschränkungen Rücksicht genommen, die sich aus den Eigenschaften der Metaphern ergeben. Allerdings können

sich bei einzelnen Metaphern weitere Vorgaben an die Ausprägungen der Attributs-Werte ergeben. Diese können sehr unterschiedlicher Natur sein und sollen gewährleisten, dass den Metaphern keine Werte zugewiesen werden können, die zu einem ungültigen Zustand führen. Um diese Einschränkungen im metaViewer umzusetzen, wurde bei der Definition von Metaphern die Möglichkeit geschaffen, *Restriktionen* anzugeben. Diese werden während der Erstellung einer neuen Metapher bei der Definition eines Merkmals diesem als Eigenschaft hinzugefügt. Wird nun dem Merkmal ein Attributs-Wert zugeordnet, der den Einschränkungen nicht genügt, wird je nach Restriktion der Wert so korrigiert, dass er im gültigen Bereich liegt. Es können zu einem Merkmal mehrere Restriktionen angegeben werden die während der Ausführung nacheinander zur Anwendung kommen. Welche Restriktionen bisher implementiert wurden und wie deren Korrekturen aussehen zeigt folgende Auflistung:

- Text-Merkmale
  - MIN_LENGTH: unterschreitet der Text die angegebene minimale Länge, wird der Defaultwert zugewiesen.
  - MAX_LENGTH: längere Texte werden hinten auf die maximale Länge beschnitten.
  - NOT_EQUAL: Texte die dem hier angegebenen entsprechen, werden durch den Defaultwert ersetzt.
- Integer- und Float-Merkmale
  - NOT_ZERO: Null-Werte werden durch Defaultwert ersetzt
  - NOT_NEGATIVE: negative Werte werden durch Defaultwert ersetzt
  - NOT_EQUAL: einzelner Wert wird ausgeschlossen in dem er durch Defaultwert ersetzt wird
  - MIN_VALUE: Werte die kleiner als die angegebene Untergrenze sind, werden auf diesen Grenzwert gesetzt.
  - MAX_VALUE: größere Werte werden auf den Maximalwert abgeändert
  - BETWEEN: Werte die außerhalb des angegebenen Bereichs liegen, werden durch den näher liegenden Grenzwert ersetzt.
  - NOT_BETWEEN: Werte in diesem Bereich werden durch Defaultwert ersetzt
- Link- und Color-Merkmale
  - NOT_EQUAL: eine einzelne URL bzw. Farbe kann ausgeschlossen d.h. durch den Defaultwert ersetzt werden



(a) Das Integer-Merkmal für die Höhe des Buches wird defaultmäßig auf 250 gesetzt und erlaubt Werte zwischen 50 und 400 (Größenangaben in mm).



(b) Der Typ des Buches wird durch ein Set-Merkmal bestimmt. Die möglichen Ausprägungen werden als Restriktionen angezeigt.

Abbildung 4.4.1: Beispiele der angezeigten Beschreibung von Metapher-Merkmalen (Properties), jeweils bestehend aus einem Hinweis, welche Eigenschaft der Metapher betroffen ist, dem Typ, dem verwendeten Defaultwert (Init Value) und einer Auflistung von Restriktionen die angewendet werden.

- Binary- und Set-Merkmale
  - Diesen Merkmalen können keine Restriktionen zugeteilt werden. Ihre Einschränkungen sind Teil ihrer Definitionen selbst. Binary-Merkmale können nur die Ausprägungen *TRUE* oder *FALSE* annehmen. Für Set-Merkmale wird bei ihrer Definition ein Enum-Typ angegeben. Die Elemente dieses Aufzählungs-Typs bilden die erlaubten Ausprägungen für das Set-Merkmal. Unerlaubte Werte werden durch den Defaultwert ersetzt, zusätzlich wird jedoch eine Fehlermeldung ausgegeben.

Eine Auflistung der Restriktionen die für eine Metapher definiert wurden, wird automatisch bei der Anzeige der Beschreibung zu einer Metapher, im Feld *Property Description*, mit ausgegeben (siehe Abbildung 4.4.1).

Auch die Unterstützung von Restriktionen wurde so umgesetzt, dass die Erweiterung um zusätzliche Restriktionen problemlos möglich ist. Allerdings muss bei Implementierung einer neuen Restriktion doch etwas tiefer in den Programmcode des metaViewers eingegriffen werden, als dies bei vielen anderen Schnittstellen der Fall ist. Das Verhalten im Falle, wenn der Wert einer Restriktion nicht gerecht wird muss festgelegt und dabei die Wechselwirkung mit anderen Restriktionen desselben Merkmals-Typs beachtet werden (Näheres zur Implementierung siehe 5.5.6).

4.5 Graphische Darstellung

Für die graphische Darstellung der Metaphern wurde die Programmbibliothek Piccolo2D (siehe 5.1) eingesetzt. Sie bietet die Möglichkeit eine große Anzahl von 2D Objekten zu erstellen, zu verwalten und darzustellen. Durch Gruppierung und Modifikation solcher Objekte, ist es möglich, den Eindruck zu erwecken, als handle es sich um die Abbildung dreidimensionaler Objekte. Zusätzlich bietet Piccolo die Möglichkeit, Objekte bewegt darzustellen, was den Eindruck es mit "realen" Objekten zu tun zu haben, noch verstärkt. Auf diese Weise werden z.B. Bücher aus Flächen gebildet, deren Erscheinungsbild über Parameter gesteuert werden kann. Mit diesen Methoden wurde eine Darstellung dreidimensionaler Bücher erstellt. Mit Piccolo2D kann auch Bewegung umgesetzt werden. So können Bücher beim darüber fahren mit dem Mauszeiger bzw. beim Anklicken aus dem Regal herausbewegt bzw. auch geöffnet werden (siehe Abbildung 4.6.4d).

Der libViewer beschränkte sich bei der Wahl der Quelldaten auf Abfrageergebnisse von Online Bibliotheken. Durch die Ähnlichkeit der so erhaltenen Einträge und der gewählten Metapher, ergaben sich die Wahl der Merkmale und deren Zuordnung nahezu von selbst. Die Bezeichnungen einiger Merkmale wie *Titel*, *Autor* oder *Verlag* zeigen, wie eng (und damit fixiert) die Verknüpfung zwischen Attributen in den Metadaten und den graphischen Metapher-Merkmalen war.

Da im metaViewer Metapher-Merkmale von den Metadaten-Feldern grundsätzlich getrennt behandelt werden, wurde z.B. aus Bezeichnungen wie *Titel*, *Autor* und *Verlag* jeweils ein Metapher-Merkmale vom Typ *Text*. Wird einem dieser Merkmale nun ein Text zugeordnet, erhält dieser, je nach Merkmal, eine bestimmte Position am Buch. Diese Position unterscheidet die einzelnen Text-Merkmale voneinander und sollte sich auch in ihrem Namen widerspiegeln (z.B. *footerFront*, *textSpine* oder *headerInside*). Zu jedem Merkmal liefert eine Beschreibung nähere Informationen dazu, wo und wie der Text platziert wird, um das Zuweisen zu Merkmalen zu erleichtern. Zusätzlich zu ihrer Position besitzen einzelne Merkmale weitere Eigenschaften bzw. Einschränkungen. So bestehen z.B. *textSpine* und *headerInside* jeweils nur aus einer Zeile Text, was ihre Verwendung beschränkt. Dies wird ebenfalls als Information während der Zuweisung angezeigt. Wie solche Restriktionen aussehen können und wie sie beim Definieren von Metapher-Merkmalen festgelegt werden, wurde im vorigen Abschnitt bereits genauer beschrieben.

Obwohl die fixe Verbindung zwischen Datenquelle und Darstellung erst nachträglich aufgebrochen wurde, stellte sich die Entscheidung, eine vorhandene, gut erprobte Darstellung als Vorlage zu verwenden, doch als positiv heraus. Durch die Implementierung einer funktionierenden Darstellung und der, vorerst fixen, Verknüpfung mit den Quelldaten, ließen sich viele Details erkennen, die auch bei der weiteren Entwicklung beachtet werden mussten. So stammen viele allgemeine Prinzipien, die später auf die Entwicklung des Zuordnungsprozesses Einfluss nahmen,

aus den Erfahrungen, die bei der Implementierung dieser ersten Version gemacht wurden. Einige dieser Erkenntnisse waren z.B.: Bilder in den Metadaten können in unterschiedlicher Auflösung vorliegen, man kann sich bei Datumsangaben kaum auf ein fixes Format verlassen, Datenfelder können sehr unregelmäßig befüllt sein bzw. dieselbe Information kann abwechselnd in unterschiedlichen Feldern vorliegen, usw. All diese Erfahrungen hatten Einfluss auf die Anforderungen die an die Manipulationsmöglichkeiten von Attributen gestellt wurden (siehe 4.3.2).

Erst nachdem die meisten Funktionen des metaViewers bereits fertig implementiert waren, wurde eine weitere Darstellung erstellt. Sie verwendet Planeten als Metapher und stellt diese in einem Sonnensystem dar. ...

Wie man an der Implementierung der Planetenmetapher sieht, können viele der hier gefunden Erkenntnisse direkt auf beliebige andere Darstellungsformen angewendet werden. Dabei sind bei der Art der Darstellung der Phantasie keine Grenzen gesetzt. Es könnten an Stelle von Büchern oder Planeten z.B. auch Brücken, Gebäude, Fahrzeuge, Pflanzen, usw. gewählt werden. Vor allem Dinge, deren optische Repräsentation allgemein geläufig ist und die eine ausreichende Bandbreite an Variationsmöglichkeiten anbieten, scheinen hierzu geeignet.

Im Diagramm aus Abbildung 4.1.1 befinden wir uns nun am unteren Ende. Der Anwender hat nun die Möglichkeit, die Anzeige direkt zu beeinflussen (zoomen, scrollen, evtl. Metaphern durch klicken hervorheben, ...) oder die Zuordnungsinformationen erneut bearbeiten. Bis das gewünschte Ergebnis vorliegt müssen im Allgemeinen mehrere Zyklen durchlaufen werden. Das Fluss-Diagramm bezieht sich dabei zwar auf die Bedienung des metaViewers, aber diese Erkenntnis kann auch treffend auf den Entwicklungsprozess des metaViewers angewendet werden. Einige Zyklen wurden bereits durchlaufen und wenn in folgenden Projekten neue Ideen mit Hilfe des metaViewers umgesetzt werden sollten, werden weitere folgen.

4.6 Bedienung und Ergebnisse

Nicht nur die graphische Darstellung an sich, sondern auch die Bedienung der Applikation an aktuelle Standards anzupassen, war eine Vorgabe für diese Arbeit. Einige der dabei umgesetzten Funktionalitäten sind:

- Verwaltung mehrerer Fenster
- Stufenlose Vergrößerung und Verkleinerung der Ergebnisgraphik
- Übersichtsfenster (Birdseye View) mit der die Ansicht der graphischen Ausgabe gesteuert werden kann
- Gestaltung der Bedienung möglichst intuitiv

- Speichern und laden von Suchergebnissen
- Speichern und laden von Attribut-Definitionen und Zuordnungen
- Dynamisch steuerbare Anpassung der Darstellung

Zusätzlich zu diesen geforderten Funktionen bietet die Grafik-Schnittstelle die Möglichkeit, Interaktionen mit den dargestellten Ergebnissen zu implementieren. So können z.B. Bücher durch darüber fahren mit dem Mauszeiger aus dem Regal heraus bewegt, durch einen Mausklick mit der Vorderseite nach vorne gedreht und durch einen weiteren Mausklick geöffnet werden. Dies soll den intuitiven Umgang fördern und weitere Möglichkeiten aufzeigen, die eine solche graphische Darstellung gegenüber der üblichen Präsentation als Liste bietet. Im Folgenden werden die Bedienungsmöglichkeiten für die verschiedenen Aufgaben des metaViewers erläutert.

4.6.1 Aufrufparameter

Manche der Möglichkeiten des metaViewers werden über Eingabeparameter gesteuert. Durch Aufruf von *„metaViewer.jar -h"*[6] werden diese angezeigt. Folgende Parameter können beim Start des metaViewers angegeben werden:

-db, –dbServer Im Standardfall (ohne Angabe dieser Option) wird für die interne Repräsentation der Daten eine Datenbank im Speicher aufgebaut. Diese kann von anderen Applikationen nicht abgefragt werden. Während der Entwicklung könnte dies jedoch sehr hilfreich sein, deshalb wird bei Angabe dieser Option zusätzlich ein Datenbankserver erstellt, der den Zugriff auf die Tabellen, z.B. für Datenbank-Tools, ermöglicht. Der Nachteil bei Verwendung des Servers ist, dass er von allen, zur gleichen Zeit gestarteten Instanzen des metaViewers verwendet wird. Beim Beenden einer dieser Instanzen wird er gestoppt und damit steht die Datenbank auch den anderen, noch laufenden Instanzen des metaViewers, nicht mehr zur Verfügung. Damit können diese dann nicht mehr verwendet werden und müssen neu gestartet werden.

-h, –help zeigt die verwendbaren Parameter an

-l, –load <filename> Datei mit gespeicherten Suchergebnissen wird beim Start geladen

-o, –open <filename> gespeicherte Attribut-Definitionen und Zuordnungsdaten werden beim Start eingelesen und angewendet

-p, –proxy <proxy:port> Proxy-Server für Zugriff auf Datenquellen

[6]Der metaViewer benötigt eine installierte Java Laufzeitumgebung ab Version 6.

Keiner dieser Parameter muss beim Start des metaViewers angegeben werden. Soll der Zugriff auf das Internet allerdings über einen Proxy-Server erfolgen, kann dies nicht nachträglich eingestellt werden und muss deshalb bereits beim Aufruf angegeben werden. Auch der Start eines DB-Servers an Stelle der reinen Speicherlösung ist während der Laufzeit nicht mehr möglich und muss bereits beim Start ausgewählt werden. Im Gegensatz dazu kann das Laden einer Ergebnis-Datei als auch das Öffnen einer gespeicherten Zuordnung jederzeit auch innerhalb der Applikation erfolgen.

4.6.2 Hauptansicht: Suche und Darstellung der Ergebnisse

Wird der metaViewer ohne Angabe von Parametern (siehe vorigen Abschnitt) gestartet, sieht man zunächst hauptsächlich leere Fenster. In der Werkzeugleiste, ganz oben unter der Menüleiste, kann man mit dem Dropdown-Menü die gewünschte Datenquelle auswählen, von der die Ergebnisse abgefragt werden sollen. In unserem Beispiel wird die Suchanfrage an Amazon Deutschland gesendet und dabei die Kategorie „Books" abgefragt. Mit dem zweiten Dropdown-Menü kann die Art der Darstellung gewählt werden. Wir verwenden hier unsere altbewährte Darstellung des Bücherregals.

Wird nun in das Textfeld ein Suchtext von mindestens drei Zeichen Länge eingegeben und anschließend *Search* gedrückt, wird die Anfrage an die gewählte Quelle gesendet und die Ergebnisse dargestellt (Abbildung 4.6.4a). Im Fenster *„Result Tree"* werden die einzelnen Ergebnisse der Abfrage in einem Baum aufgelistet. Dies erleichtert die Untersuchung einzelner Datensätze und kann auch bei der Erschließung neuer Datenquellen sehr hilfreich sein. Das größte Fenster „*Visualization*" zeigt die graphische Ausgabe der Ergebnisse. In unserem Beispiel sehen wir ein Bücherregal mit lauter identischen, leeren Büchern, dabei entspricht jedes Buch einem Eintrag aus den Ergebnissen. Da wir bisher keine Zuordnung von Attributen zu graphischen Merkmalen durchgeführt haben, wird für jedes Metapher-Merkmal ein vordefinierter Wert angenommen, d.h. keine Informationen aus den Metadaten werden für die Darstellung verwendet. Das Fenster „*Overview*" zeigt eine Übersicht der graphischen Darstellung mit einer Markierung des gerade sichtbaren Bereichs. Durch bewegen dieser Markierung kann der dargestellte Bereich im Ausgabefenster gesteuert werden.

Bewegt man den Cursor über die Bücher, bewegen sich diese aus dem Bücherregal heraus und können durch einen Mausklick mit der Vorderseite nach vorne gedreht werden. Dabei wird auch der diesem Buch zu Grunde liegende Eintrag im Ergebnisbaum (*Result Tree*) angezeigt, indem der entsprechende Zweig aufgeklappt wird.[7]

[7] Dies kann sowohl beim Erschließen neuer Datenquellen oder beim Entwurf neuer Ausgabearten aber auch während dem „normalen" Arbeiten mit dem metaViewer sehr hilfreich sein.

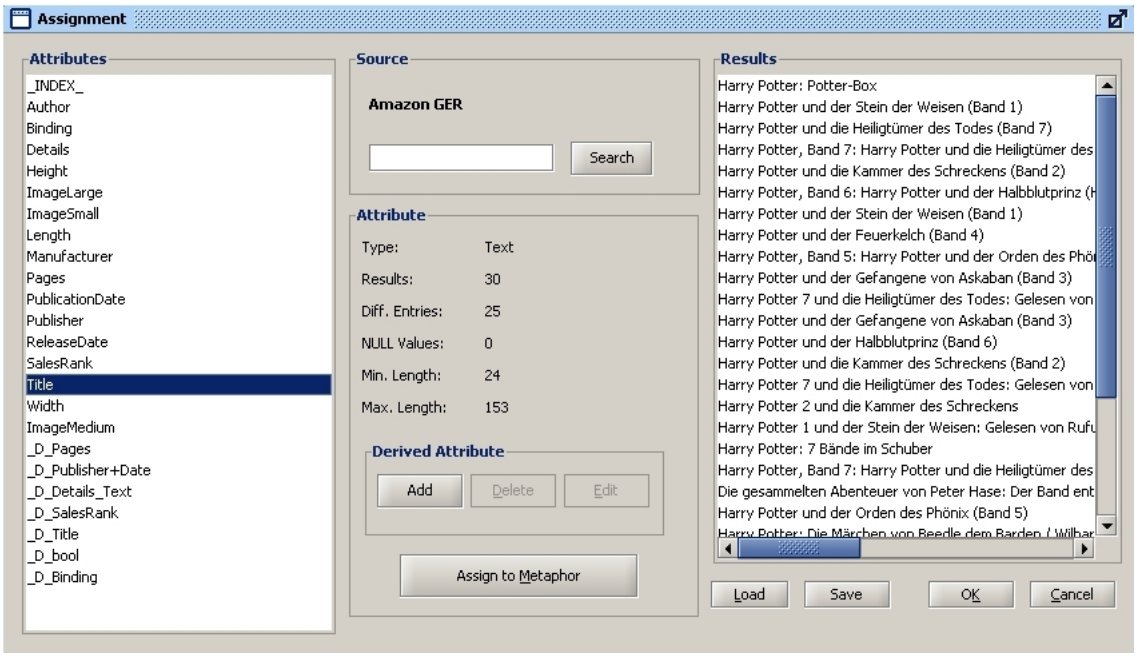


Abbildung 4.6.1: Assignment-Fenster

Ein erneuter Mausklick öffnet das Buch (in diesem Zustand befindet sich gerade das Buch im Zentrum in Abbildung 4.6.4a) und ein Klick mit der rechten Maustaste befördert das Buch wieder zurück in das Regal. Alle Flächen des Buches, ob innen oder außen, sind jedoch vorläufig noch leer. Interessanter wird das Ganze erst, wenn wir den graphischen Metapher-Merkmalen passende Attribute zuordnen. Wie das geht, wird in den Abschnitten 4.6.3 und 4.6.4 gezeigt.

4.6.3 Hinzufügen und editieren von Attributen

Nachdem man eine Datenquelle und eine Darstellungsart ausgewählt und Suchergebnisse abgefragt bzw. eingelesen hat, kann man daran gehen, passende Attribute zu erstellen. Dies geschieht durch die Schaltfläche *Assignment* in der Werkzeugleiste oder den entsprechenden Menüpunkt im Menü *Edit*. Damit öffnet sich das Fenster mit dessen Hilfe sich Attribute erstellen und bearbeiten lassen, wie in Abbildung 4.6.1 gezeigt.

Im linken Teil, unter *Attributes*, wird eine Liste aller von der Datenquelle zur Verfügung gestellten Attribute dargestellt. Dabei wird für jede Datenquelle ein Attribut mit dem Namen *_INDEX_* hinzugefügt. Dies enthält eine Nummerierung aller Ergebniseinträge und wird intern verwendet um eine eindeutige Verbindung zwischen den Ergebnissen und den dargestellten Metaphern herzustellen, kann aber auch wie jedes andere Attribut verwendet werden. Die weiteren Attribute werden durch die

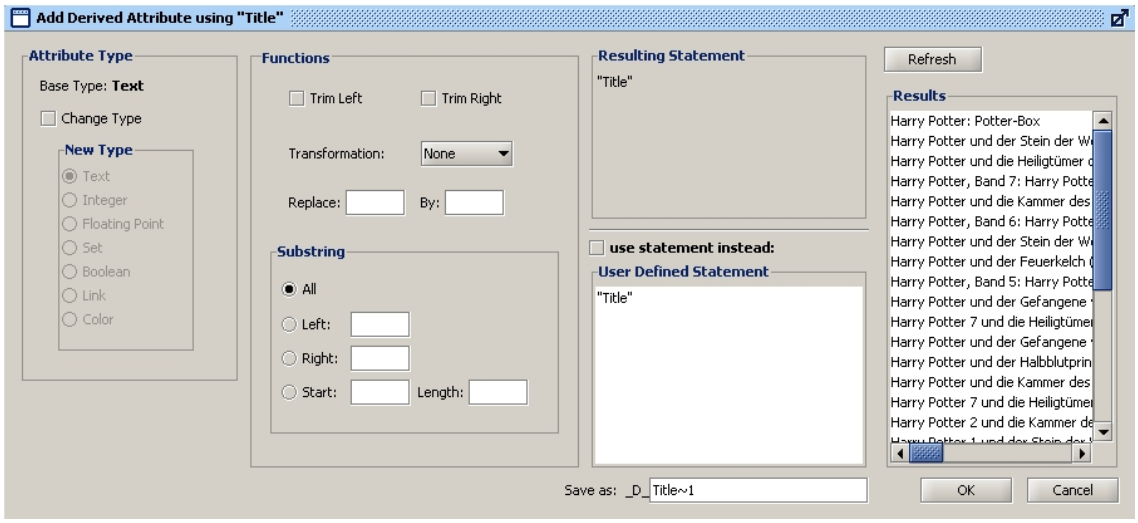


Abbildung 4.6.2: Bearbeiten eines abgeleiteten Attributes

Quelle vorgegeben und wie man sieht, sind in unserem Beispiel typische Attribute wie Autor, Titel oder Größenangaben vorhanden, wie sie bei einer Abfrage von Büchern von Amazon zu erwarten sind. Wählen wir eines dieser Attribute aus, sehen wir im rechten Teil des Fensters, unter *Results*, die Ausprägungen dieses Attributes für die durchgeführte Abfrage. Man sieht also z.B. die Titel der Bücher wie sie von Amazon bereitgestellt werden. Hätten wir vor öffnen des Assignment-Fensters keine Suchanfrage durchgeführt, würde hier nichts angezeigt werden. Dann könnte in dem Suchfeld unter *Source* auch nachträglich noch eine Suche gestartet werden, um die Ergebnisse in den einzelnen Attributen untersuchen zu können.

Bei ausgewähltem Attribut werden im mittleren Feld unter *Attribute* zusätzliche Informationen angezeigt, die für die weitere Verwendung nützlich sein können. Die wichtigste Information ist dabei sicherlich der Typ des Attributes, da dieser bestimmt, welchen Metapher-Merkmalen es später zugeordnet werden kann. Auch die Angaben der leeren Ausprägungen oder der Anzahl verschiedener Ausprägungen des Attributes in dieser Abfrage können von Interesse sein. Je nach Attribut-Typ wird dann noch die minimale und maximale Länge (z.B. von Texten) bzw. der kleinste und der größte vorkommende Wert (alle Arten von Zahlen-Attributen) angezeigt.

Die Bearbeitung eines Attributes erfolgt mit den Schaltflächen in *Derived Attribute*. Ist ein bereits vorhandenes Attribut ausgewählt, kann mit *Add* ein neues, abgeleitetes (= *derived*), Attribut erstellt werden, wobei das ausgewählte als Basis verwendet wird. Je nach Typ des Attributes können dabei unterschiedliche Funktionen ausgewählt werden (siehe 4.3.2). In Abbildung 4.6.2 sieht man das Bearbeitungs-Fenster wenn ein neues Attribut basierend auf *Title*, also einem Attribut vom Typ *Text*, erstellt bzw. bearbeitet wird. Das Fenster lässt sich grob in vier Bereiche un-

terteilen, diese erfüllen folgende Aufgaben:

Attribute Type In diesem Teil kann der Typ des Attributes geändert werden. Wird das Feld „*Change Type*" ausgewählt, kann ein neuer Typ für dieses Attribut ausgewählt werden. Je nach Auswahl ändern sich auch die Operationen die im Bereich *Functions* zur Verfügung stehen.

Functions Hier können, je nach Attribut-Typ, unterschiedliche Operationen ausgewählt werden, die auf die einzelnen Ergebnis-Felder dieses Attributs angewendet werden.

Statement-Bereich Im Fenster „*Resulting Statement*" wird das Statement angezeigt, das sich aus den gewählten Funktionen ergibt, die in *Functions* ausgewählt wurden. Dabei werden Änderungen die in *Functions* vorgenommen werden, erst nach einem *Refresh* übernommen.. Bei Auswahl des Feldes „*use statement instead:*", wird das frei definierbare Statement aus „*User Defined Statement*" an dessen Stelle verwendet (siehe dazu auch Abschnitt 4.3.2 und http://www.hsqldb.org/doc/1.8/guide/).

Results In dieser Liste werden, wie schon im Assignment-Fenster, die Ausprägungen des gewählten Attributes ausgegeben. Nach Änderungen des angewendeten Statements (also durch bearbeiten der *Functions* bzw. des Feldes „*User Defined Statement*"), können die Auswirkungen durch Betätigung von *Refresh* direkt hier Angezeigt werden.

Im Textfeld neben „*Save as:*" kann dem neuen Attribut ein Name zugewiesen werden. Ein eindeutiger, aus der Bezeichnung des Original-Attributes generierter Name, wird vorgeschlagen, kann aber durch den Anwender geändert werden. An dieser Stelle wird auch angedeutet, dass dem Namen vom System die Kennung „_D_" vorangestellt wird. Dies ermöglicht es, in der Liste der Attribute, vom Anwender erzeugte, also abgeleitete (*Derived*), Attribute von solchen zu unterscheiden, die von der Datenquelle vorgegeben wurden (Quell-Attribute). Abgeleitete Attribute können, im Gegensatz zu Quell-Attributen, im Assignment-Fenster mit *Edit* weiter bearbeitet und mit *Delete* wieder gelöscht werden. Jedes so erstellte Attribut kann jedoch auch wiederum als Basis für ein weiteres Attribut verwendet werden. Aus diesem Grund muss beim Löschen eines abgeleiteten Attributes bedacht werden, dass auch alle weiteren Attribute die von diesem abgeleitet wurden gelöscht werden. Deshalb wird beim Löschen eines solchen Attributes vom System eine Bestätigung des Löschvorganges verlangt.

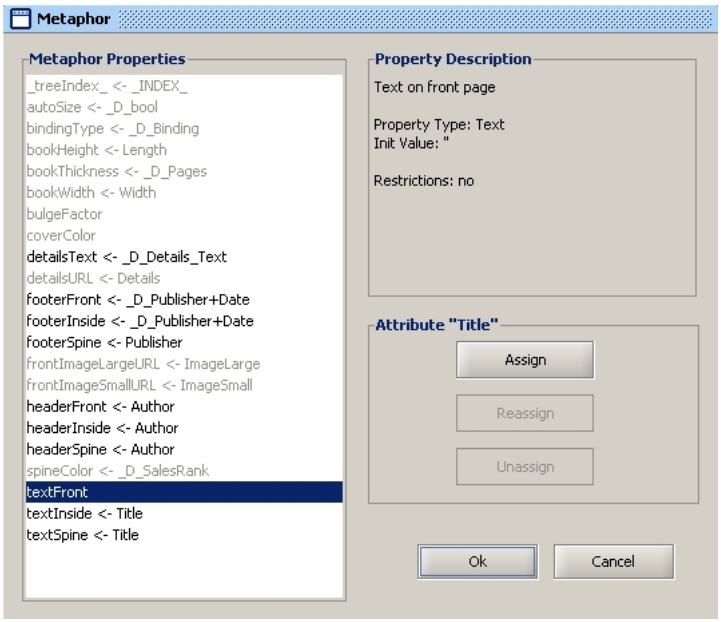


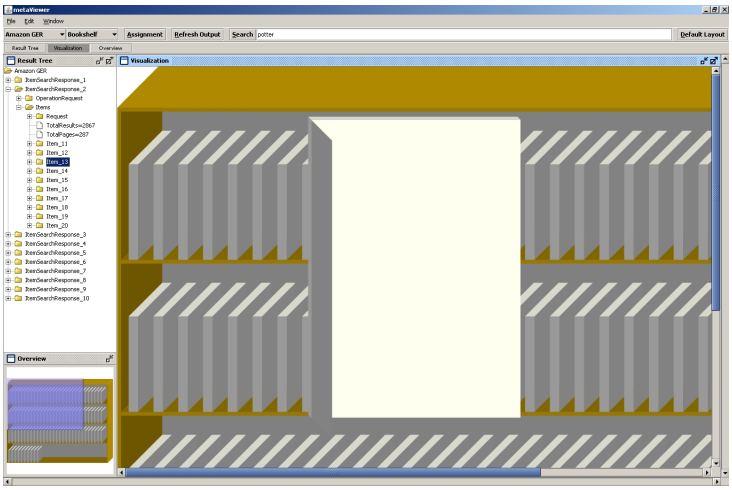
Abbildung 4.6.3: Zuordnung zu Metaphern

4.6.4 Zuordnung zu Metapher-Merkmalen

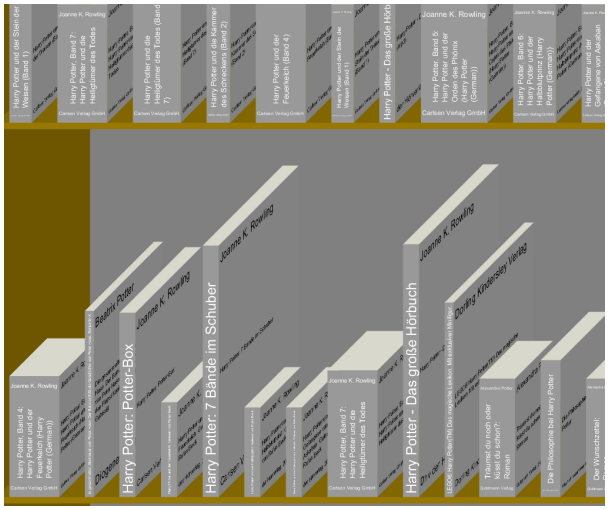
Entsprechen die Datenfelder eines Attributs der gewünschten Form, kann dieses im Assignment-Fenster ausgewählt und die Schaltfläche „*Assign to Metaphor*" betätigt werden. Damit öffnet sich das Zuordnungs-Fenster (Abbildung 4.6.3) in dem das gewählte Attribut einem oder mehreren Metapher-Merkmalen zugeordnet werden kann.

Auf der linken Seite sieht man die Liste der von der aktuellen Visualisierung zur Verfügung gestellten Metapher-Merkmale. Alle Merkmale die dem Typ des ausgewählten Attributs zugewiesen werden können werden schwarz dargestellt, die restlichen sind ausgegraut. Bereits zugewiesene Merkmale werden als *Merkmal*←*Attribut* angezeigt. Hätten wir in unserem Beispiel dieses Fenster aufgerufen, ohne vorher eine Zuordnung vorzunehmen, würden wir sehen, dass trotzdem schon einem Metapher-Merkmal ein Attribut zugeordnet ist. Die oberste Spalte würde auch in diesem Fall die Zuordnung *_treeIndex_*←*_INDEX_* anzeigen. Das in jeder Quelle vorhandene Attribut *_INDEX_*, wie in4.6.3 erwähnte, wird also dem Merkmal *_treeIndex_* zugeordnet. Dieses Metapher-Merkmal wird von jeder Visualisierung angeboten und auch direkt mit *_INDEX_* verknüpft. Damit wird die Verbindung des *Visualization*-Fensters zum *Result-Tree* hergestellt, die es ermöglicht, dass bei anklicken einer Metapher, in unserem Fall eines Buches, der entsprechende Eintrag im Ergebnisbaum aufgeklappt wird (siehe 4.6.2).

Will man nun weiteren Merkmalen ein Attribut (in diesem Fall „*Titel*") zuweisen,

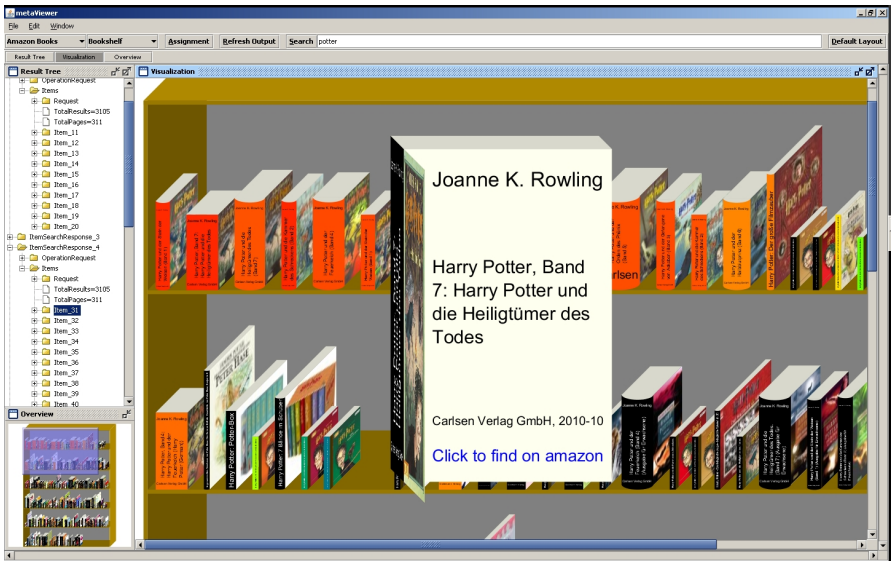


(a) Der metaViewer ohne vorgenommene Zuordnung



(b) Bücher nach Zuordnung der Merkmale für Abmessungen sowie mehrerer Textmerkmale

(c) Bindungs-Typ (Taschenbuch, gebundene Ausgabe oder CD) und Abbildungen der Umschläge wurden zugewiesen



(d) Ergebnis nach der Zuweisung einer Buchrücken-Farbe die den Verkaufsrang wiedergibt

Abbildung 4.6.4: Verwendung der Buch-Metapher

wählt man eines der schwarz geschriebenen Merkmale aus der Liste aus. Im Textfeld „*Metaphor Description*" wird eine Beschreibung zu dem markierten Merkmal angezeigt.[8] Aus dieser lässt sich ablesen, welche speziellen Eigenschaften dieses Merkmal aufweist (mehr dazu in 4.4.3). Der Bereich darunter enthält im Titel den Namen des Attributes das wir gerade zuweisen wollen und Schaltflächen um Zuweisungen durchzuführen bzw. zu löschen. Mit *Reassign* kann einem Merkmal dem bereits ein anderes Attribut zugewiesen wurde, das aktuelle Attribut zugewiesen und die bisherige Zuweisung dadurch ersetzt werden.

Um Zuweisungen eines anderen Attributes durchzuführen, muss das *Metaphors*-Fenster geschlossen (mit *Ok* werden die durchgeführten Änderungen übernommen, mit *Cancel* alle verworfen), im Assignment-Fenster das gewünschte Attribut ausgewählt und mit „*Assign to Metaphor*" das *Metaphors*-Fenster erneut aufgerufen werden.

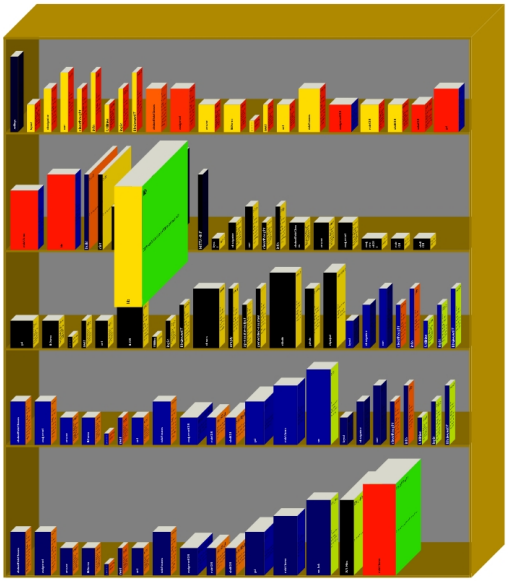
Die Bedienungsschritte der letzten Abschnitte, also jeweils erstellen eines Attributes mit dem gewünschten Inhalt und Zuweisung zu einem oder mehreren Metapher-Merkmalen, werden mehrfach durchgeführt, bis man das gewünschte Ergebnis erhält. In Abbildung 4.6.4 sieht man, wie schrittweise immer mehr Metadaten zur Steuerung der Darstellung verwendet werden:

1. In Abbildung 4.6.4b werden Textfelder wie der Titel, Autor oder Verlag auf dem Buch dargestellt, manche davon an mehreren Stellen d.h. verknüpft mit mehreren Merkmalen (z.B. am Buchrücken und auf der Vorderseite). Außerdem werden Informationen aus den Metadaten über die Größe der gefundenen Artikel für die Festlegung der Maße der dargestellten Bücher verwendet.

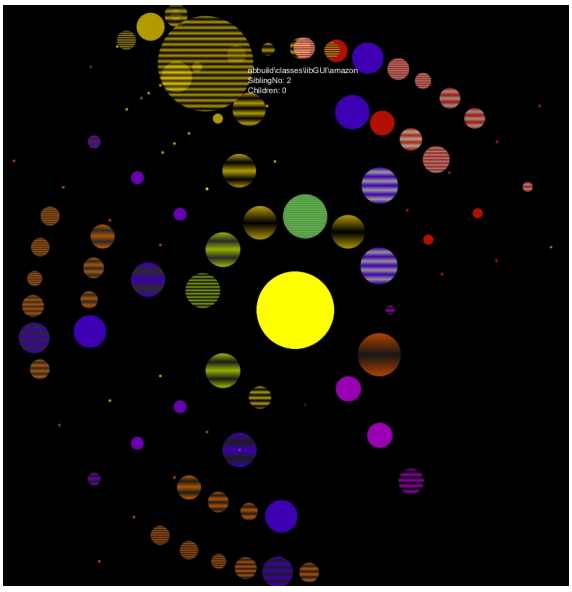
2. Abbildung 4.6.4c zeigt wie zusätzlich Abbildungen der Produkte (in den Metadaten als Links vorhanden) auf der Vorderseite der Bücher abgebildet werden. Zusätzlich werden Informationen über die Art des Artikels (gebundenes Buch, Taschenbuch, CD, Hörbuch, ...) verwendet, um die Form der Bücher zu verändern (z.B. runder Buchrücken, CD-Hülle, ...).

3. In Abbildung 4.6.4d sieht man das fertige Ergebnis einer möglichen Zuweisung. Hier wird zusätzlich ein Color-Attribut mit Farbverlauf (siehe 4.3.2) verwendet, um den Verkaufsrang eines Artikels als Farbe des Buchrückens sichtbar zu machen. Je weiter sich die Farbe von schwarz über blau, grün und gelb der Farbe Rot nähert, desto niedriger ist der Verkaufsrang des Artikels also desto häufiger wurde es in letzter Zeit gekauft.
   Außerdem wurde der Link zur Detailseite des Artikels mit dem entsprechenden

[8] Die Beschreibung wird auch angezeigt, wenn grau geschriebene Merkmale ausgewählt werden, also solche denen das gewählte Attribut nicht zugewiesen werden kann.



(a) Verzeichnisstruktur als Bücherregal



(b) Verzeichnisstruktur als Planetensystem

Abbildung 4.6.5: Darstellungen von Dateiverzeichnisdaten

Merkmal verknüpft, sodass sich diese Seite öffnet, wenn mit der Maus auf das geöffnete Buch geklickt wird.

Abbildung 4.6.5a zeigt eine weitere Darstellung mit Hilfe der Buch-Metapher. Hier werden an Stelle der Abfrage bei Amazon Informationen aus einer Verzeichnisstruktur verwendet. Zum Vergleich werden dieselben Daten in Abbildung 4.6.5b in der Form eines Planetensystem dargestellt. Einige der Metadaten-Informationen die dabei zur Darstellung kommen:

- Je höher sich das Verzeichnis in der Verzeichnisstruktur befindet, desto größer ist das zugehörige Buch bzw. desto weiter ist der entsprechende Planet von der Sonne entfernt.
- Farben geben das Alter der Verzeichnisse bzw. die Dauer seit der letzten Veränderung wieder
- Die Anzahl der enthaltenen Dateien wird als Dicke des Buches bzw. Anzahl der Streifen im Planet dargestellt
- Die Gesamtgröße eines Verzeichnisses lässt sich an der Größe der Standfläche eines Buches bzw. der Planetengröße ablesen

Bei der Planetenmetapher wurde eine *Parent-Child* Beziehung eingebaut um Unterverzeichnisse in der Nähe ihres darüber liegenden Verzeichnisses zu platzieren.

Dies wird vom metaViewer zwar nicht direkt unterstützt, konnte mit den vorhandenen Schnittstellen aber leicht zusätzlich eingebaut werden. Eine Erweiterung der Möglichkeiten in dieser Richtung könnte speziell für hierarchisch strukturierte Daten und entsprechend angeordnete Darstellungen nützlich sein und sollte in weiteren Versionen des metaViewers angedacht werden.

4.7 Zusammenfassung

Wir haben in diesem Kapitel gesehen, welche Herausforderungen bei der Implementierung des metaViewers zu meistern waren und welche Vorgehensweisen dabei zur Anwendung kamen. An Hand des Datenflusses durch die Applikation wurde gezeigt, wie Suchergebnisse eingelesen, neue Attribute definiert, Metapher-Merkmale zugeordnet und die Ergebnisse graphisch dargestellt werden. Es wurden auch die Schnittstellen erwähnt, die definiert wurden, um die Erweiterung des metaViewers um neue Datenquellen und Darstellungsformen zu unterstützen. Wie diese Schnittstellen aussehen und wie sie zu verwenden sind, wird im folgenden Kapitel näher erläutert. Am Schluss wurde noch ein Überblick gegeben, welche Bedienelemente die Oberfläche des metaViewers bietet und wie diese genutzt werden können, um aus den Quelldaten die gewünschte Darstellung zu erhalten.

Im nächsten Kapitel wird nun näher auf die Programmstruktur des metaViewers eingegangen und die verwendeten Programmierumgebungen und Programmpakete werden kurz vorgestellt. Speziell die interne Datenrepräsentation und die angebotenen Erweiterungs-Schnittstellen werden detailliert aufgezeigt.

5 Systemarchitektur

Im ersten Abschnitt dieses Kapitels werden wir kurz auf die Entwicklungsumgebung und die wichtigsten Programmpakete eingehen, die für die Entwicklung des metaViewers verwendet wurden. Danach werfen wir einen genaueren Blick auf den Aufbau des metaViewers. Dabei werden die wichtigsten Programmpakete vorgestellt, aus denen der metaViewer besteht und Klassendiagramme geben einen Überblick über den strukturellen Aufbau zentraler Komponenten. Anschließend werden wir speziell auf zwei besondere Aspekte des metaViewers eingehen, zum einen die interne Datenrepräsentation und zum anderen die Programmierschnittstellen die ein zukünftiges Erweitern des metaViewers ermöglichen sollen.

5.1 Programmierumgebung und verwendete Programmpakete

Der metaViewer wurde in *Java 1.6* (http://www.java.com/) mit Hilfe von *Eclipse* (http://www.eclipse.org/) in den Versionen *3.3* und *3.6* unter *Windows 2000/XP/7* (http://windows.microsoft.com/de-AT/windows/home) entwickelt. Als GUI-Builder wurde *Jigloo 4.6* (http://www.cloudgarden.com/jigloo/) verwendet. Getestet wurde er zusätzlich unter *Linux Ubuntu 10.10* (http://www.ubuntu-austria.at/). Der metaViewer wurde als eigenständige Applikation (nicht als Applet) entwickelt und steht in dieser Version deshalb nicht als Webapplikation zur Verfügung. Um *metaViewer.jar* zu verwenden (siehe 4.6.1), wird eine installierte Java Laufzeitumgebung (http://www.java.com/) von Sun/Oracle ab der Version 6 benötigt.

Es folgt eine kurze Beschreibung der wichtigsten Programmpakete die zur Realisierung des metaViewers verwendet wurden:

HSQLDB 1.8 HSQLDB[1] ist ein Softwarepaket geschrieben in Java zur Realisierung einer relationalen Datenbank. Der metaViewer verwendet eine damit intern erstellte Datenbank zur effizienten Speicherung von Attribut-Daten. Außerdem wird die Möglichkeit von SQL genutzt, aus vorhandenen Spalten neue zu erzeugen und dabei Funktionen auf die Inhalte anzuwenden. Auf diese Art wird ein Großteil der Operationen und Typ-Umwandlungen realisiert, die der metaViewer zur Anpassung der Attribute zur Verfügung stellt.

[1] http://www.hsqldb.org/

Piccolo2D.Java 1.3.1 Piccolo2D[2] ist ein Toolkit zur Erzeugung strukturierter, zweidimensionaler Grafik und unterstützt Möglichkeiten zum Zoomen, Zusammenfassen mehrerer 2D-Objekte und zur Auswahl solcher Objekte sowie zur Erstellung von Animationen. Die äußerst effiziente Erzeugung der graphischen Ausgabe ermöglicht die Darstellung und Animation einer sehr großen Anzahl von Objekten. Die im metaViewer umgesetzten Visualisierungen nutzen diese Möglichkeiten und sind deshalb in der Lage, auch eine sehr lange Liste an Suchergebnissen als graphische Metaphern darzustellen. Das Überblicksfenster (Birdseye View) wurde ebenfalls mit Hilfe von Piccolo2D realisiert.

5.2 Programmpakete

Hier soll ein Überblick über den groben Aufbau des metaViewers gegeben werden. Dabei werden wir nicht auf Details der Implementierung eingehen, sondern nur die einzelnen Komponenten vorstellen, aus denen die Software aufgebaut ist. Im Folgenden werden die Pakete vorgestellt, in die der Sourcecode aufgeteilt wurde. Zu jedem Pakte werden die wichtigsten darin enthaltenen Klassen vorgestellt.

mainGUI

Enthält vor allem die Klasse MainFrame die für die Darstellung des Hauptfensters der Applikation verantwortlich ist (Diagramm in Abbildung 5.3.1a).

MainFrame zentrale Ansicht des metaViewers, übernimmt die Verwaltung der Fenster und implementiert die Menüs sowie die Werkzeugleiste

BirdsEyeView verwaltet das Fenster mit der Übersichtsdarstellung und sorgt für dessen Aktualisierung

assignmentGUI

Hier werden die Klassen zusammengefasst, mit denen die Eingabemasken zur Bearbeitung von Attributen und Metapher-Merkmalen implementiert werden (Diagramm in Abbildung 5.3.1b). Dies umfasst vor allem die in 4.6.3 und 4.6.4 beschriebenen Bedienungsmöglichkeiten.

Assignment implementiert die zentrale Ansicht für die Verwaltung von Attributen (Abbildung 4.6.1)

VirtualAttributeGUI Fenster für die Erstellung und Bearbeitung abgeleiteter Attribute (Abbildung 4.6.2)

[2] http://www.piccolo2d.org/index.html

Edit*[Type]*Attribute Für jeden Attribut-Typ steht eine eigene Klasse zur Verfügung, dabei steht *[Type]* für einen der Typen *Boolean*, *Color*, *Float*, *Int*, *Link*, *List* oder *Text*. Implementiert die, für den jeweiligen Attribut-Typ zur Verfügung stehenden Funktionen, die direkt über die Eingabemaske angewendet werden können (Abbildung 4.3.1).

MetaphorGUI Eingabemaske zur Zuordnung von Attributen zu Metapher-Merkmalen (Abbildung 4.6.3)

assignment

Jeder Attribut-Typ und jeder Merkmals-Typ (hier Property genannt) wird durch eine Klasse repräsentiert, die alle spezifischen Informationen und Funktionalitäten enthält, die für seine Verwendung notwendig sind (Diagramm in Abbildung 5.3.1c).

Im Folgenden steht *[Type]* für einen der möglichen Attributs- bzw. Merkmals-Typen: *Boolean*, *Color*, *Float*, *Int*, *Link*, *List* oder *Text*.

*[Type]*Attribute Implementierungen der Attribut-Typen; enthalten Funktionen zur Erzeugung Typ-spezifischer Anweisungen für die Datenbankabfragen (siehe z.B. auch 5.4)

*[Type]*Property Implementierungen der Metapher-Merkmals-Typen; enthalten vor allem die Umsetzung der jeweils anwendbaren Restriktionen (siehe 4.4.3)

abstractDataClasses

Dieses Paket enthält abstrakte Klassen von denen neue Klassen abgeleitet werden müssen, die zur Erweiterung des metaViewers dienen. Sie sind in diesem Paket zusammengefasst, weil in vielen der Pakete auf ihre Definition zurückgegriffen wird. Deshalb kommen auch in allen Klassen-Diagrammen in Abbildung 5.3.1 manche von ihnen vor.

AbstractAttribute Basisklasse für jeden Attribut-Typ (siehe *[Type]*Attribute in Paket *assignment*)

AbstractEditAttribute Basisklasse der Eingabemasken mit denen Attribute bearbeitet werden können (siehe Edit*[Type]*Attribute in Paket *assignmentGUI*)

AbstractMetaphor Klasse von der alle implementierten Metaphern abgeleitet werden (z.B. *Book* oder *Planet*)

AbstractProperty Basisklasse aller Metapher-Merkmale (siehe *[Type]*Property in Paket *assignment*)

AbstractQueryResult Diese Klasse verwaltet die von einer Datenquelle eingelesenen Suchergebnisse und implementiert die Schnittstelle zur intern verwendeten Datenbank, um die Ergebnisse dort zu speichern. Für jede Datenquelle wird von dieser Klasse abgeleitet (siehe */Source/*QueryResult in Paket dataSources). Dabei müssen die Methoden zur Abfrage der Quelle überschrieben werden.

AbstractRequest Basisklasse für Datenquellen die eine Online-Abfrage benötigen (z.B. *Amazon*); die notwendigen Schritte zur Bildung der entsprechenden URL werden hier durchgeführt

AbstractVisualization Diese Klasse bildet die Basis für die Darstellung der jeweiligen Visualisierung (z.B. *BookShelf* oder *SolarSystem*) im Ausgabefenster. Sie kümmert sich um die Synchronisation mit dem Übersichtsfenster (siehe *BirdsEyeView* in Paket *mainGUI*) und stellt Methoden zur Verfügung, mit denen Metaphern der Darstellung hinzugefügt werden können. Für jede implementierte Visualisierung werden hier die Metapher-Merkmale definiert. Außerdem müssen die gewünschten Interaktionsmöglichkeiten (z.B. Bücher aus dem Regal nehmen usw.) hier implementiert werden (siehe Diagramm 5.3.1a).

dataSources

In diesem Paket sind Klassen enthalten, mit denen die Abfrage von Ergebnissen aus den zur Verfügung stehenden Quellen ermöglicht wird (siehe Abbildung 5.3.1a).

*/Source/*QueryResult Für jede Datenquelle (aktuell vorhandene Ausprägungen für */Source/* sind *Amazon* oder *File*) werden hier die notwendigen Schritte umgesetzt, um Daten abzufragen und in der internen Datenbank zu speichern.

*/Source/*Request Erstellung der URL für die Datenabfrage (aktuell nur für *Amazon* benötigt)

Weiters werden in diesem Paket Klassen zur Verfügung gestellt, die eine Umwandlung und Speicherung strukturierter Daten in XML-Form unterstützen.

GUI's für Output

Für jede Visualisierung werden mehrere Klassen benötigt. Die Klassen zu einer Visualisierung werden jeweils in einem eigenen Paket zusammengefasst. Aktuell gibt es die Pakete *shelfGUI* und *solarGUI* (siehe Diagramm 5.3.1a). Zumindest zwei Klassen müssen in jedem solchen Paket enthalten sein: eine Klasse die ein Element der

gewählten Metapher repräsentiert (z.B. *Book* oder *Planet*) und eine weitere Klasse die für die Darstellung einzelner Elemente in der zugehörigen Umgebung verantwortlich ist (z.B. *BookShelf* oder *SolarSystem*). Diese Klassen werden jeweils von *AbstractMetaphor* bzw. *AbstractVisualization* abgeleitet (siehe Paket *abstractDataClasses*).

5.3 Klassendiagramme

Die Klassendiagramme in Abbildung 5.3.1 zeigen die Zusammenhänge zwischen den Kernkomponenten des metaViewers. Auch hier wurde auf die Angabe von Details wie Methoden oder Attribute verzichtet, nur die Beziehungen der Klassen zueinander sollen aufgezeigt werden.

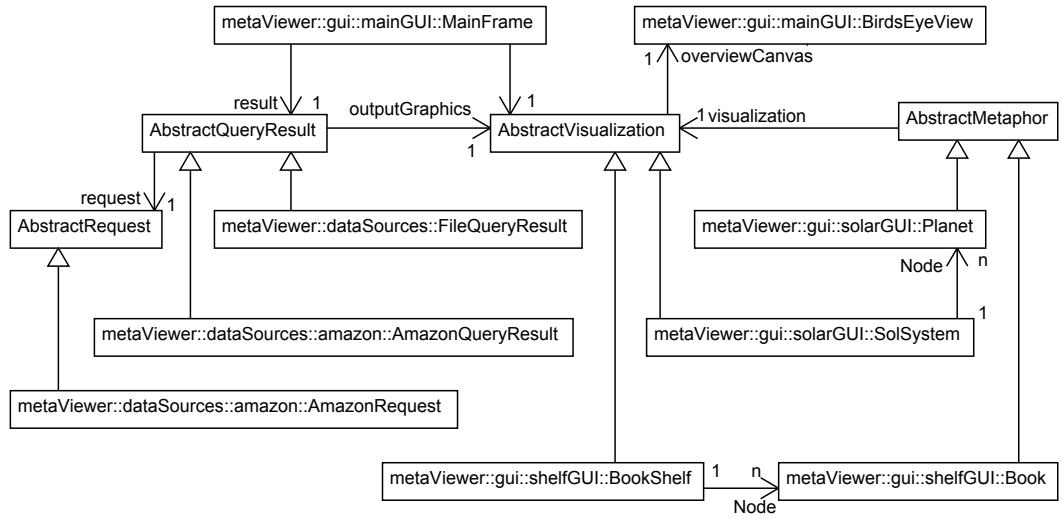
Abbildung 5.3.1a zeigt das Hauptfensters *MainFrame* als zentrales Element. Es besitzt eine Referenz auf die aktuell ausgewählte Datenquelle, dargestellt im linken Teil mit der Klasse *AbstractQueryResult* als Basis. Der rechte Teil zeigt die Verbindung zur gewählten graphischen Darstellung. Ausgehend von *AbstractVisualization* erkennt man die Verbindung zum Übersichtsfenster *BirdsEyeView* und die abgeleiteten Klassen der implementierten Visualisierungen.

Bild 5.3.1b zeigt die Struktur der Bedienelemente mit denen die Zuordnung der Quell-Datenfelder zu den Metapher-Merkmalen definiert werden kann. Ausgehend von der zentralen Klasse *Assignment* können über *VirtualAttributeGUI* neue Attribute erstellt und bearbeitet werden. Teile dieses Eingabefenster ändern sich, je nach Attributs-Typ der gerade bearbeitet wird (siehe auch Abbildungen 4.6.2 und 4.3.1). Die Klasse *MetaphorGUI* ermöglicht anschließend die Zuordnung eines Attributes an ein oder mehrere Metapher-Merkmale.

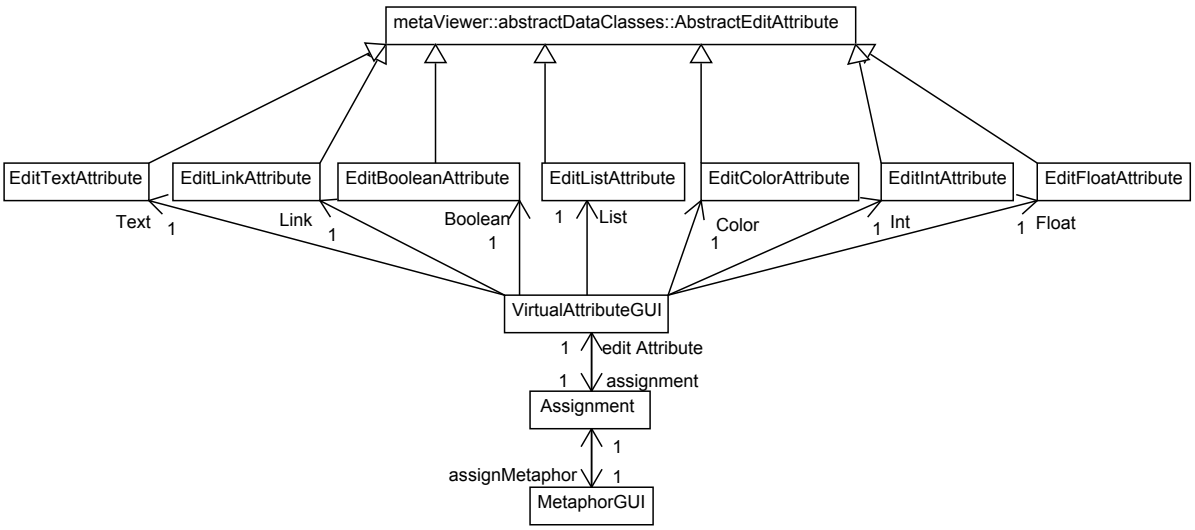
Wie Attribute im metaViewer intern repräsentiert sind zeigt Abbildung 5.3.1c im oberen Bereich. Ausgehend von *AbstractAttribute* werden alle Attributs-Typen abgeleitet, auch *VirtualAttribute*, das die Basis aller abgeleiteten Attribute bildet und von dem *VirtualAttributeGUI* jeweils eine Instanz verwaltet. Der untere Teil dieses Diagramms zeigt die Basisklasse aller Metapher-Merkmale *AbstractProperty* von der für jeden Merkmals-Typ eine eigene Klasse abgeleitet wird.

5.4 Interne Datenrepräsentation - Die View „V_Attributes_*[Source]*"

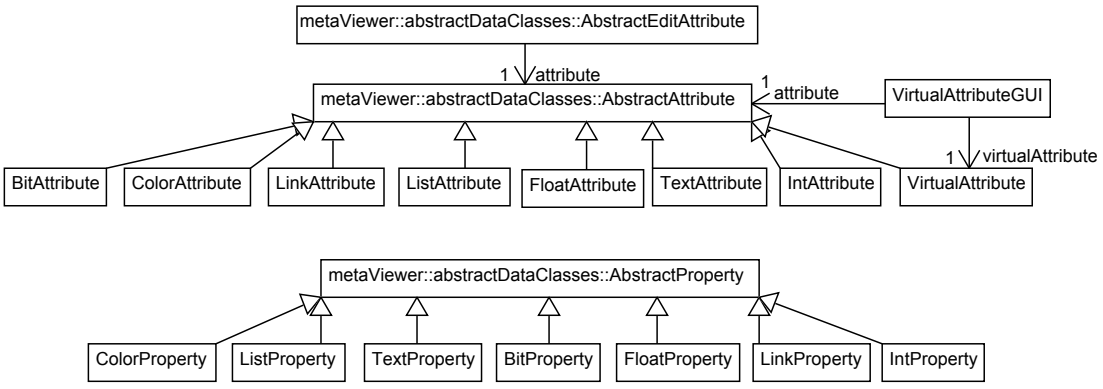
Im Abschnitt 4.3.2 wurde aufgezeigt, welche Vorteile die Verwendung einer Datenbank zur internen Datenrepräsentation bieten kann. Sie kommt im metaViewer zur Speicherung und Verarbeitung der Attribute (mittlerer Bereich im Datenflussdiagramm in Abbildung 4.1.1) zum Einsatz.



(a) Die zentralen Klassen im Paket mainGUI und ihre Verbindungen zu den Klassen für Datenquellen und Visualisierung



(b) GUI-Klassen für die Bearbeitung und Zuordnung von Attributen



(c) Klassen die Attribute und Merkmale repräsentieren

Abbildung 5.3.1: Die wichtigsten Klassen des metaViewers

Die Datenbank wird bei jedem Aufruf des metaViewers im Speicher aufgebaut. Sie enthält für jede verwendete Datenquelle jeweils eine Tabelle mit dem Namen *Attributes_[Source]* und eine View *V_Attributes_[Source]*, wobei *[Source]* für die Bezeichnung der verwendeten Quelle steht. In der Tabelle werden bei jeder Abfrage alle Daten der Quell-Attribute gespeichert. Die View wird nach jeder Änderung an den abgeleiteten Attributen neu erstellt. Sie enthält alle Spalten der zugehörigen Tabelle, also alle Quell-Attribute. Zusätzlich enthalten ihre Spalten die komplette Liste abgeleiteter Attribute inklusive aller anzuwendenden Funktionen. Die Abfrage aller Attribute kann somit in einheitlicher Weise über diese View erfolgen, was z.B. bei jedem Aufbau der graphischen Ausgabe zur Anwendung kommt.

Die Verwaltung der Datenbank also ihre Erzeugung, die Erstellung der Tabellen und Views und den Aufbau der Verbindung zu ihr übernimmt die Klasse *AbstractQueryResult.* Zusätzlich stellt sie Methoden zum Befüllen der Tabellen und zur Abfrage zur Verfügung. Die für jedes Attribut spezifischen Teile der Abfrage werden in den von *AbstractAttribute* abgeleiteten Klassen bzw. deren Instanzen verwaltet (siehe Diagramm in Abbildung 5.3.1c).

5.5 Programmschnittstellen - Erweiterung des metaViewers

Eine der Kernkomponenten des metaViewers sind die Möglichkeiten die zur Definition von Attributen und deren Zuweisung zu Metapher-Merkmalen zur Verfügung stehen. Dem Anwender sollte die Nutzung dieser Fähigkeiten so einfach wie möglich gemacht werden und so wurden die entsprechenden Bedienelemente aus den Abschnitten 4.6.3 und 4.6.4 entwickelt. Dabei wurde mehr Augenmerk auf die erstellten Konzepte gelegt, als auf die Vollständigkeit der implementierten Funktionen. Viele Ideen für weitere Funktionalitäten tauchten auch erst im Laufe der Entwicklung auf, und konnten aus zeitlichen Gründen in dieser Version nur teilweise oder gar nicht umgesetzt werden. Um das Potenzial des metaViewers dadurch nicht einzuschränken, wurde speziell auf die Offenheit des Systems und die Möglichkeit zu Erweiterungen geachtet. Die wichtigsten Schnittstellen an denen zusätzliche Komponenten angefügt werden können (einige sind in Abbildung 4.1.1 erkennbar), sollen in diesem Abschnitt beschrieben werden. Die folgenden Beschreibungen sind vorrangig dazu gedacht, die Stellen im Programmcode zu identifizieren, an denen Änderungen bei der Erweiterung des metaViewers durchgeführt werden müssen. Ein bloßes Durchlesen, ohne dabei den Sourcecode näher zu betrachten, erscheint für diesen Unterabschnitt jedoch weniger sinnvoll.

5.5.1 Datenquellen

Die zentrale Klasse zur Verwaltung einer Datenquelle muss von *AbstractQueryResult* abgeleitet werden (siehe 5.2 und Abbildung 5.3.1a). Diese übernimmt bereits Aufgaben wie die Verwaltung der Datenbankelemente und bietet Unterstützung z.B. zur Erstellung des Baumes für die Darstellung im Fenster *Result Tree* (*4.6.2*) oder das Einlesen und Speichern der Daten als XML-Dokument. Die zwei Methoden *showResult(String searchString)* und *refreshOutput()* müssen überschrieben werden. Mit *showResult* soll eine neue Abfrage mit dem angegebenen Suchtext an die Quelle übermittelt werden. Wenn man die vorhandenen Implementierungen *AmazonQueryResult* und *FileQueryResult* als Vorbild verwendet sieht man, dass die Abfrage in einem eigenen Thread durchgeführt wird und der Fortschritt in einem Fenster angezeigt werden kann. Dieses Fenster wird ebenfalls von der Basisklasse zur Verfügung gestellt.

Wird für die Abfrage die Erstellung einer URL benötigt, in der auch notwendige Parameter codiert sind, kann dazu die Klasse *AbstractRequest* verwendet werden, um diese Aufgabe, getrennt von der eigentlichen Abfrage der Quelle, zu übernehmen. Dies wurde z.B. mit *AmazonRequest* so durchgeführt.

Notwendige Einstellungen und Parameter können in der Klasse *SourceSettings* hinzugefügt werden und nach Erweiterung der Methode *getResult()* in *mainGUI*, kann die neue Quelle bereits verwendet werden.

5.5.2 Attribut-Typen

Die Erstellung eines neuen Attribut-Typs ist etwas aufwändiger als andere Erweiterungen, da die definierten Typen den innersten Kern des metaViewers bilden. Soll ein neuer Typ geschaffen werden, orientiert man sich am besten am Code der bereits vorhandenen. Hier soll nur grob aufgelistet werden, welche Schritte dabei notwendig sind und welche Basisklassen dafür zur Verfügung stehen.

- eintragen der neuen Typ-Bezeichnung im Aufzählungstyp *typeCategory* in der Klasse *Settings*
- Erweiterung der statischen Tabellen in *AbstractAttribute* in denen der Spaltentyp in der Datenbank festgelegt und mit dem Attribut-Typ verknüpft wird
- erstellen der neuen Attribut-Klasse durch ableiten von *AbstractAttribute*
  - überschreiben der internen Klasse *AttributeOptions* in der die speziellen Parameter zur Manipulation dieses Attributes gespeichert werden (siehe auch nächsten Unterabschnitt)

  - überschreiben der Methode *getStatement(String s)* in der, abhängig von den Informationen aus *AttributeOptions*, das entsprechende SQL-Statement gebildet wird

- erstellen des Fensters mit den für diesen Typ spezifischen Bedienelementen indem von *AbstractEditAttribute* abgeleitet und die gewünschten Bedienelemente erstellt werden; die daraus erhaltenen Parameter müssen in den Methoden *loadAttributeOptions()* und *writeAttributeOptions()* gespeichert bzw. geladen werden (siehe auch nächsten Unterabschnitt)

- erweitern der Klasse *VirtualAttributeGUI* so dass der neue Typ für Ziel-Attribute zur Verfügung steht und das richtige Fenster mit den eben erstellten Bedienelementen zurückgeliefert wird

5.5.3 Funktionen zur Manipulation von Attributen

Sollen die Funktionen erweitert werden, die bei der Bearbeitung eines Attributes direkt über Bedienelemente angewendet werden können, ohne das SQL-Statement manipulieren zu müssen, kann dies durch Bearbeitung der Klasse *Edit*[Type]*Attribute* erfolgen. Dabei steht [Type] für den zu Grunde liegenden Attribut-Typ.

Bisherige graphische Bedienelemente wurden mit Hilfe von Jigloo[3] erstellt, es empfiehlt sich also, ebenfalls dieses Tool zur Bearbeitung bzw. Erweiterung dieses Fensters zu verwenden. Ändern sich durch die Anpassung Art oder Anzahl der Parameter mit denen die Manipulationen beschrieben werden, müssen die Klasse *AttributeOptions* in der Basisklasse des entsprechenden Attribut-Typs, sowie die Methoden *loadAttributeOptions()* und *writeAttributeOptions()* angepasst werden (siehe auch vorigen Unterabschnitt). Zusätzlich muss evtl. die Methode *getStatement()* in der Attribut-Klasse so verändert werden, dass sie aus den geänderten Parametern das korrekte SQL-Statement erzeugen kann.

5.5.4 Metaphern

Die Darstellung einer Metapher bzw. der zugehörigen Visualisierung wird im Wesentlichen durch zwei Klassen definiert. Die erste Klasse wird von *AbstractMetaphor* abgeleitet und bestimmt das Erscheinungsbild eines Elementes (also z.B. eines Buches). Da AbstractMetaphor von *PNode* (die Basis eines graphischen Elementes in Piccolo[4]) abstammt, können zur Darstellung der Metapher alle von Piccolo zur Verfügung gestellten Funktionen verwendet werden. Insbesondere auch zur Kapselung mehrerer zweidimensionaler Objekte und zur Definition von Transformationen zur

[3] http://www.cloudgarden.com/jigloo/
[4] http://www.piccolo2d.org/index.html

Erzeugung von Bewegungsabläufen. Wie das im Detail aussieht, lässt sich am besten durch studieren vorhandener Umsetzungen (Klassen *Book* und *Planet*) erlernen. Die Vorgaben für die Erstellung dieser Klasse sind relativ gering, allerdings müssen die Methoden an die Anforderungen der zweiten Klasse angepasst werden, deren Beschreibung nun folgt.

Die zweite Klasse ist für die Ausgabe der Umgebung zuständig, in der die Metaphern dargestellt sind, also z.B. ein Bücherregal oder ein Sonnensystem. Sie wird von der Klasse *AbstractVisualization* abgeleitet und kümmert sich auch um die Verwaltung der Interaktionsmöglichkeiten mit den Elementen im Ausgabefenster. Folgende Methoden müssen dabei überschrieben und implementiert werden:

createProperties() hier werden alle Merkmale der Metapher definiert und zu jedem Merkmal alle anzuwendenden Restriktionen angeführt

resetOutput() entfernt alle Metapher-Elemente und setzt die Ausgabe zurück

getNewMetaphor() gibt ein neu erzeugtes Metapher-Element zurück

getMainCanvas() erzeugt eine neue Zeichenfläche und implementiert dabei alle Interaktionsmöglichkeiten, insbesondere die Definition von Maus Events

finishGraphics() löscht die Ausgabe und stellt alle enthaltenen Metapher-Elemente neu dar

5.5.5 Merkmals-Typen

Sollten die vorhandenen Merkmals-Typen nicht ausreichen, bzw. durch die Erstellung eines neuen Attribut-Typs ein passender Merkmals-Typ benötigt werden, kann ein neuer Typ definiert werden. Durch Ableitung von der Klasse *AbstractProperty* wird sichergestellt, dass der neue Typ alle wichtigen Eigenschaften besitzt. Zusätzlich muss die Methode *createProperty(...)* in *AbstractVisualization* angepasst werden, sodass sie Merkmale des neuen Typs erzeugen kann.

In der neu erstellten Merkmals-Klasse muss die Methode *setValue(AbstractMetaphor metaphor, Object value)* überschrieben werden. Diese muss den Wert eines Merkmals setzen können und dabei auf die Einhaltung der geltenden Restriktionen achten. Alle weiteren Methoden die bei der Erstellung eines neuen Merkmals implementiert werden müssen, betreffen die für diesen Typ auswählbaren Restriktionen und werden im folgenden Unterabschnitt beschrieben.

5.5.6 Restriktionen

Mit Restriktionen lassen sich zusätzliche Beschränkungen von Merkmalen durchsetzen, die bei der Zuweisung eines Wertes gelten. Soll für einen Merkmals-Typ

eine neue Restriktion auswählbar sein bzw. werden für einen neuen Merkmals-Typ Restriktionen benötigt, können diese in der Klasse [Type]*Property* implementiert werden, wobei [Type] hier für den Typ des Merkmals steht.

Die Art einer Restriktion definiert sich stets über die Anzahl und den Datentyp der anzugebenden Parameter. Je nach Parameter-Kombination der neuen Restriktion, muss die dazu passende Methode *addRestriction(restriction r [,Parameterliste])* angepasst bzw. erzeugt werden. Für die Umsetzung der neuen Restriktion muss in der Methode *setValue(AbstractMetaphor metaphor, Object value)* gesorgt werden. Zur Definition der Beschreibung, die als Teil der Merkmals-Beschreibung ausgegeben wird (siehe Abbildung 4.4.1), muss die Methode *getRestrictionDescription()* überschrieben werden.

Wird eine Restriktion mit bisher nicht verwendeter Parameter-Kombination benötigt, muss ein passender Aufzählungstyp definiert und der Variable *allRestrictions* hinzugefügt sowie eine dazugehörige Liste erstellt werden. Zusätzlich muss eine entsprechende Methoden-Definition in der Klasse *AbstractProperty* erstellt werden.

5.6 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Aspekte im Aufbau des metaViewers genauer betrachtet. Entwicklungsumgebung und Programmbibliotheken die bei der Implementierung zur Anwendung kamen wurden erwähnt. Anschließend folgte eine Beschreibung der einzelnen Sourcecode Pakete des metaViewers.

Drei Klassendiagramme zeigen die Struktur der wichtigsten Programmteile und ein eigener Abschnitt behandelt die Datenbank die zur internen Speicherung und Verarbeitung der Ergebnisdaten verwendet wird. Eine Beschreibung der vom metaViewer bereitgestellten Programmschnittstellen, mit denen sich neue Funktionalitäten implementieren lassen, beschließt dieses Kapitel.

6 Schlussbetrachtungen

Diese Arbeit entstand aus dem Wunsch heraus, die Ideen die zur Entwicklung des libViewer führten, weiter zu verfolgen und ihre Umsetzung auf einen aktuelleren technischen Stand zu bringen. Aus diesen, anfangs eher unspezifischen Anforderungen, entstanden Zug um Zug immer konkretere Vorstellungen davon, welche Möglichkeiten der metaViewer haben soll. In diesem Kapitel wird noch einmal zusammengefasst, welche Ideen im metaViewer umgesetzt wurden. Dabei werden auch Erkenntnisse angeführt, die wir während dieser Arbeit gewinnen konnten. Abschließend folgt ein Ausblick darauf, welche Möglichkeiten noch im metaViewer stecken, die nur darauf warten, in zukünftigen Projekten erforscht zu werden.

6.1 Ergebnisse der Arbeit

Wir haben mit dem metaViewer gezeigt, wie eine Software aussehen kann, mit deren Hilfe graphische Darstellungen von Metadaten-basierten Trefferlisten entwickelt und optimiert werden können. Anfangs war es unser Ziel, neue Aspekte der Darstellung mit Hilfe von graphischen Metaphern auszuprobieren und eine weiterentwickelte Form der Buch-Metapher auch auf andere Datenquellen als nur Online-Bibliotheken anzuwenden. Bald entwickelte sich daraus der Wunsch, ein offenes System zur Umsetzung und zum Ausprobieren graphischer Metaphern für unterschiedliche Datenquellen zu schaffen. Bald wurde klar, dass die Herausforderungen dabei ganz andere sind, als bei der Implementierung einer konkreten Darstellung. Der Prozess der Zuordnung von Metadaten-Feldern zu graphischen Metapher-Merkmalen musste abstrahiert von einer konkreten Zuweisung betrachtet werden, um jene Stellen zu identifizieren, an denen verallgemeinerte Schnittstellen eine möglichst hohe Flexibilität gewährleisten.

Wenn man den Aufwand vergleicht, der notwendig war, um die erste Version der Buch-Metapher zu implementieren, mit jenem, eine weitere Metapher zu realisieren, nun mit Hilfe der vordefinierten Basisklassen, lässt sich die Nützlichkeit des metaViewers schnell erkennen. Wurden für die Entwicklung der ersten Darstellung noch Wochen investiert, war die Umsetzung einer weiteren Metapher nur noch eine Sache von Tagen. Auch die Erweiterung um eine zusätzliche Datenquelle konnte in weniger als einer Woche durchgeführt werden. Es muss natürlich erwähnt werden, dass die Art der Quelle und die Komplexität der Darstellung dabei eine entscheidende

Rolle spielen, die Einbindung in den metaViewer erfolgte jedoch in beeindruckend einfacher Weise. Auch nach dem Entschluss, sich auf die Offenheit des Systems zu konzentrieren, konnten wir nicht damit rechnen, einen so hohen Grad an Flexibilität erreichen zu können.

Das Ergebnis dieser Arbeit ist ein Programm, mit dem bereits jetzt eine Vielzahl an Visualisierungsmöglichkeiten sehr schnell umgesetzt und auf ihre Tauglichkeit hin untersucht werden können. Bereits mit den aktuell zur Verfügung stehenden Datenquellen und Metaphern macht es Spaß, mit den Funktionen des metaViewers herumzuspielen und unterschiedliche Zuordnungen einfach auszuprobieren. Das große Potential liegt jedoch in der Möglichkeit, Erfahrungen beim Herumprobieren zu sammeln und die daraus entstehenden Ideen für Erweiterungen des metaViewers schnell umsetzen zu können. Einige solcher Ideen, die bereits während der Entwicklung auftauchten, in dieser Version jedoch noch nicht umgesetzt werden konnten, sollen abschließend kurz erwähnt werden.

6.2 Diskussion und Ausblick

Sicherlich eine der größten Herausforderungen bei der weiteren Arbeit mit dem metaViewer, liegt darin, viele unterschiedliche Metaphern und Darstellungen zu implementieren. Dabei wurden auch in den vorhandenen Visualisierungen noch lange nicht alle Möglichkeiten ausgeschöpft. So könnten hierarchisch organisierte Darstellungen von großem Nutzen sein, besonders dann, wenn auch Methoden zur Clusterung der Ergebnisse zum Einsatz kommen. Auch wurden bisher noch kaum Methoden zur flexiblen Platzierung von Elementen eingesetzt. Sortierung nach unterschiedlichsten Kriterien oder z.B. im Falle der Buch-Metapher, das Verschieben der Bücher nach vorne oder hinten wären z.B. Erweiterungen die sehr leicht umzusetzen sind.

Die Erschließung weiterer Datenquellen könnte die Einführung zusätzlicher Attribut-Typen sinnvoll machen. Auch die Erweiterung der über Eingabemasken einstellbaren Funktionen, die auf Attribute angewendet werden können, kann in weiterer Folge nützlich sein. Beides sollte mit den zur Verfügung gestellten Schnittstellen relativ leicht durchführbar sein. Welche Attribute und Funktionen dabei am sinnvollsten sind, muss die praktische Anwendung des metaViewers zeigen.

Wenn es um die Bedienung des metaViewers geht, ist vor allem die Nutzung einer Datenbank und die damit zur Verfügung stehende Abfragesprache SQL ein Grund für die hohe Flexibilität des Systems. Damit stehen sehr vielfältige Funktionen zur Bearbeitung der Datenfelder zur Verfügung, ohne dass diese einzeln implementiert werden müssen. Auch die Verarbeitung von Datums- und Zeitformaten ist in SQL kein Problem. Obwohl in dieser Version kein entsprechender Attribut-Typ implementiert wurde, können auf diese Weise Datumsfunktionen durchgeführt werden. Trotzdem könnte es hilfreich sein, wenn die wichtigsten Datums-Funktionen mit

Hilfe eines eigenen Typs umgesetzt werden, so dass sie auch über Eingabemasken zur Verfügung stehen. Der einzige Attribut-Typ für dessen Bearbeitung die Datenbank kaum Unterstützung liefern kann, ist das Color-Attribut. Um die Verwendung von Farben zu unterstützen, mussten alle Bearbeitungsmöglichkeiten selbst implementiert werden. So unterstützt der metaViewer z.B. bereits die Definition eines Farbverlaufes auf komfortable Weise. Weitere Methoden zur Definition und für die Zuordnung von Farben wären denkbar. So könnten z.B. Farbverläufe unterschiedlicher Merkmale aufeinander abgestimmte werden. Das Abspeichern erstellter Farbverläufe wäre nützlich oder auch die Bereitstellung einer Auswahl vordefinierter Verläufe, die sich besonders bewährt haben.

Ein weiterer Schritt in der Entwicklung des metaViewers könnte durch eine zusätzliche Klassifizierung von Metapher-Merkmalen erfolgen. Speziell für numerische Daten würden sich dadurch neue Möglichkeiten eröffnen. Bei der Verwendung der Daten die aus einer Verzeichnisstruktur ausgelesen werden, wurde klar, dass oft Kombinationen von Datenfeldern in einem wiederkehrenden Verhältnis stehen und sich daraus Muster ablesen lassen. Ein Beispiel wäre etwa die Anzahl aller in einem Verzeichnis enthaltenen Dateien sowie dessen Gesamtgröße. Aus dem Verhältnis dieser zwei Werte lässt sich die durchschnittliche Größe der enthaltenen Dateien ermitteln. Bei anderen zwei Datenfeldern sind vielleicht deren Einzelwerte interessant, aber auch deren Summe oder Differenz. In beiden Fällen besteht eine Verbindung zwischen mehreren Feldern die evtl. auch in der Visualisierung sichtbar gemacht werden sollte. Metapher-Merkmale könnten schon vorab nach Kriterien gruppiert werden, die Aussagen darüber treffen, welche Art von Datenfeld-Kombinationen mit ihnen anschaulich dargestellt werden können. Auch die Bereitstellung eines Wizards wäre denkbar, der auf Grund der Beschreibung der Verhältnisse von darzustellenden Werten eine Auswahl passender Metapher-Merkmale anbietet oder sogar Vorschläge macht, welche Metapher dafür geeignet wäre. Dies könnte den Prozess zur Findung einer aussagekräftigen Darstellung besonders für numerische Daten entscheidend beschleunigen.

Die Möglichkeiten die der metaViewer bereits in dieser ersten Version bietet, lassen hoffen, dass er einen fruchtbaren Boden für viele weitere interessante Entwicklungen bildet.

Literaturverzeichnis

[1] Mehrnaz Sadat Akhavi, Mohammad Rahmati, and Nerssi Nasiri Amini. 3d visualization of hierarchical clustered web search results. In *Computer Graphics, Imaging and Visualisation, 2007. CGIV '07*, pages 441–446, August 2007.

[2] Jeff Baker, Donald Jones, and Jim Burkman. Using visual representations of data to enhance sensemaking in data exploration tasks. *Journal of the Association for Information Systems*, 10(7), 2009.

[3] Harald Bina. Visualisierung von digitalen bibliotheken. Master's thesis, TU Wien, 2000.

[4] Nicolas Bonnel, Alexandre Cotarmanac'h, and Annie Morin. Meaning metaphor for visualizing search results. *Ninth International Conference on Information Visualisation (IV'05)*, pages 467–472, 2005.

[5] Michael Chau. Visualizing web search results using glyphs: Design and evaluation of a flower metaphor. *ACM Transactions on Management Information Systems (TMIS)*, 2:27, March 2011.

[6] Herman Chernoff. Use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973.

[7] Geoffrey M. Draper, Yarden Livnat, and Richard F. Riesenfeld. A survey of radial methods for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15:759–776, September 2009.

[8] Jennifer Golbeck and Chang Hu. Impact of visualization methods on interaction with search results. *CHI 2009*, 2009.

[9] Susan Havre, Elizabeth Hetzler, Ken Perrine, Elizabeth Jurrus, and Nancy Miller. Interactive visualization of multiple query results. In *Proceedings IEEE Information Visualization Symposium*, pages 105–112. IEEE, 2001.

[10] Ilknur Icke. Visual analytics: A multifaceted overview. Technical report, CUNY, The Graduate Center, March 2009.

[11] Andreas Kerren and Ilir Jusufi. 3d kiviat diagrams for the interactive analysis of software metric trends. *Proceedings of the 5th ACM Symposium on Software Visualization (SoftVis '10)*, s. 203-204, 2010.

[12] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

[13] Yan Liu. Multivariate data visualization: A review from the perception aspect. In Michael Smith and Gavriel Salvendy, editors, *Human Interface and the Management of Information. Interacting with Information*, volume 6771 of *Lecture Notes in Computer Science*, pages 221–230. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21793-7 26.

[14] Monique Noirhomme-Fraiture. Visualization of large data sets: The zoom star solution. *The Electronic Journal of Symbolic Data Analysis*, 0(0), July 2002.

[15] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza. Visualizing multiple evolution metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization (SoftVis '05)*, pages 67–75, New York, NY, USA, 2005. ACM.

[16] Andreas Rauber and Dieter Merkl. The SOMLib digital library system. In *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries (ECDL '99)*, pages 323–342, London, UK, 1999. Springer-Verlag.

[17] Andreas Rauber and Alexander Müller-Kögler. Integrating automatic genre analysis into digital libraries. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '01)*, pages 1–10, New York, NY, USA, 2001. ACM.

[18] Arnaud Sallaberry, Faraz Zaidi, Christian Pich, and Guy Melançon. Interactive visualization and navigation of web search results revealing community structures and bridges. In *Proceedings of Graphics Interface 2010 (GI '10)*, pages 105–112, Toronto, Ont., Canada, 2010. Canadian Information Processing Society.

[19] Ben Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 3–12, New York, NY, USA, 2008. ACM.

[20] Greg Smith, Mary Czerwinski, Brian Meyers, Daniel Robbins, George Robertson, and Desney S. Tan. Facetmap: A scalable search and browse visualization.

IEEE Transactions on Visualization and Computer Graphics, 12:797–804, September 2006.

[21] Kenneth Treharne and David M. W. Powers. Search engine result visualisation: Challenges and opportunities. In *Proceedings of the 13th International Conference Information Visualisation (ICIV '09)*, pages 633–638, Washington, DC, USA, 2009. IEEE Computer Society.

[22] Edward R. Tufte. The visual display of quantitative information. *Graphics Press*, 1983.

[23] Ozgur Turetken and Ramesh Sharda. Clustering-based visual interfaces for presentation of web search results: An empirical investigation. *Information Systems Frontiers*, 7(3):273–297, July 2005.

[24] Jesús Vegas, Fabio Crestani, and Pablo De La Fuente. Context representation for web search results. *Journal of Information Science (JIS '07)*, 33:77–94, February 2007.

[25] Max L. Wilson, Bill Kules, Monica M. C. Schraefel, and Ben Shneiderman. From keyword search to exploration: Designing future search interfaces for the web. *Foundations and Trends in Web Science*, 2:1–97, January 2010.

[26] Songhua Xu, Tao Jin, and Francis C. M. Lau. A new visual search interface for web browsing. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM '09)*, pages 152–161, New York, NY, USA, 2009. ACM.

[27] C.C. Yang, Hsinchun Chen, and K. Hong. Internet browsing: visualizing category map by fisheye and fractal views. In *Proceedings International Conference on Information Technology: Coding and Computing*, pages 34 – 39, april 2002.

[28] Gang Zhang, Yue Liu, Songbo Tan, and Xueqi Cheng. A novel method for hierarchical clustering of search results. In *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology Workshops 2007*, pages 181 –184, November 2007.

[29] Kosara R. Ziemkiewicz, C. The shaping of information by visual metaphors. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1269–1276, 2008.