Humboldt-Universität zu Berlin
Institut für Bibliotheks- und informationswissenschaft

Berliner Handreichungen
zur Bibliotheks- und
Informationswissenschaft

Heft 275

SWD-Explorer

Design und Implementation eines Software-Tools zur erweiterten Suche und grafischen Navigation in der Schlagwortnormdatei

von
Jan Frederik Maas

SWD-Explorer

Design und Implementation eines Software-Tools zur erweiterten Suche und grafischen Navigation in der Schlagwortnormdatei

von
Jan Frederik Maas

Berliner Handreichungen zur
Bibliotheks- und Informationswissenschaft

Begründet von Peter Zahn

Herausgegeben von
Konrad Umlauf
Humboldt-Universität zu Berlin

Heft 275

Maas, Jan Frederik

SWD-Explorer : Design und Implementation eines Software-Tools zur erweiterten Suche und grafischen Navigation in der Schlagwortnormdatei / von Jan Frederik Maas. - Berlin : Institut für Bibliotheks- und Informationswissenschaft der Humboldt-Universität zu Berlin, 2010. – 4, III, 122 S. : graph. Darst. - (Berliner Handreichungen zur Bibliotheks- und Informationswissenschaft ; 275)

ISSN 14 38-76 62

Abstract:

Die Schlagwortnormdatei (SWD) stellt als kooperativ erstelltes, kontrolliertes Vokabular ein aus dem deutschsprachigen Raum nicht mehr wegzudenkendes Mittel zur Verschlagwortung von Medien dar. Die SWD dient primär der Vereinheitlichung der Verschlagwortung. Darüber hinaus sind in der Struktur der SWD Relationen zwischen Schlagwörtern definiert, die eine gut vorbereitete Suche stark erleichtern können. Beispiel für solche Relationen sind die Unterbegriff-/Oberbegriffrelationen (Hyponym/Hyperonym) oder die Relation der Ähnlichkeit von Begriffen. Diese Arbeit unternimmt den Versuch, durch die Erstellung eines Such- und Visualisierungstools den Umgang mit der SWD zu erleichtern. Im Fokus der Arbeit steht dabei zum einen die Aufgabe des Fachreferenten, ein Medium geeignet zu verschlagworten. Diese Aufgabe soll durch die Optimierung der technischen Suchmöglichkeiten mit Hilfe von Schlagwörtern geschehen, z.B. durch die Suche mit Hilfe Regulärer Ausdrücke oder durch die Suche entlang der hierarchischen Relationen. Zum anderen sind die beschriebenen Relationen innerhalb der SWD oft unsauber spezifiziert, was ein negativer Seiteneffekt der interdisziplinären und kooperativen Erstellung der SWD ist. Es wird gezeigt, dass durch geeignete Visualisierung viele Fehler schnell auffindbar und korrigierbar sind, was die Aufgabe der Datenpflege um ein Vielfaches vereinfacht.

Diese Veröffentlichung geht zurück auf eine Master-Arbeit im postgradualen Fernstudiengang Master of Arts (Library and Information Science) an der Humboldt-Universität zu Berlin.

Online-Version: http://edoc.hu-berlin.de/series/berliner-handreichungen/2010-275/

Inhaltsverzeichnis

1 Einleitung

> *"Chesire-Puss," she began, rather timidly, as she did not at all know whether it would like the name: however, it only grinned a little wider. "Come, it's pleased so far," thought Alice, and she went on. "Would you tell me, please, which way I ought to go from here?"*
> *"That depends a good deal on where you want to get to," said the Cat.*
> *L. Carroll a.k.a. C. Dodgson*[1]

In den vergangenen, wenigen Jahren haben sich in der Welt der Bibliotheken Veränderungen ergeben, deren Ergebnisse trotz ihrer Radikalität von uns heute als normal erachtet werden: Ein Nutzer sucht z.B. Monografien eines bestimmten Autors. Statt sich durch möglicherweise unangenehmes Wetter zu der Bibliothek seiner Wahl zu begeben, dort mit Hilfe eines freundlichen Bibliothekars große, gedruckte Zettelkataloge zu durchblättern um dann schließlich festzustellen, dass die gefundenen Werke nicht am Standort stehen, da sie verliehen sind, kann er heute einfach von zu Hause aus im Web-OPAC recherchieren, das gesuchte Buch bestellen oder vormerken und es dann zu einem geeigneten Zeitpunkt abholen.

Trotz der Nützlichkeit der modernen Techniken darf aber nicht übersehen werden, dass sie oft nur die Fortführung klassischer bibliothekarischer Arbeit mit neuen Mitteln sind. Die notwendige Frage ist in vielen Fällen also, wie man die bisher geleistete Arbeit optimal im Kontext der neuen Möglichkeiten nutzen, aber auch erweitern kann. Diese Arbeit sieht sich als ein Versuch, den Umgang mit einem klassischen bibliothekarischen Instrument – der *Verschlagwortung* – im Kontext moderner informationstechnischer Verfahren zu verbessern.

Zentral für diese Arbeit ist die Arbeit mit der Schlagwortnormdatei (SWD), die als kooperativ erstelltes, kontrolliertes Vokabular ein aus dem deutschsprachigen Raum nicht mehr wegzudenkendes Mittel zur Unterstützung der Verschlagwortung darstellt. Ziel der Verschlagwortung von Medien ist die Verbesserung der Recherchierbarkeit des Mediums, der Nutzer kann das Medium über mehr Suchbegriffe finden, als die reinen Titelstichwörter bilden. Dieser Umstand unterscheidet sich heute – in der Zeit der Suche per Texteingabe und OPAC – nur marginal von der Zeit der Zettelkataloge; ggf. ist jetzt z.B. die Notwendigkeit von Permutationen in Schlagwortketten – früher notwendig, damit jedes recherchierbare Schlagwort mindestens einmal durch alphabetische Suche auffindbar ist – nicht mehr gegeben.

Die SWD dient primär der *Vereinheitlichung* der Verschlagwortung. Darüber hinaus sind aber in der Struktur der SWD Relationen zwischen Schlagwörtern definiert, die eine geschickte Suche stark erleichtern können. Beispiel für solche Relationen sind die Unterbegriffs-/Oberbegriffsrelationen (Hyponym/Hyperonym) oder die Relation der Ähnlichkeit von Begriffen („Verwandte Begriffe"). Ein Nutzer, der z.B. unter einem Schlagwort keine passende Literatur gefunden hat, kann nach dem weniger spezifischen Begriff suchen (Oberbegriff). Ebenso kann andersherum eine zu große Trefferliste durch Einengung eines Schlagwortes – der Suche nach einem Unterbegriff – gezielt eingeschränkt werden. Trotz der großen Vorteile, die eine solche Strukturierung für

[1] (Carroll, 1982, S.39)

die Benutzung darstellt, entsteht für die Erstellung von Einträgen der SWD bzw. allgemein für die Erstellung von Einträgen in kontrollierten Vokabularen mit hierarchischen Relationen die zusätzliche Schwierigkeit, dass man eben auch das semantische Umfeld des Eintrags mit berücksichtigen muss.

Diese Arbeit stellt den Versuch dar, mit Hilfe der Erstellung eines Softwaretools zur Visualisierung der SWD den Umgang mit derselben zu erleichtern. Im Fokus der Arbeit steht dabei zum einen die Aufgabe des Fachreferenten, ein Medium geeignet zu verschlagworten. Diese Aufgabe soll durch die Optimierung der technischen Suchmöglichkeiten mit Hilfe von Schlagwörtern geschehen, z.B. durch die Suche mit Hilfe Regulärer Ausdrücke oder durch die Suche entlang der hierarchischen Relationen. Zum anderen sind die beschriebenen Relationen innerhalb der SWD trotz der umfangreichen und gut strukturierten Datenpflegebemühungen oft „unsauber“ spezifiziert, was ein Resultat der interdisziplinären und kooperativen Erstellung und Bearbeitung der SWD ist, aber auch ein Ergebnis ihrer Entstehungsgeschichte (nachträgliche Konvertieung etc.). Ich möchte zeigen, dass durch geeignete Visualisierung viele Fehler schnell auffindbar und korrigierbar sind, was die Aufgabe der Datenpflege um ein Vielfaches vereinfacht.

Die Arbeit gliedert sich wie folgt: Im Anschluss an diese Einleitung möchte ich zunächst auf die Verschlagwortung mit der Schlagwortnormdatei, also den bibliothekarischen Hintergrund der Arbeit eingehen (Kapitel 2), da die hier präsentierten Informationen zum Verständnis der folgenden Kapitel notwendig sind. Daraufhin stelle ich die Motivation dieser Arbeit sowie bisherige Arbeiten dar (Kapitel 3). Anschließend folgt eine detaillierte Darstellung der Software, die als Anleitung für Benutzer gedacht ist (Kapitel 4). Die für Benutzer weniger interessanten, technischen Details finden sich in Kapitel 5. Die Arbeit schließt mit einem Ausblick auf mögliche Weiterentwicklungen (Kapitel 6) und einem kurzen Fazit (Kapitel 7).

2 Methoden der Sacherschließung

Bibliotheken haben vielfältige Aufgaben, die vom Typ der Bibliothek, den Nutzergruppen, den verschiedenen Medientypen etc. abhängig sind. Eine der Kernaufgaben von Bibliotheken besteht jedoch im Umgang mit gedruckten und neuerdings auch elektronischen Medien, in jedem Fall Monografien, bei wissenschaftlichen Bibliotheken auch Tagungsberichten, Kongressen, Zeitschriften etc. Betrachtet man die Bibliothek als reine Sammlung von Werken, dann bestehen ihre Hauptaufgaben im Wesentlichen im *Sammeln*, *Erschließen* und *Vermitteln* von diesen Werken[2].

Die Erschließung von Literatur ist z.B. notwendig, um die jeweiligen Werke einer recherchierenden Person so einfach und schnell wie möglich zur Verfügung zu stellen. Im Gegensatz zu mittelalterlichen bzw. barocken Bibliotheken kann die Verortung des gesamte Inventars einer modernen Bibliothek nicht mehr im Kopf eines Bibliothekars vorhanden sein, daher sind gedruckte oder elektronische Kataloge notwendig bzw. üblich. Zum Recherchieren und Lokalisieren eines bekannten Werkes sollte es reichen, wenn ein Werk *formal* erschlossen im Katalog verzeichnet ist[3]. In diesem Fall kann z.B. einfach nach dem Titel gesucht werden. Möchte man eine Bibliografie eines bestimmten Autors erstellen, ist die formale Erschließung ebenfalls ausreichend, sofern ein Autorenkatalog bzw. ein nach Autorennamen indizierter elektronischer Katalog vorliegt. Ein grundsätzliches Merkmal der formalen Erschließung von Literatur ist, dass eindeutige[4] Regelwerke existieren, anhand derer ohne Kenntnis des Inhalts das jeweilige Werk katalogisiert werden kann. Beispiele für solche Regelwerke sind z.B. die „Regeln für die alphabetische Katalogisierung“ (RAK)[5] im deutschsprachigen Raum, bzw. die „Anglo-American Cataloguing Rules“ (AACR)[6], die im angloamerikanischen Raum üblich sind.

Auch wenn die Katalogisierung nach einheitlichen Regeln nach wie vor das wichtigste Instrument zur Erschließung von Werken ist, ist sie als solches nicht in jedem Fall ausreichend. Das gesamte Feld der *thematischen Literaturrecherche*, wie sie z.B. für die Erstellung einer Bibliografie eines Sachgebietes als Voraussetzung einer Dissertation notwendig ist, kann nur bruchstückhaft – durch die Suche nach Titelstichwörtern – abgedeckt werden. In diesem Fall muss als weiteres Hilfsmittel die inhaltliche Erschließung herangezogen werden.

Grundsätzlich kann jede Form von thematischer Literaturzusammenstellung oder Abstractbildung als inhaltliche Erschließung angesehen werden. Im Kontext des Bibliothekswesens existieren jedoch – nicht zuletzt auch aufgrund der großen Menge anfallender Literatur – insbesondere zwei Methoden: Die klassifikatorische und die verbale Sacherschließung[7]. Ich werde in den beiden folgenden Abschnitten genauer auf diese Erschließungsmethoden eingehen.

[2] vgl. (Hacker, 2000) S. 12

[3] Weiterhin muss das Katalogisat natürlich mit dem Standortbezeichner bzw. der Signatur versehen worden sein.

[4] In der Praxis können die jeweiligen Regeln natürlich unscharf sein, aber trotzdem besteht zumindest der Anspruch, möglichst eindeutige Regelwerke zu schaffen.

[5] z.B. für Wissenschaftliche Bibliotheken: (Popst, 1993)

[6] (AACR, 2002)

[7] vgl. (Hacker, 2000) S. 195ff., ebd. S. 201ff.

2.1 Klassifikatorische Sacherschließung

Die Klassifikatorische Sacherschließung ordnet Literatur genau einer oder mehreren thematischen Klassen zu. Die Klassen haben üblicherweise ein hierarchisches Verhältnis, so existiert z.B. in der u.a. im Gemeinsamen Bibiotheksverbund (GBV) verwendeten *Basisklassifikation* (BK)[8] die Kategorie 54.00 : Informatik mit der Unterkategorie 54.10 : Theoretische Informatik. Auf diese Weise können sowohl spezifische als auch allgemeine Werke in Bezug gesetzt werden. Wird die Klassifikation als Aufstellungssystematik verwendet, dann wird wie oben geschildert i.A. genau eine Klasse je Werk vergeben. Ein bekanntes Beispiel für eine als Aufstellungssystematik gedachte Klassifikation ist die *Dewey-Decimal-Classification* (DDC)[9].

2.2 Verbale Sacherschließung

Im Kontrast zu der Klassifikatorischen Sacherschliessung wird mit Hilfe der Verbalen Sacherschliessung versucht, durch die Spezifikation von den Inhalt besonders gut beschreibenden (Schlag-)Wörtern, das Werk für den Nutzer auffindbar zu machen. Ein Spezialfall der Verbalen Sacherschließung ist die syntaktische Sacherschließung, bei der der Inhalt in ganzen Sätzen wiedergegeben wird. Aufgrund mangelnder Zeit wird dieses Verfahren eher selten eingesetzt. Die Sacherschließung durch Schlagwörter kann durch festgelegte Vokabulare (sog. *Kontrollierte Vokabulare*) oder durch freie Schlagwörter geschehen. Auch wenn nur Schlagwörter vergeben werden, können diese (wie es für die Verschlagwortung mit Hilfe der Schlagwortnormdatei ebenfalls vorgesehen ist) Schlagwortketten bilden, die als solche eine eigene Bedeutung besitzen[10]. Ich möchte im Folgenden auf die Sacherschließung mit der Schlagwortnormdatei eingehen.

2.3 Sacherschließung mit der Schlagwortnormdatei (SWD)

1980[11] erhielt die Kommission für Sacherschließung des Deutschen Bibliotheksinstituts (DBI) von dem Fachbeirat desselben den Auftrag, ein einheitliches Regelwerk für die verbale Sacherschließung in öffentlichen und wissenschaftlichen Bibliotheken zu erstellen. Eine erste Version dieses Regelwerks – den „Regeln für den Schlagwortkatalog“ (RSWK) – wurde im Jahre 1986 fertiggestellt. In diesem Jahr führte die Deutsche Bibliothek (DB) diese Regeln ein und ersetzte damit das Prinzip des gleichgeordneten Indexierens, was u.a. Konsequenzen für die Deutsche Bibliografie hatte.

Grundlegendes Werkzeug für die Sacherschließung mit den RSWK innerhalb der DB war eine maschinell geführte Schlagwortnormdatei, die „Schlagwortliste“ (SWL). Diese wurde in Kooperation der DB und des Bayerischen Bibliotheksverbundes erstellt. In einem Projektantrag bei der DFG beantragten die beiden Partner, die SWL als *kooperativ* geführte Schlagwortnormdatei (SWD) umzusetzen, was am 1. Oktober 1988 bewilligt wurde. In den 90er Jahren schlossen

[8] (Recker-Kotulla, 1992)

[9] (Mitchell, 2003)

[10] So hat die Schlagwortkette `Geschichte - Weimarer Republik` die Bedeutung „Geschichte der Weimarer Republik“

[11] Zu dem folgenden Abschnitt vgl. v.a. (Werner, 1990) (Einleitung), (Capellaro, 2003) und (Bisig, 1994).

sich viele Bibliotheksverbünde der Arbeit an der SWD an, 1994 wurde durch die Hinzunahme der wissenschaftlichen Bibliotheken Österreichs sogar die Grenze Deutschlands überschritten. Mittlerweile ist die SWD – trotz aller gerechtfertigter Kritik – ein etabliertes Werkzeug zur Sacherschließung. Änderungen und Ergänzungen derselben werden auf praxisnaher Ebene z.B. von den Fachreferenten (o.ä.) der jeweiligen Bibliotheken vorgeschlagen und an Zentralredaktionen weitergeleitet. Die zentrale Aufsicht über die Arbeiten an der SWD hat die Zentralredaktion in der DB (heute: Deutsche Nationalbibliothek (DNB)).

Im folgenden möchte ich die theoretischen Hintergründe darstellen, die die Arbeit mit der Schlagwortnormdatei bedingen. Eine umfassende Einführung in die SWD kann dabei aufgrund der Komplexität der Materie nicht das Ziel sein, vielmehr muss ein grober, zielgerichteter Überblick genügen, bei dem im Detail schon Bezug auf die Eigenschaften des SWD-Explorers genommen wird. Die hier dargestellten Zusammenhänge sind – soweit nicht anders beschrieben – den Regeln für den Schlagwortkatalog[12] entnommen.

In der Schlagwortnormdatei wird grundsätzlich zwischen verschiedenen Typen von Schlagwörtern unterschieden. Die zu verwendenen Schlagworttypen sind[13]:

- Personenschlagwörter: Eindeutige Personennamen, wobei eng mit der Personennormdatei (PND) zusammengearbeitet wird. Zu beachten ist, dass nicht die Autoren, sondern die Personen, von denen ein Werk *handelt*, verzeichnet werden. Somit wäre „Das Kapital“ von Karl Marx im Sinne der Sacherschließung mit der SWD nicht mit dem Personendatensatz von Karl Marx zu verschlagworten, hingegen eine Fremdbiografie über Karl Marx schon.
- Geographische/ethnographische Schlagwörter einschließlich Sprachbezeichnungen
- Sachschlagwörter: Diese dienen der Verschlagwortung des Gegenstandsbereich eines Mediums, der nicht über die anderen Kategorien abgedeckt ist. Somit handelt es sich hierbei um eine sehr umfassende Kategorie, die – wie später ersichtlich ist – im Fokus dieser Arbeit steht.
- Zeitschlagwörter, z.B. „Geschichte 1939–1945“
- Formschlagwörter, z.B. „Formelsammlung“ oder „Lehrbuch“

Grundsätzlich werden die Schlagwörter in einer Schlagwortkette in der genannten Reihenfolge angeführt, wobei die Reihung der Schlagwörter eines einzelnen Typs weiteren Bestimmungen unterliegt. So können z.B. Sachschlagwörter in permutierenden Schlagwortketten notiert werden. Die Bildung mehrerer Schlagwortketten ist möglich, um Aspekte eines Werkes besser fassen zu können.

Alle Schlagwortkategorien und weitere Begriffskategorien werden i.A. mit Indikatoren versehen. Schlagwortindikatoren sind:

[12] (Kunz u. a., 2007)
[13] (Kunz u. a., 2007) S. 22

c Körperschaft, deren Ansetzungsform mit einem Geographikum beginnt
f Formschlagwort
g Geographisches/ethnographisches Schlagwort, Sprachbezeichnung
k Körperschaft (soweit nicht c)
p Personenschlagwort (in der PND durch die Satzart tp ersetzt)
s Sachschlagwort
t Titel eines Werkes
z Zeitschlagwort

Im Kontext dieser Arbeit möchte ich mich auf die Arbeit mit *Sachschlagwörtern* beschränken (weswegen auch den Schlagwortindikatoren keine besondere Beachtung geschenkt wird). Diese Einschränkung hat vor allem den Grund, dass insbesondere unter Sachschlagwörtern hierarchische Beziehungen existieren können, die eine strukturierte Navigation und Visualisierung von Interesse machen. Die Erweiterung des hier vorgestellten Tools um weitere Schlagworttypen ist unproblematisch, auch wenn eine Erweiterung des Suchraumes bei einer Suche ggf. zu mehr irrelevanten Treffern führen kann. So findet z.B. eine rechtstrunkierte Suche nach dem Schlagwort „Schraube“ auch alle Personen, die z.B. „Schraube“, „Schrauber“ etc. heißen.

2.3.1 Eigenschaften und Beziehungen von Sachschlagwörtern

Im Folgenden möchte ich die Eigenschaften von Sachschlagwörtern kurz umreißen[14]. Sachschlagwörter repräsentieren Allgemeinbegriffe oder Individualbegriffe. Sie sind in der Regel durch eine im Singular stehende Hauptansetzungsform vertreten, wobei auch Komposita möglich sind.

2.3.1.1 Ansetzungsformen Für ein Schlagwort wird der gebräuchlichste Begriff als Hauptansetzungsform gewählt[15]. Üblicherweise ist die Hauptansetzungsform in deutscher Sprache, es sei denn, ein fremdsprachiger Begriff ist die üblichste (vom Benutzer erwartete) Form und es existiert kein gleichwertiger Begriff in deutscher Sprache. Synonyme Bezeichnungen, ebenso wie veraltete Verwendungsformen oder einschlägige fremdsprachliche Bezeichnungen können und sollen ebenfalls als sog. Nebenansetzungsformen mit angegeben werden[16]. Nebenansetzungsformen werden als Synonymieverweisungen modelliert, der folgende Abschnitt 2.3.1.3 erläutert diese. Eine Software, die umfassende Suche auf Schlagwörtern ermöglichen soll, muss natürlich die Suche auf allen Ansetzungsformen ermöglichen. Dies wurde im SWD-Explorer umgesetzt.

2.3.1.2 Homonyme Sind Homonyme – also lexikalisch identische Schlagwörter differierender Bedeutung – vorhanden, werden sie mit Hilfe eines Zusatzes in spitzen Klammern disambiguiert. Z.B. existiert sowohl politische als auch medizinische Immunität, die entsprechenden Schlagwörter wären somit:

[14] vgl. (Kunz u. a., 2007) §§301–325 S. 113 ff.
[15] vgl. (Kunz u. a., 2007) §9 S. 19ff
[16] So kann z.B. die Haupansetzungsform des Schlagwortes „Amsel“ Amsel sein, eine weitere Ansetzungsform „Schwarzdrossel“. „Amsel“ ist die übliche Bezeichnung und wird daher als Hauptansetzungsform gewählt.

Immunität <Medizin> und
Immunität <Recht>.

Wenn ein Homonym besonders gebräuchlich ist, kann für diese Verwendungsform der Klammerzusatz entfallen. Homonymzusätze werden im SWD-Explorer ganz einfach als Teil des Namens / der Ansetzungsform eines Schlagwortes behandelt.

2.3.1.3 Verweisungen In der SWD können *Verweisungen* spezifiziert werden[17]. Dabei handelt es sich um SWD-interne Referenzen zwischen Einträgen (Begriffen) und Bezeichnungen auf andere Einträge oder Bezeichnungen. Eine mögliche Form von Verweisungen sind wie schon erwähnt *Synonymieverweisungen* von einer nicht als Hauptansetzungsform zugelassenen Form eines Schlagwortes (Bezeichnung bzw. Nebenansetzungsform) auf den mit der Hauptansetzungsform verknüpften Begriff.

2.3.1.4 Hierarchische Verweisungen Besonders relevant im Rahmen dieser Arbeit sind die *hierarchischen Verweisungen*[18]. Hierarchische Verweisungen existieren zwischen Unter- und Oberbegriffen. Dabei gibt es grundsätzlich zwei Typen von hierachischen Relationen, die auf diese Weise repräsentiert werden:

1. Die „klassische“ Abstraktionsrelation, bei der die Menge der durch den Unterbegriff spezifizierten Objekte eine Teilmenge der durch den Oberbegriff spezifizierten Objekte ist, wobei die Teilmenge durch die Hinzunahme einer spezifischen Eigenschaft gebildet wird

2. Die partitive Relation, bei der der Unterbegriff gewissermaßen physisch ein Teil des Oberbegriffs ist (Oberschenkel – Bein)

Als besonders interessant und als Motivation für diese Arbeit kann §12 3c der RSWK[19] gelten, in dem als Ziel eine vollständige und durchgehende Relationierung aller Sachschlagwörter gesetzt wird. Von diesem Ziel ist die SWD nach Erachten des Autors noch weit entfernt, was die Entwicklung und den Einsatz von Werkzeugen wie dem SWD-Explorer sinnvoll macht.

Ein Spezialfall der hierarchischen Verweisungen sind die Verweisungen zwischen einem Unterbegriff und einem mehrgliedrigem Oberbegriff[20] (einer Schlagwortkette). Sie werden z.B. bei Individualnamen verwendet und stellen die Verschlagwortung des Begriffs ohne die Verwendung des Individualnamens dar. Im SWD-Explorer werden diese Verweisungen auf spezielle Weise behandelt, mehr dazu in Abschnitt 4.5.3.3 auf Seite 26.

[17] vgl. §12 der RSWK, (Kunz u. a., 2007) S. 23ff.
[18] vgl. (Kunz u. a., 2007) S. 25ff.
[19] vgl. ebd.
[20] vgl. (Kunz u. a., 2007) §12.4

2.3.1.5 Verwandte Begriffe Eine weitere Relation ist die so genannte Assoziative Verweisung, speziell in Form der Verweisung von einem Schlagwort auf einen *verwandten* Begriff[21]. So hat das Schlagwort „Speicherzelle“ das Schlagwort „Speicher <Informatik>“ als verwandten Begriff. Relevant ist, dass sich verwandte Begriffe zwar von ihrer Bedeutung her überschneiden, aber in keiner hierarchischen Beziehung zueinander stehen. Verwandtschaftsbeziehungen werden im SWD-Explorer noch nicht visualisiert, vgl. dazu Abschnitt 4.5.3.4 auf Seite 27.

2.3.1.6 Weitere Eigenschaften von Schlagwörtern Aufgrund der hohen Komplexität der RSWK habe ich die Eigenschaften von Schlagwörtern nur äußerst exemplarisch skizziert. Über die bisherige Darstellung hinaus sollen/müssen für Schlagwörter noch weitere Eigenschaften spezifiziert werden, von denen ich die wichtigsten im Folgenden kurz aufzähle:

- Indikator, bei Sachschlagwörtern s bzw. (bei Wertktiteln) t (Angabe obligatorisch bei allen Sachschlagwörtern)
- Quelle – die Angabe ist obligatorisch, vgl. §19,1. Erwähnenswert ist, dass für die SWD nur eine festgelegte Menge an Quellen zur Verwendung zugelassen ist.
- Definition – die Angabe ist fakultativ, vgl. §19,2
- Verwendungshinweis – die Angabe ist fakultativ, vgl. §19,3
- Redaktionelle Bemerkung – die Angabe ist fakultativ, vgl. §19,4
- Notation – die Angabe ist obligatorisch, vgl. §18,1
- Ländercode – bei Schlagwörtern, die räumlich zugeordnet werden können; obligatorisch, wenn in einer Synonymie-Verweisung ein geographisches Schlagwort enthalten ist, vgl. §18,2
- Zeitcode – obligatorisch bei historischen Begriffen, die einen eingrenzbaren, engeren Zeitraum umfassen, vgl.§§18,4; 418,3

Nachdem ich die bibliothekarischen Hintergründe der Schlagwortnormdatei dargestellt habe, möchte ich im folgenden Kapitel auf die Motivation für die Erstellung des SWD-Explorers eingehen.

[21] vgl. (Kunz u. a., 2007) §12,5 S. 28 ff; §316; §317

3 Technischer Hintergrund und Motivation

Das Tool wurde grundsätzlich unter der Annahme von zwei verwandten Anwendungsszenarien erstellt:

1. Unterstützung der alltäglichen Sacherschließung im Fachreferat
2. Datenpflege, im Fachreferat oder in den Zentralredaktionen

Ich möchte in den folgenden Abschnitten kurz auf diese beiden Anwendungsfälle eingehen. Zuvor ist es jedoch nötig, die technischen Umstände der Entwicklung des SWD-Explorers zu schildern, da die Motivation der Entwicklung eines solchen Programms diesen direkt entspringt.

3.1 Technischer Hintergrund, PICA 3

Im Kontext des GBV – zu dem auch die TIB/UB Hannover gehört – wird zur Arbeit mit der SWD fast ausschließlich das Frontend „WinIBW“ verwendet, das die exakte und rechtstrunkierte Suche auf Schlagwortansetzungsformen (keine Unterscheidung zwischen Haupt- und Nebenansetzungsformen) ermöglicht. Je nach Konfiguration werden Unterbegriffe gefundener Schlagwörter ebenfalls mit angezeigt, die Darstellung erfolgt jedoch in einer unstrukturierten und unsortierten (bzw. nach Erstellungsdatum der Datensätze sortierten) Liste. Die WinIBW ist Teil des proprietären Bibliotheksverwaltungssystems PICA, welches im Kontext des GBV weitgehend Anwendung findet.

Die Schlagwortdatensätze werden üblicherweise in dem Format PICA 3 angezeigt, die Darstellung in anderen Formaten wie PICA+ oder im gefelderten Format ist möglich und per Voreinstellung wählbar. Da für diese Arbeit das konkrete Datenformat eine eher untergeordnete Rolle spielt – sofern die RSWK-Vorgaben in dem Format umgesetzt werden – wurde das im GBV übliche PICA 3-Format als Datengrundlage gewählt. Ein Beispiel für einen Schlagwortdatensatz im PICA 3-Format findet sich in Beispiel 1. Wie erkennbar ist, sind die Eigenschaften eines Schlagwortes in numerisch indizierten Kategorien festgehalten. Kategorie 800 enthält die Hauptansetzungsform, in 808 finden sich Angaben zur Quelle. Kategorie 830 enthält die synonymen Nebenansetzungsformen, die Oberbegriffe sind in der Kategorie 850 (die wie 830 mehrfach spezifiziert sein kann) angegeben. Nicht im Beispiel enthalten sind z.B. die Kategorien 860 (verwandte Begriffe) und 845 (mehrgliedrige Oberbegriffe), so wie zahlreiche weitere.

Das zentrale Problem dieser Darstellung ist die Modellierung der hierarchischen Oberbegriffs/Unterbegriffsrelation: Zum einen wird immer nur eine Ebene an Oberbegriffen angezeigt, die Oberbegriffe der Oberbegriffe zu finden kann also nur durch mühevolles Navigieren entlang den Datensätzen selbst geschehen. Zum anderen ist einem Datensatz nicht entnehmbar, ob es verknüpfte Unterbegriffe gibt. Beide Probleme werden durch die grafische Visualisierung im SWD-Explorer gelöst.

Der folgende Abschnitt beschäftigt sich mit der Frage, wie durch den SWD-Explorer für einen unter den dargestellten technischen Umständen arbeitenden Fachreferenten eine Arbeitserleichterung geschaffen werden kann.

3.2 Fachreferat

Anhand der folgenden *usecases* möchte ich den Nutzen einer erweiterten Suche und grafischen Navigation für die Arbeit eines Fachreferenten, der Medien verschlagworten will, exemplarisch darstellen.

1. Ein Fachreferent sucht nach einem Schlagwort, dessen Namen er selbst nicht genau kennt, z.B. einem bestimmten Schraubentyp. Durch Suche nach dem Schlagwort „Schraube“ kann er sich alle bekannten Schraubentypen (Unterbegriffe, inklusive der Unterbegriffe der Unterbegriffe etc.) sofort anzeigen lassen. Besonders vorteilhaft ist an dieser Stelle, dass der Referent auch Schrauben finden würde, die die Zeichenkette „Schraube“ nicht im Namen tragen.

2. Der Referent sucht alle Schlagwörter, deren Ansetzungsform den Namen „Lyer“ enthalten. Bspw. weiß der Verschlagwortende nicht, ob mehrere mathematische Gleichungen existieren, die nach dieser Person benannt sind. Die Suche nach `.*lyer.*` als Regulärem Ausdruck findet alle Ansetzungsformen, die die Zeichenkette „lyer“ an einer beliebigen Stelle besitzen.

3. Der Fachreferent sucht nach dem Schlagwort HD DVD. Mögliche Schreibweisen für diesen Datenträger sind HD DVD, HD-DVD, HDDVD etc. Wenn der Schlagwortdatensatz nicht sauber spezifiziert wurde, kann es sein, dass erst mehrere Suchbegriffe eingegeben werden müssen, bis der gesuchte Schlagwortdatensatz gefunden oder sein Fehlen bemerkt wurde. Durch die Suche nach der Zeichenkette `HD.?DVD` als Regulärem Ausdruck werden alle genannten Ansetzungsformen gefunden. Auf diese Weise können auch Mehrfachansetzungen unter unterschiedlichen Datensätzen – Dubletten – gefunden bzw. verhindert werden. Dass dublette Ansetzungen durchaus einen realen Faktor in der SWD darstellen, zeigt Abbildung 2. Hier findet sich eine Dublette, die im Rahmen der Arbeit mit dem SWD-Explorer gefunden, der Zentralredaktion Sacherschließung an der SUB Göttingen gemeldet und mittlerweile korrigiert wurde.

4. In bestimmten Fällen muss ein Fachreferent ein noch nicht vorhandenes Schlagwort neu ansetzen, bevor er ein Dokument verschlagworten kann. Durch Betrachtung des hierarchischen Kontextes analoger Schlagwörter und durch die Möglichkeit, deren Datensätze

```
005 Tsv
021 46159228
800 |s| Paraphrasenrelation
808 |b| Synonymiebeziehung zwischen Saetzen u. Aussagen
808 |a| Metzler-Lex. Sprache und Bussmann als Paraphrase 2.
810 11.2b
830 |s| Paraphrase <Synonymie>
850 |s| Synonymie
```

Abbildung 1: Beispiel für einen PICA 3-Datensatz

```
005 Tsv
021 41332799
800 |s| Kognitive Entwicklung
808 |a| Dorsch
810 5.2
830 |s| Kind / Entwicklung / Kognition
830 |s| Kind / Kognitive Entwicklung
830 |s| Kognition / Entwicklung
830 |s| Denkentwicklung
830 |s| Denken / Entwicklung
830 |s| Geistige Entwicklung / Kognition
850 |s| Entwicklung
860 |s| Kognitive Kompetenz
```

```
005 Tsv
021 41490770
800 |s| Denkentwicklung
808 |a| Dorsch
830 |s| Denken / Entwicklung
850 |s| Kognitive Entwicklung
850 |s| Entwicklung
```

Abbildung 2: Beispiel für eine Dublette in der SWD: Mehrfache Ansetzung des Schlagwortes „Denkentwicklung“

einfach zu kopieren und angepasst zu übernehmen, kann auch dieser Prozess beschleunigt werden.

Diese Beispiele sind natürlich nicht erschöpfend, aber ich hoffe, den Nutzen der Software für die Alltagsarbeit im Fachreferat hinreichend motiviert zu haben.

3.3 Datenpflege

Die für das Fachreferat genannten Benutzungsvorteile gegenüber bisherigen Werkzeugen lassen sich natürlich teilweise 1:1 auf die Aufgabe der Datenpflege – z.B. durch die Zentralredaktionen – übertragen. Ein weiterer großer Vorteil, den die Visualisierung mit sich bringt, besteht in der schnellen – da „auf einen Blick“ möglichen – Entdeckung bestimmter Fehler:

Aufgrund der Bearbeitung der SWD durch Programme, die nur eine flache Ansicht der hierarchischen Relationen erlauben, können bei der Ansetzung neuer Schlagwörter in der SWD strukturelle Fehler – z.B. zyklische Relationsketten[22] – geschehen. Der häufigste strukturelle Fehler besteht in der Verwendung eines Oberbegriffs, der gleichzeitig Oberbegriff eines anderen Oberbegriffs für das Schlagwort ist.

Ein einfaches Beispiel findet sich in Abbildung 3 auf der nächsten Seite: Das Schlagwort „Strömungsmechanik“ hat „Technische Mechanik“ und „Mechanik“ als Oberbegriffe, wobei aber „Mechanik“ Oberbegriff von „Technische Mechanik“ ist. Zunächst handelt es sich um einen strukturellen Fehler, da der Umstand, dass „Mechanik“ Oberbegriff von „Strömungsmechanik“ ist, schon

[22] Hiermit ist gemeint, dass ein Schlagwort Unterbegriff eines Unterbegriffs (eines Unterbegriffs...) von sich selbst ist, weswegen in der Vererbungshierarchie eine Schleife entsteht.

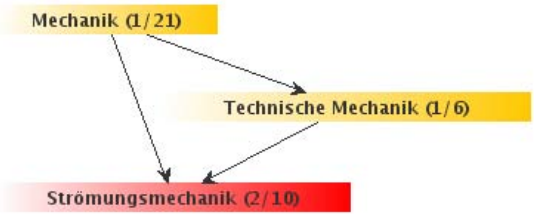

Abbildung 3: Struktureller Fehler in der SWD, Beispiel Strömungsmechanik

transitiv über „Technische Mechanik“ dargestellt wurde[23]. In diesem Fall liegt aber sogar ein inhaltlicher Fehler vor: „Technische Mechanik“ ist *kein* korrekter Oberbegriff von „Strömungsmechanik“, da sich die Stömungsmechanik auch mit natürlich vorkommenden Strömungen beschäftigen kann. Der Fehler wurde an der TIB/UB mit Hilfe des SWD-Explorers im Fachreferat Maschinenbau identifiziert und bereits durch die Zentralredaktion Sacherschließung Göttingen korrigiert. Das Beispiel zeigt, wie man sogar inhaltlichen Fehlern durch eine Visualisierung auf die Spur kommen kann.

3.4 Konkurrierende Ansätze

Visualisierungen der SWD gibt es bereits. Um die zwei bekanntesten zu nennen:

- Es existiert eine maschinenlesbare Form der SWD, die als Daten-CD von der Deutschen Nationalbibliothek regelmäßig veröffentlicht wird. Auf dieser CD befindet sich ebenfalls eine Software zur grafischen Suche in der SWD.
- Die DNB bietet unter `http://melvil.d-nb.de/swd`[24] eine webunterstützte Suche in der SWD an, die die Schlagwortrelationen in einer Baumdarstellung repräsentiert.

Diesen und allen anderen, dem Autor bekannten Visualisierungen ist gemein, dass sie zur Darstellung mehrerer Oberbegriffe eines Schlagwortes mehrere Darstellungsbäume aufspannen, so dass das Schlagwort[25] mehrfach angezeigt werden muss. Ein einfacher Überblick über die Oberbegriffe bzw. die Anzahl der Oberbegriffe eines Schlagwortes wird so verhindert, die parallele Darstellung mehrerer verschiedener Schlagwörter – wie der SWD-Explorer sie in der OPAC-ähnlichen Suche anbietet[26] würde extrem unübersichtlich werden. Die im SWD-Explorer sehr einfache Navigation entlang von Unter- und Oberbegriffsrelationen ist nur eingeschränkt möglich.

Weiterhin unterstützen alle dem Autor bekannten Visualisierungen der SWD höchstens die exakte und rechtstrunkierte Suche auf den Ansetzungsformen. Die Suche mit Hilfe fortschrittlicher

[23] Oberbegriffsbeziehungen sind i.A. transitiv, der Oberbegriff eines Oberbegriffs von a ist Oberbegriff von a. Der Grund dafür ist die i.A. als Teilmengenbeziehung definierte Relation zwischen den denotierten Objekten des Oberbegriffs und des Unterbegriffs.

[24] Stand vom 17.02.2009

[25] Z.B. „Gießereimechaniker“ als Unterbegriff von „Gießer“ und „Mechaniker“, vgl. dazu das Beispiel 4 auf Seite 14 und das Beispiel 5 auf Seite 15.

[26] vgl. Abschnitt 4.4.6.3

Suchwerkzeuge wie Regulärer Ausdrücke oder dem OPAC-Stil wird bisher nur von dem SWD-Explorer unterstützt, was eine gravierende Legitimation für eine solche Software darstellt. Auf Dauer wäre es natürlich nützlich, die Funktionen des SWD-Explorers in die Standardtools zu integrieren, aber auch hierfür kann der SWD-Explorer als einfache Machbarkeitsstudie von großem Nutzen sein.

Nachdem ich in dem vorliegenden Abschnitt auf den Nutzen der Software eingegangen bin, möchte ich nun zu dem Kernteil der Arbeit kommen, nämlich der Dokumentation der Software für Benutzer.

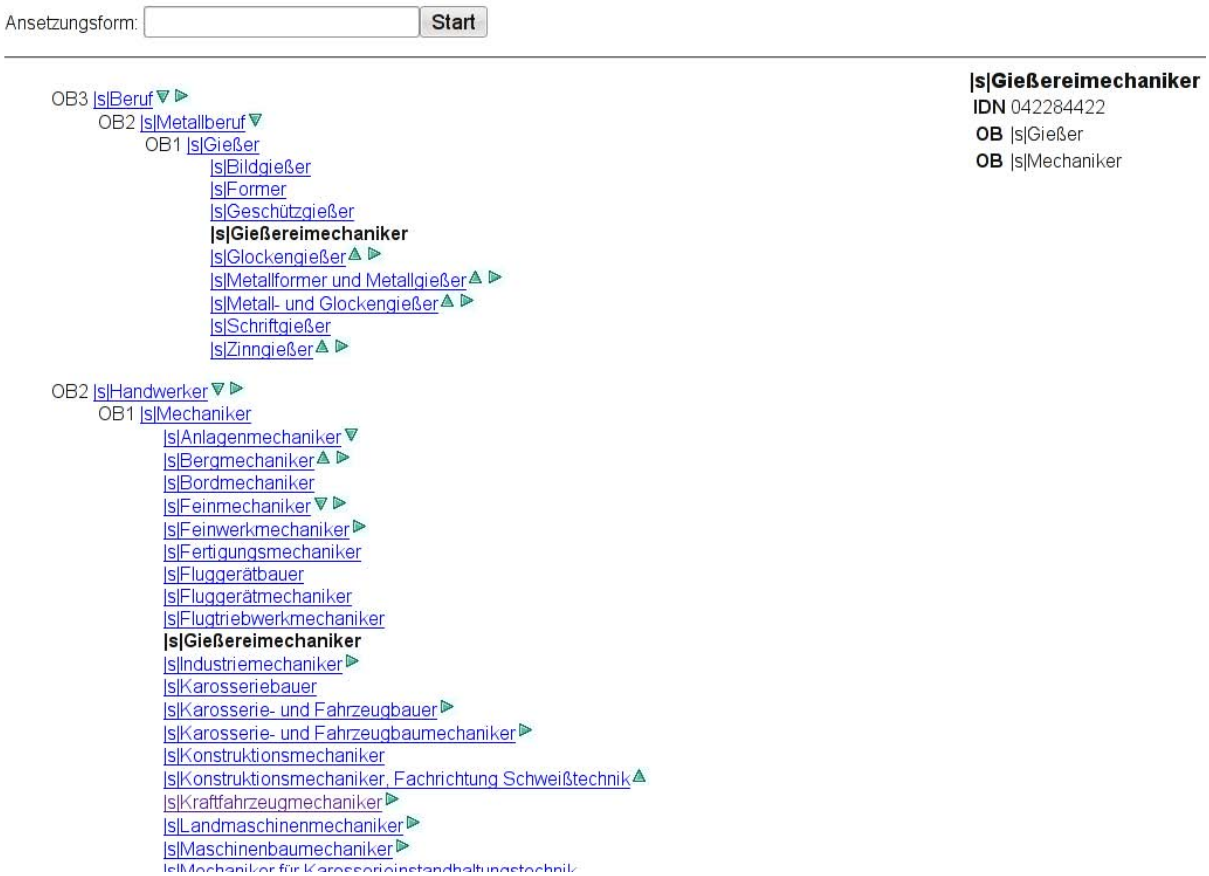

Abbildung 4: Darstellung des Schlagwortes „Gießereimechaniker“ in der strukturierten Darstellung auf den Seiten der DNB. Stand vom 16.2.2009. Aufgrund der gewählten Darstellungsform wird das Schlagwort je Oberbegriffsbaum einmal angezeigt.

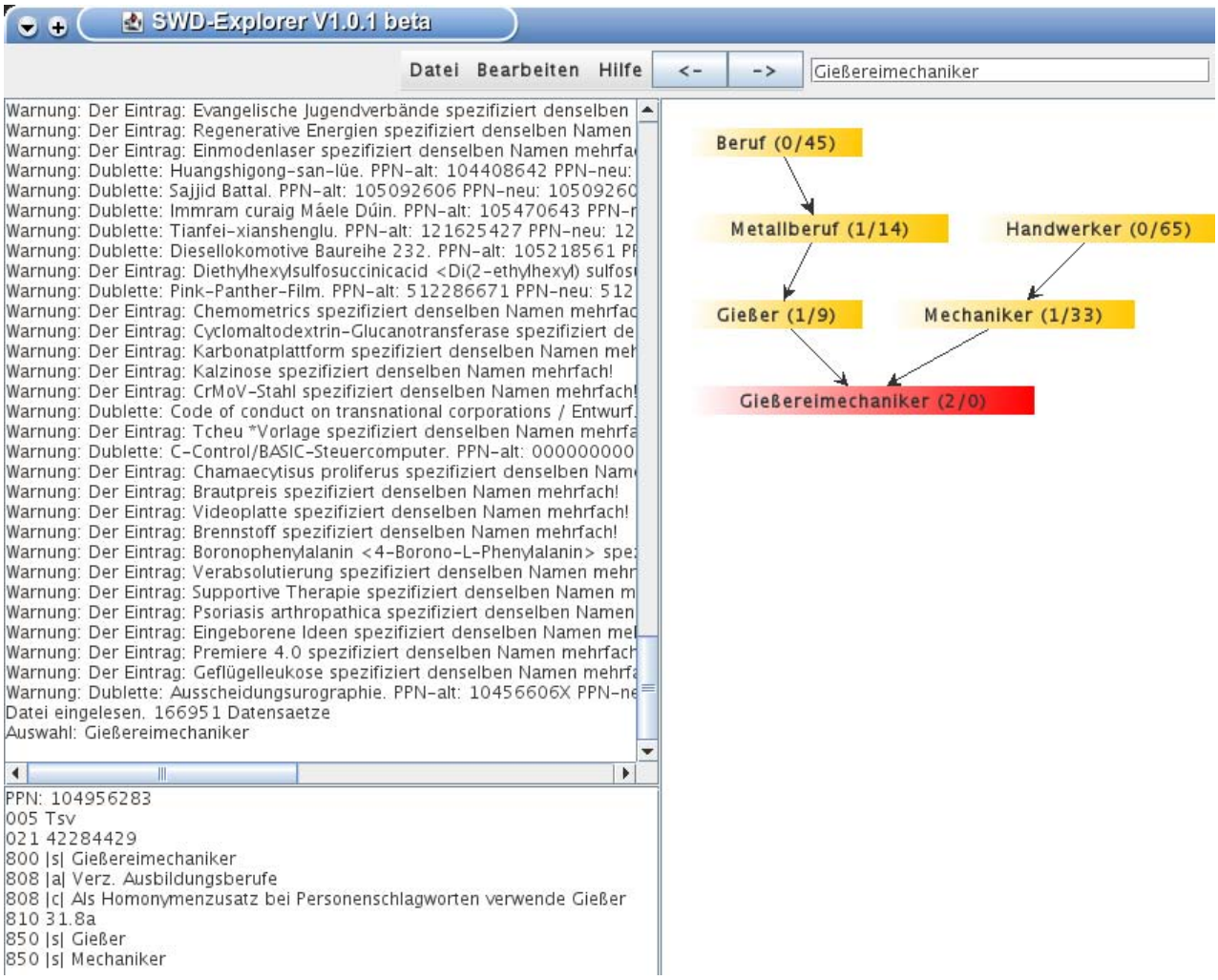

Abbildung 5: Darstellung des Schlagwortes „Gießereimechaniker“ mit Hilfe des SWD-Explorers. Alle Oberbegriffe werden verknüpft dargestellt, das Schlagwort „Gießereimechaniker“ muss nur ein einziges mal angezeigt werden.

4 SWD-Explorer: Benutzerhandbuch

Die grundsätzliche Funktionalität des SWD-Explorers besteht in der hierarchischen Visualisierung von Schlagwörtern der Schlagwortnormdatei, wobei einzelne Schlagwörter oder Schlagwortgruppen möglichst komfortabel durch textuelle Suche auffindbar sein sollen. Weiterhin besteht die Möglichkeit der grafischen Navigation durch die SWD, um so z.B. anhand der hierarchischen Relationen zwischen einzelnen Schlagwörtern – also Hyponym-Beziehungen (Oberbegriffe) und Hyperonym-Beziehungen (Unterbegriffe) semantisch weitere oder engere, verwandte Schlagwörter zu finden.
In dem folgenden Abschnitt möchte ich die Benutzung des SWD-Explorers skizzieren. Dazu werde ich detailliert auf die einzelnen Bedienelemente und Funktionen der Software eingehen[27]. Sinnvollerweise fahre ich jedoch mit den notwendigen Schritten fort, die zur Installation und zum Starten des Programms erforderlich sind.

4.1 Systemvoraussetzungen

Folgende Voraussetzungen müssen minimal erfüllt sein, damit der SWD-Explorer grundsätzlich lauffähig ist:

Hardwarevoraussetzungen:

- Prozessor mit mind. 1 GHz Taktfrequenz
- Mindestens 500 MB freier Arbeitsspeicher (entspricht mindestens 768MB RAM bei handelsüblichen PCs und Windows XP)
- Empfohlen wird ein Monitor mit mindestens 1280x1024 Pixel Auflösung, da sonst die Darstellung aufgrund mangelnder Fläche ggf. unübersichtlich wird. Niedrigere Auflösungen werden aber unterstützt.
- Maus o.ä.

Softwarevoraussetzungen:

- Sun Java Version 1.6 oder höher
- JGraph Version 5.9.2 oder höher
- Grafische Oberfläche[28]

[27] Vgl. Abschnitt 4.3 ff.
[28] Unter Linux/Unix z.B. ein laufender X-Server, bei Microsoft Windows standardmäßig aktiv

4.2 Installation und Start

Eine eigene Installation des Tools ist nicht notwendig. Es reicht, das Programm mit allen benötigten Dateien in einen *für den Benutzer schreibbaren* Ordner zu kopieren. Durch Ausführen der Scriptdatei `start.bat` (Windows) bzw. durch Starten von Java mit der .jar-Datei als Parameter (Unix/Linux) `java -Xmx700m -jar SWDexplorer.jar` wird der SWD-Explorer gestartet. In beiden Fällen muss jedoch JGraph in einer ausreichend hohen Version im Java-Klassenpfad (`classpath`) liegen. Dies wird üblicherweise über die Definition einer Umgebungsvariable oder direkt über den Java-Parameter `-cp` getan; bei Schwierigkeiten sollte die Dokumentation der verwendeten Java-Distribution zu Rate gezogen werden.

4.3 Grafische Benutzeroberfläche des Explorers

Nach Start des Programms erscheint der in Abbildung 6 dargestellte Bildschirm.

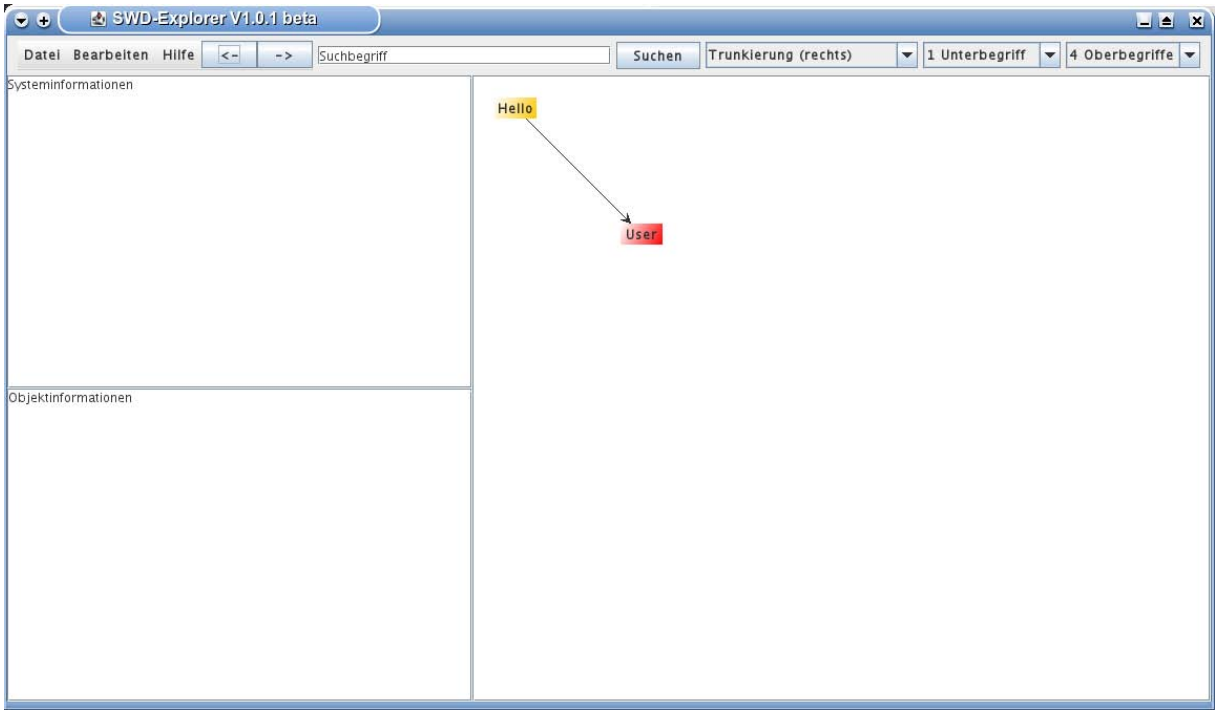

Abbildung 6: Ansicht des SWD-Explorers nach dem Start

Der untere Teil der GUI[29] besteht aus drei Fenstern.

Sie dienen der Ausgabe von Systeminformationen, Datensätzen und der Visualisierung ausgewählter Teile der SWD. Auf alle drei Fenster wird in Abschnitt 4.5 auf Seite 23 detailliert eingegangen werden.

Das zentrale Element zur Steuerung des SWD-Explorers ist die oberhalb der Fenster gelegene Menüleiste. Ich werde ihre Funktionen im folgenden Abschnitt beschreiben.

[29] GUI steht für „graphical user interface“ = grafische Benutzeroberfläche

4.4 Menüleiste

Die Menüleiste verfügt über ein klassisches Pull-Down-Menü mit den Untermenüs `Datei`, `Bearbeiten` und `Hilfe`. Rechts davon befinden sich zwei Buttons, mit denen man in den bisherigen Suchvorgängen vor- und zurückblättern kann. Noch weiter rechts befindet sich das eigentliche Fenster zur Eingabe von Suchbegriffen, gefolgt von einem Pull-Down-Menü zur Auswahl der Anfrage-Sprache (im Beispiel: Trunkierung (rechts), so wie zwei Menüs zur Auswahl der Tiefe der Verschachtelung bei der Anzeige von gefundenen Schlagwörtern (im Beispiel: 1 Unterbegriff.bzw. 4 Oberbegriffe).

4.4.1 Menü „Datei“

Im Dateimenü (vgl. Abbildung 7) finden sich die Unterpunkte „Datei öffnen“, „Speichere alle Hauptansetzungsformen in...“, „Speichere alle Ansetzungsformen in...“ und „Beenden“. Außerdem können bis zu vier schon vorher geöffnete Dateien direkt zum Öffnen ausgewählt werden.

Abbildung 7: Das Datei-Menü des SWD-Explorers

4.4.1.1 Öffnen von Dateien Der SWD-Explorer kann einen oder mehrere SWD-Abzüge einlesen. Die Datensätze können in Form von einzulesenden Dateien über den Menüpunkt „Datei öffnen“ ausgewählt werden. Die letzten vier geladenen Dateien können wie geschildert über ein Schnellzugriffsmenü direkt geladen werden. Es ist zu beachten, dass das Einlesen mehrerer Dateien dazu führt, dass sich die Datensätze parallel im Speicher befinden, wobei ggf. schon vorhandene Schlagworteinträge von gleichnamigen neuen überschrieben werden. Im Fall des Überschreibens warnt der SWD-Explorer, dass eine Dublette gefunden wurde. Eine Löschung des Speichers des SWD-Explorers ist nur durch Neustart des Programms möglich.

Durch passende Einteilung der SWD in Dateien – z.B. unter Hinzuziehung der SWD-internen Systematik – ist es möglich auf thematischen Partitionen der SWD zu arbeiten, aber auch die gesamte SWD gleichzeitig einzulesen.

4.4.1.2 Erstellen von SWD-Abzügen

Eine grundsätzliche Problematik im Umgang mit der SWD besteht in der schlechten Verfügbarkeit[30] derselben. Ein offener Online-Zugriff existiert nicht, weswegen bedauerlicherweise auf Offline-Abzügen gearbeitet werden muss. Diese unterliegen natürlich der Gefahr, veraltet zu sein.

Für diese Arbeit wurde daher mit Datenbankabzügen gearbeitet, die mit Hilfe der Pica-WinIBW erzeugt wurden. Es ist möglich mit dem Befehl `rec n; f mak Ts? and nad s`[31] alle Sachschlagwort-Normsätze der SWD in einem Set zu speichern. Mit Hilfe des Befehls `dow <Setnummer> d` lassen sich dann die gefundenen Datensätze im PICA 3-Format[32] in eine Textdatei schreiben. Textdateien dieser Art lassen sich ohne weitere Modifikationen in den SWD-Explorer einlesen. Ggf. kann – um Abstürze der WinIBW-Software anzufangen, die Suchanfrage noch auf Ansetzungsformen beschränkt werden, die mit einem bestimmten Buchstaben beginnen; die Resultate der Suchanfragen können dann anschließend durch Aneinanderhängung der Dateien verknüpft werden. Um den Suchbefehl auf Ansetzungsformen einzuschränken, die z.B. mit dem Buchstaben „a“ beginnen, kann die Suchanfrage im GBV-Katalog wie folgt formuliert werden: `rec n; f mak Ts? and nad s and ans a?`. Zu beachten ist allerdings, dass zusätzlich eine Suchanfrage formuliert werden muss, die alle Ansetzungsformen, die mit keinem Buchstaben beginnen, abfragt. Eine solche Suchabfrage wäre:

`rec n; f mak Ts? and nad s and not ans a? not ans b? not ans c? not ans d? (...).`

Die Eingabe dieser Anweisung ist leider etwas umständlich. Weiterhin übersteigt sie knapp die Länge einer erlaubten Eingabe zumindest bei den alten WinIBW-Versionen. Aus diesem Grund – aber auch aus Gründen der Zeiteffizienz – sollten durch diese Anweisung die selten vorkommenden Buchstaben x, y und z zusammen mit den sonstigen Zeichen abgefragt werden[33].

Grundsätzlich arbeitet der SWD-Explorer mit dem PICA 3-Format, wobei allerdings die von der WinIBW für jedes Schlagwort erzeugten Header zusätzlich ausgewertet werden. In diesen Headern befindet sich die Pica-Produktionsnummer (PPN), die für die Arbeit mit Schlagwörtern z.B. im Kontext des Gemeinsamen Bibliotheksverbundes von sehr großem Nutzen sein kann, da sie jeden Schlagwortdatensatz eindeutig identifiziert. Ein Beispiel für einen Datensatz findet sich in Abbildung 8.

Die auf diese Weise gewonnenen Dateien lassen sich wie geschildert über das Datei-Menü des SWD-Explorers öffnen.

4.4.1.3 Speichern von (Haupt-)Ansetzungsformen

Weiterhin ist es möglich, über die gleichnamigen Punkte die (Haupt-)Ansetzungsformen, die sich z.Zt. im Speicher befinden, in

[30] Bezogen auf die Möglichkeit, aktuelle Online-Abzüge in einem generischen Format wie z.B. XML oder einem spezifischen Format wie PICA 3 zu erstellen.

[31] `rec n;` schränkt die Suchanfrage auf alle Normdatensätze ein. `f mak Ts?` findet nur die Datensätze von Schlagwörtern. `nad s` beschränkt die Suche auf die Sachschlagwörter. Dieser Befehl funktioniert zumindest im Zentralen Katalogisierungssystem (CBS) des GBV, andere Systeme benötigen aufgrund unterschiedlicher Indexierungen ggf. andere Anweisungen.

[32] Je nach Konfiguration der WinIBW auch in anderen Formaten.

[33] Dies gelingt durch die `and not`-Spezifizierung aller Buchstaben ausser x,y und z in dem Suchstring.

```
 SET: S7 [6489] TTL: 6456      PPN: 10411424X      SEITE1 .

Eingabe: 2012:26-12-92 Aenderung: 2012:29-03-93 22:31:00
Status: 2012:29-03-93

005 Tsv
021 42745913
800 |s| AIX <Betriebssystem>
808 |a| Lex. Informatik unter UNIX
810 30m
830 |s| IBM AIX
830 |s| Advanced Interactive Executive
850 |s| Betriebssystem
850 |s| UNIX
```

Abbildung 8: Beispiel für einen Schlagwortdatensatz im PICA 3-Format mit einem durch die WinIBW erzeugten Header

einer Textdatei zu speichern. Dies kann in bestimmten Fällen, z.B. bei der Suche nach dubletten Ansetzungsformen nützlich sein.

4.4.1.4 Beenden

Dieser Punkt beendet das Programm.

4.4.1.5 Schnellwahlmenü

Unterhalb des Trennungsstriches werden bis zu vier zuletzt geöffnete Dateien zur Schnellwahl angezeigt.

4.4.2 Menü „Bearbeiten“

Mit Hilfe dieses Menüs (vgl. Abbildung 9) lassen sich einzelne Schlagwortdatensätze ganz oder teilweise kopieren. Dazu muss ein Schlagwort zuvor durch einen einfachen Klick mit der rechten Maustaste ausgewählt worden sein, so dass der Schlagwortdatensatz im Datenfenster unten links angezeigt wird. Jetzt lassen sich die PPN, die Hauptansetzungsform oder der gesamte Datensatz in die Zwischenablage des Betriebssystems kopieren. Auf diese Weise kann man z.B. über die PPN ein im SWD-Explorer gefundenes Schlagwort direkt in der WinIBW aufrufen oder für einen Änderungsvorschlag einen Datensatz direkt in eine EMail kopieren.

4.4.3 Menü „Hilfe“

In diesem Menü verbergen sich trivialerweise Hinweise zur Programmversion.

Abbildung 9: Das Bearbeiten-Menü des SWD-Explorers

4.4.4 Buttons „Letzte Suche/Nächste Suche“

Die beiden Buttons mit einem Pfeil nach links und einem Pfeil nach rechts (Letzte Suche/Nächste Suche) ermöglichen wie in einem Internet-Browser den Zugriff auf die Suchhistorie. Dabei kann beliebig weit (bis zur ersten Eingabe nach dem Programmstart) in den Sucheingaben rückwärts und – wenn zwischendurch keine neuen Suchanfragen getätigt wurden – wieder vorwärts navigiert werden. Zu beachten ist, dass die Darstellungsoptionen[34] *nicht* abgespeichert werden – die Ausgaben werden immer anhand der aktuellen Konfiguration angezeigt. Dieses Verhalten des Explorers ist beabsichtigt, damit schon durchgeführte Suchen schnell und einfach mit neuen Suchparametern betrachtet werden können.

4.4.5 Eingabefenster

In diesem Fenster werden die Sucheingaben getätigt. Eine Suche wird durch Drücken von „Return“ oder durch Anklicken des „Suchen“- Knopfes ausgeführt.

4.4.6 Format der Sucheingabe

In diesem Menü kann ausgewählt werden, in welchem Format die Sucheingaben getätigt werden. Folgende Möglichkeiten stehen zur Auswahl;

1. Exakte Übereinstimmung
2. Trunkierung (rechts)
3. OPAC-Stil
4. Regulärer Ausdruck

[34] Format der Sucheingabe, Anzahl Ebenen Unterbegriffe, Anzahl Ebenen Unterbegriffe

Ich werde die verschiedenen Sucheingaben im Folgenden beschreiben. Grundsätzlich gilt, dass der SWD-Explorer nicht zwischen Groß- und Kleinschreibung unterscheidet. Eine solche Differenzierung schien im Kontext der SWD nicht sinnvoll.

4.4.6.1 Exakte Übereinstimmung In diesem Fall wird das Schlagwort angezeigt, das eine Haupt- oder Nebenansetzungsform besitzt, die mit dem eingegebenen Begriff exakt übereinstimmt. Leerzeichen (z.B. „HD DVD“) sind erlaubt.

4.4.6.2 Trunkierung (rechts) Es werden alle Schlagwörter angezeigt, die mindestens eine Haupt- oder Nebenansetzungsformen besitzen, die mit dem Anfang des Suchbegriffs übereinstimmt. So lassen sich in diesem Modus z.B. durch Eingabe des Suchbegriffs „Thyristor“ u.a. die Schlagwörter „Thyristor“, „Thyristorsteuerung“, „Thyristorschaltung“ etc. finden.

4.4.6.3 OPAC-Stil Der OPAC-Stil ermöglicht folgende Suchoptionen:

- Trunkierung der Suchbegriffe mit dem Dollarzeichen ($)
- Verknüpfung von Suchanfragen mit „AND“ – es werden alle Schlagwörter gefunden, die sowohl dem ersten als auch dem zweiten Suchschlüssel entsprechen
- Verknüpfung von Suchanfragen mit „OR“ – es werden alle Schlagwörter gefunden, die dem ersten oder dem zweiten Suchschlüssel entsprechen
- Klammerung von Suchanfragen

Einer der Hauptnutzen der Suche im OPAC-Stil ist die parallele Darstellung von verschiedenen Schlagwörtern z.B. durch Veroderung, so kann man z.B. vor der Ansetzung eines interdisziplinären Schlagwortes Oberbegriffe aus beiden Teildisziplinen anzeigen lassen. Weiterhin wurde diese Suchmöglichkeit implementiert, da sie für viele Bibliothekare und Nutzer durch den täglichen Umgang mit dem OPAC vertraut ist.

4.4.6.4 PPN Die im Kontext des GBV vergebenen Pica-Produktionsnummern (PPN) können auch direkt eingegeben werden, um z.B. ein in der WinIBW gefundenes Schlagwort anzuzeigen.

4.4.6.5 Regulärer Ausdruck Die Suchoption „Regulärer Ausdruck“ stellt die gesamte Mächtigkeit regulärer Ausdrücke zur Verfügung. Reguläre Ausdrücke sind im Bereich der maschinellen Textverarbeitung ein sehr mächtiges und weit verbreitetes Werkzeug zur Suche (und in erweiterter Form zur Editierung), weswegen sie im SWD-Explorer implementiert wurden.

Mit Hilfe Regulärer Ausdrücke lassen sich z.B. Suchanfragen wie `.*Stokes.?(Gleich|equat).*` gestalten: Der Punkt „.“ ist Platzhalter für ein beliebiges Zeichen, der Asterix-Operator „*“ steht

für null bis beliebig viele Zeichen des vorangegangenen Typs. Somit entspricht „.*“ einer beliebig langen (auch leeren) Zeichenkette beliebiger Zeichen. Der Operator „?“ steht für ein oder kein Zeichen des vorangegangenen Typs. Zeichenketten, die in einer Klammer mit einem senkrechten Strich kombiniert werden, gelten als Alternativen. Somit ist der Ausdruck interpretierbar als „Finde alle Schlagwörter, deren Haupt- oder Nebenansetzungsformen die Zeichenkette ‚stokes‘[35] enthalten, wenn diese von einem oder keinem beliebigen Zeichen gefolgt wird und dann die Zeichenkette „gleich“ oder die Zeichenkette „equat“ folgt“. Durch diese Suchanfrage lassen sich mit großer Wahrscheinlichkeit alle deutschen *und englischen* Schlagwörter finden, die den Namen dieser Gleichung in einer ihrer Ansetzungsformen tragen.

Reguläre Ausdrücke sind im Detail dann z.B. nützlich, wenn unbekannt ist, wie ein Kompositum gebildet wurde. So lassen sich die Schlagwörter „Stokesgleichung“, „Stokes-Gleichung“ und „Stokes Gleichung“ alle mit der Suchanfrage „`stokes.?gleichung`“ finden.

Wenn keine Operatoren angegeben sind, entspricht die Suche nach Regulären Ausdrücken der Option „Exakte Übereinstimmung“.

Das Format der Regulären Ausdrücke entspricht dem der Programmiersprache Java. Somit entspricht es weitgehend dem POSIX-Standard, Abweichungen lassen sich in der Java-Dokumentation[36] nachlesen.

4.4.7 Menüs „Oberbegriffe / Unterbegriffe“

In diesen Menüs lässt sich definieren, wieviele *Ebenen* von Unter- und Oberbegriffen relativ zu einem gefundenen Schlagwort angezeigt werden. Die Beschriftung ist somit irreführend, da sie andeutet, dass die Anzahl der gefundenen Unter- und Oberbegriffe eingestellt werden kann; in der geringen Breite des Menüs ließ sich jedoch keine bessere Beschriftung finden.

In Abbildung 10 auf der nächsten Seite findet sich ein Beispiel für die Suche nach dem Schlagwort „Halbleiterschaltung“ mit der Anzeige von einer Unterbegriffsebene und zwei Oberbegriffsebenen. Zu beachten ist, dass die Unterbegriffe „Integrierte Schaltung“, „Thyristorschaltung“, „Halbleiterdiodenschaltung“ und „Triacschaltung“ zur selben Ebene gehören – die Darstellung in mehreren Ebenen geschieht nur aufgrund der beschränkten Bildschirmbreite. Auf die Details der Darstellung wird in Abschnitt 4.5 genau eingegangen.

4.5 Anzeige und grafische Navigation

In diesem Abschnitt möchte ich im Detail auf die Anzeigeelemente des Explorers eingehen, sowie auf die grafische Navigation mit einer Maus.

[35] Wie oben geschildert wird Groß- oder Kleinschreibung ignoriert.
[36] vgl. z.B. http://java.sun.com/javase/6/docs/api/ zuletzt 16.02.2009

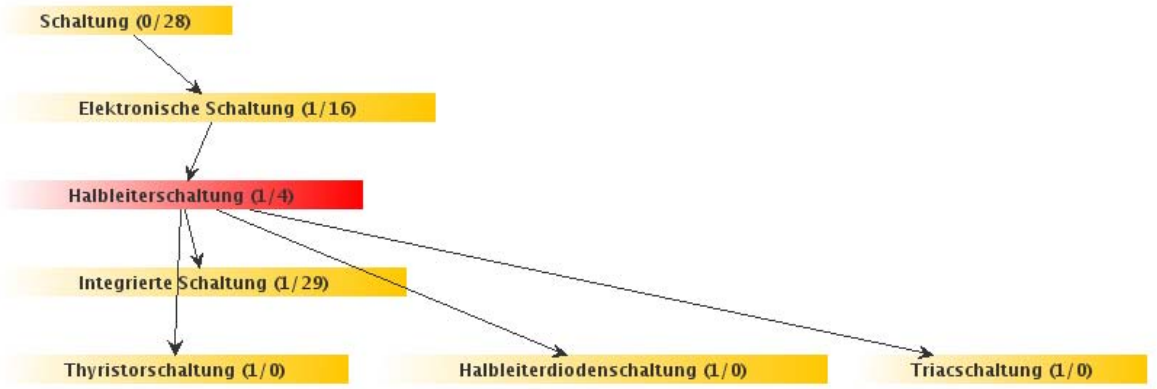

Abbildung 10: Beispiel für die Anzeige von einer Unterbegriffsebene und zwei Oberbegriffsebenen

4.5.1 Ausgabe von Systeminformationen

Wie aus Abbildung 11 auf der nächsten Seite ersichtlich, besteht der untere Teil des SWD-Explorers aus drei Fenstern. Das Fenster links oben (2) dient der Ausgabe von Systeminformationen. So wird hier z.B. nach Einlesen eines Datensatzes die Anzahl der geladenen Datensätze, der Dubletten und der fehlerhaft spezifizierten Schlagwörter angezeigt. Weiterhin werden hier alle Suchanfragen aufgeführt, um auf diese Weise einen Überblick über die Suchhistorie zu erhalten. Sollte eine Suchanfrage kein Ergebnis geliefert haben, erscheint ebenfalls hier eine passende Information. Grundsätzlich werden hier alle Warnungen, Fehlermeldungen und allgemeinen Informationen des Systems ausgegeben. Zu beachten ist, dass der Explorer diese Informationen ebenfalls in eine Logdatei schreibt (standardmäßig im Arbeitsverzeichnis unter dem Namen `SWDlog.log`).

4.5.2 Ausgabe von Objektinformationen

Im Fenster unten links (3) wird bei Selektion eines der visualisierten Schlagwörter durch die rechte Maustaste der Datensatz des Schlagwortes angezeigt. Im Fall von zusammengesetzten Oberbegriffen (in Abbildung 11 z.B. `Postscript / Font`) wird ein künstlich generierter Eintrag angezeigt, vgl. dazu Abschnitt 4.5.3.3 auf Seite 26.

Wie oben beschrieben (vgl. Abschnitt 4.4.2) können die Inhalte dieses Fensters komplett oder teilweise über das Bearbeiten-Menü in die Zwischenablage des Betriebssystems kopiert werden.

4.5.3 Visualisierung und grafische Navigation

Das Fenster auf der rechten Seite (1) stellt das Herzstück der grafischen Ausgabe dar. Alle Schlagwortdatensätze, die durch eine Suchanfrage gefunden wurden, so wie die entsprechenden Ober- und Unterbegriffe[37] werden hier in Baumstrukturen dargestellt.

[37] Vgl. Abschnitt 4.4.7 auf der vorherigen Seite, die Anzahl der angezeigten Ebenen kann nahezu beliebig gewählt werden.

4.5.3.1 Darstellung der Schlagwörter Schlagwörter werden in farbigen Kästchen visualisiert, die von zwei Zahlen in Klammern gefolgt werden (z.B. `Font (1/10)`). Es gibt insgesamt drei Farben:

- **Rote** Kästchen[38] entsprechen den Schlagwörtern, die genau durch die Suchanfrage gefunden wurden
- **Orange** Kästchen[39] sind Schlagwörter, die zwar nicht von der Suchanfrage gefunden wurden, aber die Unter- oder Oberbegriffe gefundener Schlagwörter sind.
- **Grüne** Kästchen[40] entsprechen *mehrgliedrigen Oberbegriffen*. Ich werde in Abschnitt 4.5.3.3 im Detail auf sie eingehen. Erwähnenswert ist, dass mehrgliedrige Oberbegriffe auch durch Suchanfragen gefunden werden können. In diesem Fall werden sie rot dargestellt.

4.5.3.2 Darstellung der Hierarchie Das wichtigste Element der Darstellung der hierarchischen Relationen sind die Pfeile zwischen den Schlagwörtern. Ein Pfeil von Schlagwort a nach b bedeutet, dass a ein Oberbegriff von b ist (Ausnahmen im Abschnitt 4.5.3.3). Weiterhin werden Oberbegriffe und Unterbegriffe auf unterschiedlichen Ebenen dargestellt, die Schlagwörter werden also von oben nach unten immer spezifischer.

[38] Im Beispiel `Zeichensatz`.
[39] Im Beispiel `Adobe Type 1 Font`.
[40] Im Beispiel `PostScript / Font`.

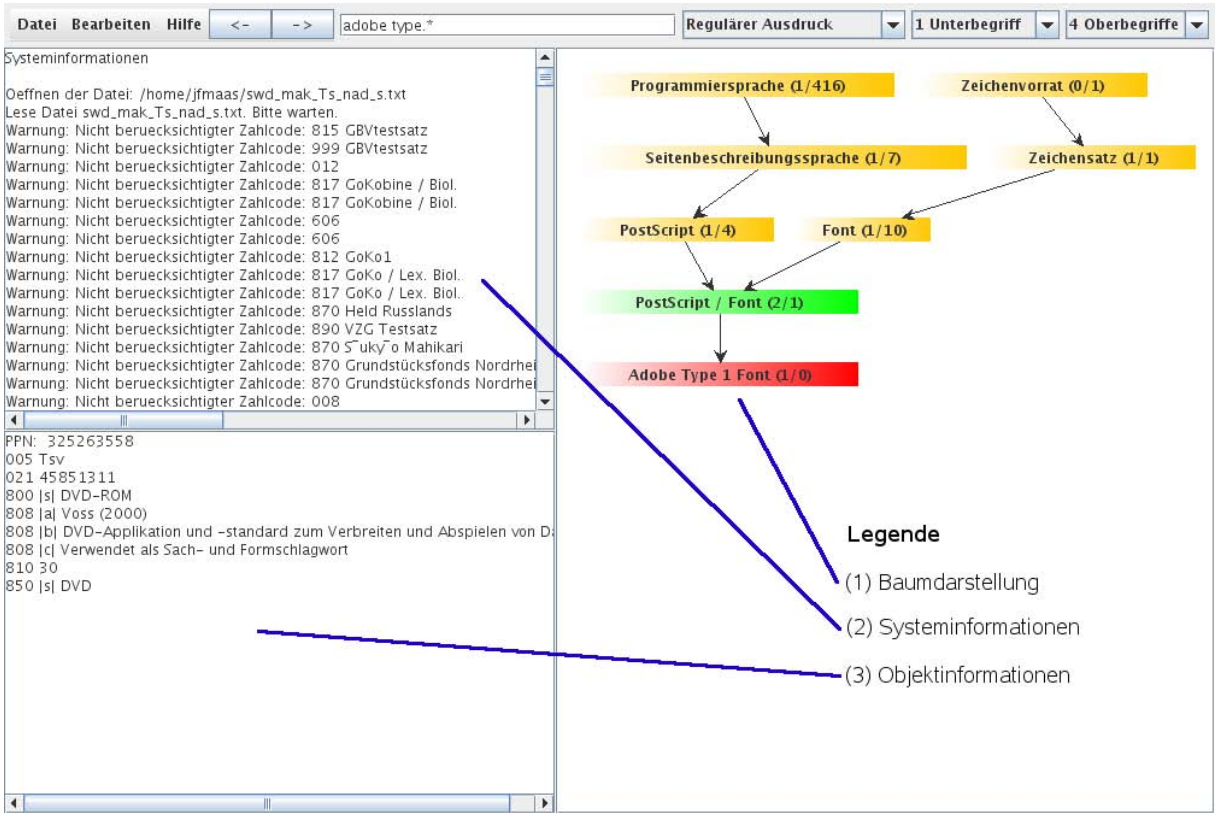

Abbildung 11: Fensteraufbau des SWD-Explorers

Schlagwörter, die nebeneinander dargestellt sind, teilen sich meistens ein oder mehrere Oberbegriffe. Zu beachten ist, dass zur Optimierung der Darstellung Zeilenumbrüche vorgenommen werden, wenn die Zeile eine bestimmt Breite überschreitet.

Grundsätzlich wird jedes gefundene Schlagwort in einem eigenen Baum mit den entsprechenden Unter- und Oberbegriffen dargestellt. Sollten sich zwei oder mehr Schlagwörter die Unter- oder Oberbegriffe teilen, so werden die Bäume verschmolzen.

Einzelne Schlagwörter – also Schlagwörter ohne Unter- und Oberbegriffe – werden *nebeneinander* an der obersten Position des Ausgabefensters angezeigt. Dies dient der Verbesserung der Übersichtlichkeit.

4.5.3.3 Behandlung mehrgliedriger Oberbegriffe

Manche Schlagwörter verfügen über zusammengesetzte (mehrgliedrige) Oberbegriffe. Ein Beispiel für einen mehrgliedrigen Oberbegriff findet sich in Abbildung 12; der Oberbegriff „PostScript / Font“ besteht aus den Einzelschlagwörtern „PostScript“ und „Font“. „Font“ alleine wäre zwar ein adäquater Oberbegriff, „PostScript“ allerdings nicht, da eine Schrift keine Begriffsverengung von „PostScript“ darstellt. Somit wurde ein zusammengesetzter Oberbegriff gebildet, bei dem Font noch zusätzlich durch das Schlagwort „PostScript“ in der Bedeutung spezifiziert wurde.

```
005 Tsv
021 42805387
800 |s| Adobe Type 1 Font
808 |a| Vorlage
810 30m- 2.1- 31.14
830 |s| PostScript-Type-1-Schrift
830 |s| Type 1 Font
845 |s| PostScript / Font
845 |s| Font / PostScript
```

Abbildung 12: Datensatz im PICA 3-Format mit zusammengesetztem Oberbegriff PostScript / Font

Im Kontext des SWD-Explorers werden zusammengesetzte Oberbegriffe weitestgehend wie normale Oberbegriffe behandelt. Allerdings werden sie in der grafischen Ausgabe grün hinterlegt, sofern sie nicht direkt durch eine Suche gefunden und somit rot hinterlegt werden. Außerdem haben im Zusammenspiel mit zusammengesetzten Oberbegriffen die Pfeile in der grafischen Darstellung eine etwas andere Bedeutung: Die Pfeile, die auf einen zusammengesetzten Oberbegriff weisen, stammen von den einzelnen Teilen des zusammengesetzten Oberbegriffs. Somit handelt es sich nicht um eine echte Oberbegriffs/Unterbegriffsrelation. Trotzdem wird diese Beziehung aus Gründen der Übersichtlichkeit identisch dargestellt und behandelt.

Ein Beispiel für die Anzeige von mehrgliedrigen Oberbegriffen findet sich in Abbildung 11 auf der vorherigen Seite.

Da zusammengesetzte Oberbegriffe keinen eigenen Datensatz besitzen, wird bei der Selektion des Oberbegriffs mit der rechten Maustaste ein („virtueller“) Platzhalter-Datensatz erstellt. Ein Beispiel findet sich in Abbildung 13.

```
PPN: 000000000
Kein Datensatz vorhanden. Zusammengesetzter Begriff.
(Alternativer) Name: PostScript / Font
(Alternativer) Name: Font / PostScript
Virtueller Oberbegriff: PostScript
Virtueller Oberbegriff: Font
```

Abbildung 13: Virtueller Datensatz des zusammengesetzten Oberbegriffs PostScript / Font

4.5.3.4 Bemerkung: Behandlung von verwandten Begriffen Wie weiter oben beschrieben, können in der SWD *verwandte Begriffe* spezifiziert werden. Diese stehen in keiner hierarchischen Relation zueinander, überschneiden sich jedoch in ihrer Bedeutung. Im PICA 3-Format werden verwandte Begriffe in der Kategorie 860 angegeben.

Obgleich eine Visualisierung bzw. eine grafische Navigation entlang der Verwandtschaftsrelationen im SWD-Explorer vermutlich wünschenswert wäre, wurde diese bisher nicht umgesetzt. (Verwandte Begriffe lassen sich natürlich durch Betrachtung des Datensatzes eines Schlagwortes finden, somit sind sie im SWD-Explorer nicht unsichtbar.) Der Grund für die mangelnde Behandlung von verwandten Begriffen ist, dass noch keine adäquate Visualisierungsform für dieselben gefunden wurde. Schon die Baumdarstellung der Unter- und Oberbegriffsrelationen ist trotz diverser Heuristiken zur Verbesserung der Übersichtlichkeit in Extremfällen schwer zu entziffern, die Hinzunahme einer weiteren Relation würde dies noch stark verkomplizieren.

Für die Zukunft ist ggf. eine „Reise“ von einem Schlagwort zu seinen verwandten Begriffen über eine Leiste von Knöpfen denkbar, die sich bei Auswahl des Schlagwortes öffnet. Da aber auch von den an der Evaluation beteiligten Fachreferaten bisher noch nicht nach einer solchen Modifikation gefragt wurde, hatte die Implementierung eines solchen Mechanismus bisher keine Priorität. Die Vermutung liegt nahe, dass verwandte Begriffe i.A. in der Sacherschließung eine eher untergeordnete Rolle spielen oder dass verwandte Begriffe bei einer geschickten Suche im SWD-Explorer gleich mit gefunden werden.

4.5.3.5 Grafische Interaktion Der SWD-Explorer unterstützt drei Formen von Interaktion mit der grafischen Ausgabe:

1. **Selektion:** Ein beliebiges Schlagwort kann mit einem einfachen Klick der rechten Maustaste selektiert werden. Dies hat die Anzeige des Datensatzes im Objektinformationsfenster zur Folge, so dass mittels des Bearbeiten-Menüs auf den Datensatz zugegriffen werden kann.

2. **Navigation:** Durch doppeltes Anklicken mit der rechten Maustaste kann ein beliebiges Schlagwort in den Mittelpunkt der Betrachtung gehoben werden. Das Anzeigeergebnis ist dasselbe, als ob exakt nach dem Schlagwort gesucht worden wäre. Auf diese Weise kann man einfach Schlagworthierarchien expandieren und uninteressante Information ausblenden.

3. **Verschiebung:** Durch „Drag-and-Drop“ kann mit Hilfe der linken Maustaste ein Schlagwort verschoben werden, um z.B. die Übersichtlichkeit der Darstellung noch zu verbessern.

4.6 Bekannte Fehler

Keine Software ist fehlerfrei. Natürlich habe ich im Kontext dieser Arbeit versucht, alle mir bekannten Fehler zu beseitigen. Diejenigen, die trotzdem in der Abgabeversion des SWD-Explorers vorhanden sind, sind hier aufgelistet.

4.6.1 Probleme bei fehlender Homonymkennzeichnung

Ein gravierendes Problem tritt auf, wenn in dem eingelesenen SWD-Datensatz Homonyme vorkommen, die nicht – wie in den RSWK gefordert – disambiguiert wurden (vgl. Abschnitt 2.3.1.2). Ein Beispiel war hierfür das Sachschlagwort LED. Üblicherweise würde ein technisch versierter Mensch unter dieser Abkürzung eine Nebenansetzungsform des Schlagwortes „Leuchtdiode“ erwarten. Da jedoch das Schlagwort „Systemischer Erythematodes“ als Nebenansetzungsform nicht disambiguiert den Eintrag „LED“ hatte, wurde dieser Sucheinstieg für „Leuchtdiode“ überschrieben. Das Problem wurde bereits an die Zentralredaktion gemeldet und behoben, der Autor verfügt aber noch über einen älteren Datensatz, in dem der Fehler vorhanden ist.

Grundsätzlich ist dieses Verhalten erwünscht, da auf diese Weise aktuelle Daten ohne Neustart der Software in das System eingelesen werden. Da bei korrekter Anwendung der RSWK dieses Problem nicht auftreten würde und eine technische Kompensation dieses Fehlers das Programm stark verlangsamen würde, wurde er nicht behoben.

4.6.2 Keine Terminierung bei zu umfangreichen Suchen

Da nicht absehbar ist, auf welchen Rechnersystemen der SWD-Explorer zum Einsatz kommt, wurde bisher noch keine automatische Terminierung des Suchprozesses implementiert. Bestimmte Suchen, die auf einzelnen Systemen aufgrund von Speicher- und Rechengeschwindigkeitsbeschränkungen nicht möglich sind, können auf anderen funktionieren. Trotzdem wäre aber eine zeitbasierte Terminierung des Suchprozesses („suche nicht länger als 20 Sekunden“) möglich.

Bis eine zukünftige Version des SWD-Explorers dies implementiert, liegt es an den Nutzern des Programms, z.B. auf Suchen nach allen Schlagwörtern, die ein „e“ enthalten, zu verzichten, da diese den Absturz des Programms zur Folge haben.

5 Technische Aspekte

In diesem Abschnitt möchte ich für den interessierten Leser kurz auf die technischen Details der Arbeit eingehen.

5.1 Gestaltungskriterien

Der SWD-Explorer wurde darauf konzipiert, eine modulare Stand-Alone-Anwendung zu sein, die als Open-Source-Projekt publizierbar ist.

5.1.1 Open-Source

Auch wenn die Software noch nicht als Open-Source-Projekt publiziert wurde, wurde doch sehr viel Wert darauf gelegt, dies zu ermöglichen. Es wurden grundsätzlich nur Softwarebibliotheken verwendet, die selbst open-source sind bzw. in Open-Source-Projekten verwendet werden dürfen. Ein beabsichtigter Seiteneffekt ist, dass Bibliotheken sich die Software ohne Rechteverletzung kopieren und ohne zusätzliche Kosten verwenden dürfen.

5.1.2 Modularität

Bei dem Design der Software wurde viel Wert darauf gelegt, sie modular zu gestalten. Auf diese Weise kann die Funktionalität der Software einfach erweitert werden – z.B. durch die saubere, objektorientierte Trennung der Datenspeicherung kann das Tool verhältnismäßig einfach mit einer externen Datenbankanwendung wie MySQL verbunden werden. Auch die einfache Erweiterung durch Dritte im Kontext eines Open-Source-Projektes wurde so ermöglicht.

5.1.3 Stand-Alone

Die Software wurde als Stand-Alone-Produkt konzipiert, die jedesmal ihre Daten erneut aus einem Datenbankabzug einlesen muss. Im Gegensatz zu einer netzbasierten Implementation, die stets auf den aktuellen Stand der SWD zugreifen könnte, ergeben sich die Nachteile des hohen Speicherverbrauchs so wie der Nicht-Aktualität, auch wenn der Vorteil der sehr hohen Verarbeitungsgeschwindigkeit gewonnen wird. Der Hauptgrund für die Offline-Herangehensweise besteht in der nicht vorhandenen Möglichkeit, online über eine offene Schnittstelle auf die SWD zuzugreifen. Dies kann sich in Zukunft ändern, war aber im Kontext dieser Arbeit nicht realisierbar.

Auf die Anbindung an eine Datenbankanwendung, die die Schlagwortnormdatensätze speichert, wurde aus zwei Gründen verzichtet: Zum einen würde eine Netzanbindung der Software diese Herangehensweise obsolet machen, zum anderen soll die Software möglichst einfach installierbar sein. Da die Installation einer Datenbankanwendung leider die Rechte oder Fähigkeiten mancher Nutzer übersteigt, wurde auf diese Herangehensweise bisher verzichtet. Trotzdem ist auf mittelfristige zeitliche Distanz ein solcher Schritt zu überlegen.

5.2 JGraph

Zur Unterstützung der grafischen Darstellung mittels der Java-eigenen Toolkits AWT und Swing wurde die Graph-Visualisierungsbibliothek JGraph[41] verwendet. Diese Bibliothek ist quelloffen und frei verfügbar und stellt mittlerweile einen Standard in der Java-gestützen Graphvisualisierung dar. Problematischerweise sind die Graph-Layout-Algorithmen Teil von JGraph-Pro, dem proprietärem und kostenpflichtigem Zusatzpaket. Aus diesem Grund wurde nur auf sehr basale Funktionen von JGraph zugegriffen, das Layouten der Graphenstruktur erfolgt wie im folgenden Abschnitt beschrieben über die vom Autor durchgeführte Implementierung des Sugiyama-Layout-Algorithmus.

5.3 Verwendete Algorithmen

Die verwendeten Algorithmen wurden entweder von Autor selbst erfunden, sind Standardverfahren, sind Teil der Java-Klassenbibliotheken oder sind Teil der JGraph-Bibliotheken, die hauptsächlich für die Grafikausgabe herangezogen wurden.

Eine Ausnahme dieser Regel sind die Algorithmen zur Visualisierung. Da trotz aufwändiger Recherche keine Implementierungen solcher Algorithmen entsprechend des Open-Source-Paradigmas zu finden waren, wurde in dieser Arbeit der Standardalgorithmus von Sugiyama[42] zur Baum-Darstellung gerichteter Graphen verwendet. Eine detaillierte Darstellung des Algorithmus findet sich u.a. in (Di Basstista u. a., 1999) S. 265ff. Der Sugiyama-Algorithmus ist in vier Teilschritte untergliedert:

1. Cycle Removal
2. Layer Assignment
3. Crossing Reduction
4. Horizontal Coordinate Assignment

Jeder der vier Schritte kann auf unterschiedliche Weise implementiert werden, daher möchte ich im Folgenden die Besonderheiten der Implementation im SWD-Explorer darstellen.

5.3.1 Parallele Darstellung mehrerer Graphen

Wenn man im Wesentlichen nur die Datensätze der SWD und die hierarchischen Beziehungen zwischen denselben betrachtet, so liegt eine baumartige Struktur vor, welche in Form von azyklischen, gerichteten Graphen darstellbar ist. Der wohl wichtigste Unterschied zur Visualisierung eines gerichteten Graphen mit der Hilfe des Sugiyama-Algorithmus besteht darin, dass die darzustellenden Elemente in verschiedenen, unverbundenen Graphenstrukturen vorliegen können,

[41] Homepage: `http://www.jgraph.com`, zuletzt am 15.10.2008
[42] Sugiyama u. a. (1981), Sugiyama (2002)

von denen einige sogar nur aus einem Element bestehen. Aus diesem Grund werden in einem ersten Schritt die einzelnen Teilgraphen trivial bestimmt. Die unverbundenen Elemente werden in der ersten Reihe nacheinander dargestellt, für die restlichen Teilgraphen wird jeweils einzeln der Sugiyama-Algorithmus angewandt. Die gefundenen unverbundenen Elemente und strukturierten Graphen werden anschließend nacheinander angezeigt.

5.3.2 Cycle Removal

Ein zyklischer, gerichteter Graph hat die Eigenschaft, dass ein Element des Graphen durch Navigation entlang der gerichteten Relationen von sich selbst ausgehend wieder erreicht werden kann. Ein Beispiel: Element A verweist auf Element B, das auf Element C verweist. Verweist Element B oder Element C (oder beide) wieder auf A, existiert ein Zyklus. Die Eigenschaft der Azyklizität der Graphendarstellung der SWD sollte sich inhaltlich dadurch ergeben, dass

1. ein Oberbegriff nicht Oberbegriff von sich selbst sein kann,
2. dass ein Oberbegriff nicht Unterbegriff eines seiner (direkten) Unterbegriffe sein kann, da die Unterbegriffs-/Oberbegriffsrelation eine semantische Verengung impliziert, und
3. dass ein Oberbegriff aus demselben Grund nicht in der Menge der Unterbegriffe seiner Unterbegriffe (usw.) einen Begriff haben sollte, der von ihm selbst Oberbegriff ist.

Der Cycle Removal-Schritt dient der temporären Entfernung von Zyklen aus dem Graphen. Sein Nutzen beschränkt sich ausschließlich darauf, die Terminierung der nachfolgenden Algorithmen zu garantieren: Die entfernten Zyklen werden am Ende des Sugiyama-Algorithmus üblicherweise wieder eingefügt. Angesichts der obigen Überlegungen, aufgrund derer Zyklen in der SWD eigentlich ausgeschlossen sein sollten, wurde zunächst *kein* Schritt dieser Art implementiert, da ein solcher rechenzeitintensiv sein würde. Testläufe haben jedoch gezeigt, dass die SWD aufgrund von Fehlern keinesfalls frei von Zyklen der oben beschriebenen ersten und zweiten Art ist. Aus diesem Grund wurden einfache Tests implementiert, die solche Zyklen erkennen, eliminieren und den Benutzer warnen. Mehrere Zyklen wurden an die Zentralredaktion gemeldet.

5.3.3 Layer Assignment

Um die Verteilung der grafischen Elemente auf Ebenen – also Zeilen – zu berechnen, wurde der klassische Coffman-Graham-Algorithmus implementiert[43]. Dieser Algorithmus garantiert, dass für eine Ebene `n` alle Kindelemente von Elementen von `n` in den Ebenen `n+1` oder tiefer lokalisiert sind. Umgangssprachlich ausgedrückt weisen also in der grafischen Darstellung alle Pfeile „nach unten“.

Weiterhin erlaubt der Algorithmus die Angabe einer Darstellungsbreite, deren Überschreiten zu einem Zeilenumbruch führt. Durch eine Approximation der Breite der grafischen Elemente, die

[43] vgl. (Coffman und Graham, 1972), (Di Basstista u. a., 1999) S. 284ff.

die Schlagwörter repräsentieren, kann auf diese Weise komfortabel eine fast optimale Ausnutzung der Bildschirmbreite gewährleistet werden[44].

5.3.4 Crossing Reduction

Nach der Ermittlung der horizontalen Ebenen, in denen jedes grafische Element dargestellt werden soll, sollte zur Verbesserung der Übersichtlichkeit die Anzahl der Überschneidungen der die Relationen anzeigenden Pfeile minimiert werden. Ein Beispiel für einen Graphen mit einer Überschneidung ist Abbildung 14, derselbe Graph ohne die Überschneidung findet sich in 15.

Da eine vollständige, optimale Crossing Reduction als NP-vollständiger Algorithmus zu lange dauern würde, wird die Crossing Reduction durch die iterative Anwendung der Median Crossing Reduction[45] durchgeführt.

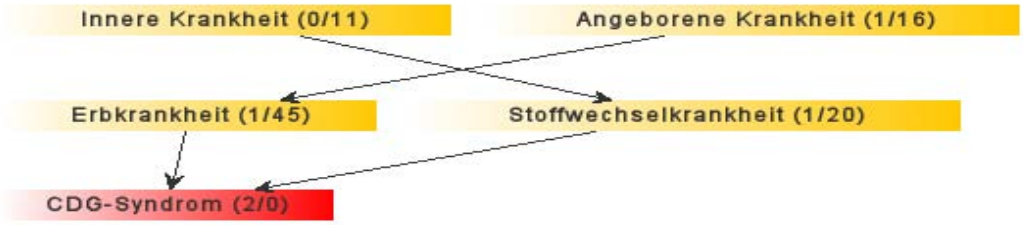

Abbildung 14: Beispiel für die Anzeige eines Graphen mit Überschneidung

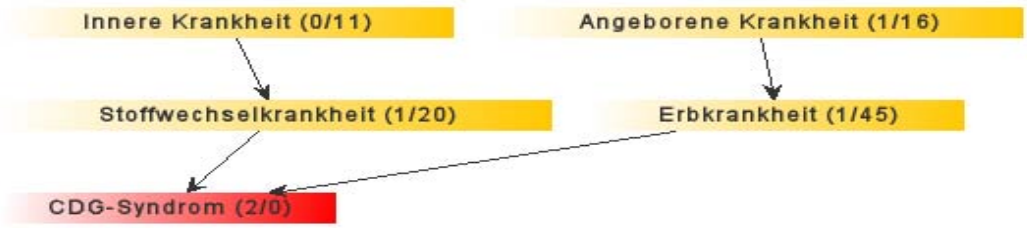

Abbildung 15: Beispiel für die Anzeige eines Graphen ohne Überschneidungen

5.3.5 Horizontal Coordinate Assignment

Die grafischen Elemente, die jeweils für einen Schlagwortdatensatz stehen, werden in einem letzten Schritt einfach mit einer festen Distanz hintereinander angezeigt. Die Breite der Elemente wird durch Addition eines festen Offsets für die Ränder mit dem Produkt der durchschnittlichen Buchstabenbreite mit der Buchstabenanzahl ermittelt. Somit ist dieser Schritt, in dem die Position der Schlagwörter repräsentierenden Symbole in einer Reihe festgelegt wird, trivial (bzw. Resultat des Crossing Reduction-Prozesses).

[44] Aufgrund von Schwierigkeiten der Bestimmung der zur Verfügung stehenden Fläche ist der SWD-Explorer per default für eine Bildschirmauflösung von 1280x1024 Pixeln optimiert. Zukünftige Versionen sollten dies dynamisch anpassen, durch die Verwendung des Coffman-Graham-Algorithmus ist die Grundlage dafür geschaffen.

[45] vgl. (Di Basstista u. a., 1999) S. 284ff

6 Rezeption und Ausblick

> *[Es] [...] interessieren [...] ja nicht die individuellen Beiträge der daran beteiligten Einzelnen, sondern vielmehr die erst durch (mehr oder weniger) gleichartigen Gebrauch sprachlicher Terme von Vielen [...] sichtbar werdenden* Verwendungsregularitäten [...]
> *B. Rieger*[46]

Der SWD-Explorer wurde an der TIB/UB Hannover in den Fachreferaten Mathematik, Informatik, Maschinenbau, Wirtschaftswissenschaften und Wirtschaftsinformatik erfolgreich getestet. Er steht hausintern zur Verfügung und wird ggf. von weiteren Referent/innen genutzt.

Im Kontext der Entwicklung des Programms wurden 180 Dubletten und 71 unsauber formulierte Schlagwortdatensätze automatisch identifiziert und nach Sichtung an die Göttinger Zentralredaktion gemeldet. Weiterhin konnten im Kontext der Arbeit der o.g. Fachreferate diverse strukturelle Fehler in der SWD, die auf inhaltliche Fehler zurückgingen, identifiziert und gemeldet werden.

6.1 Editierung von Thesauri und hierarchisch strukturierten, kontrollierten Vokabularen

Es ist naheliegend, den vorliegenden, rein zur *Darstellung* der SWD geeigneten SWD-Explorer dahingehend zu erweitern, dass ein Werkzeug zur *Editierung* geschaffen wird. Aufgrund des Umstandes, dass Änderungswünsche in der SWD überlicherweise nur im proprietärem PICA-System durchführbar sind, wurde auf eine solche Modifikation bisher verzichtet. Für eine zukünftige, mögliche Offenlegung der Schnittstellen oder für die Anwendung im Kontext eines anderen kontrollierten Vokabulars ist eine solche Erweiterung jedoch denkbar. Die Einführung eines Mechanismus, der die Verknüpfung eines neu angesetzten Schlagwortes mit per Maus anzuzeigenden Ober- und Unterbegriffen semiautomatisch durchführt bzw. die Verfügbarmachung eines Fensters, in dem die Eigenschaften eines Schlagwortdatensatzes per Tastatur editierbar wären, würden die beiden nächsten Schritte zu diesem Ziel darstellen.

6.2 Darstellung alternativer Formate. Beispiel: SKOS

Aufgrund des modularen Aufbaus des SWD-Explorers lassen sich vergleichbar gestaltete kontrollierte Vokabulare mit geringem Aufwand ebenfalls visualisieren. Einen besonders interessanter Fall ist dabei die Visualisierung von Vokabularen, die in dem *Simple Knowledge Organization System* (SKOS) repräsentiert sind.

Bei SKOS handelt es sich um eine von W3C definierte, formale Sprache auf der Basis von XML[47] bzw. RDF[48]. Das Ziel von SKOS ist die Repräsentation sowohl von der **Struktur** als auch

[46] (Rieger, 1989, S.196), Löschungen und Einfügungen in Klammern von mir.
[47] vgl. http://www.w3.org/TR/2008/REC-xml-20081126/
[48] vgl. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

von dem **Inhalt** einfacher Wissensorganisationssysteme (KOS). Um dies zu erreichen, definiert SKOS Entitäten sowie häufig bzw. üblich vorkommende Relationen und Eigenschaften und liefert standardisierte Methoden, diese zu repräsentieren.

Der Grund, von dem her die verhältnismäßig einfach umzusetzende Darstellung von SKOS-basierten KOS ein sehr lohnendes Ziel ist, liegt in der geplanten Verwendung von SKOS als *lingua franca* der kontrollierten Vokabulare: Durch Weglassung spezifischer Besonderheiten können kontrollierte Vokabulare (Thesauri, Ontologien...) einfach in SKOS übersetzt werden. Da SKOS komplett in RDF/XML definiert ist, ist es maschinenlesbar und maschinell interpretierbar. Auf diese Weise können z.B. Suchmaschinen auf die SKOS-Repräsentationen kontrollierter Vokabulare in Datenbanken zugreifen, und diese – ohne sich mit der Interpretation des individuellen Datenformats des KOS der Datenbank beschäftigen zu müssen – durchsuchbar machen. Wie ich im Folgenden zeigen werde, ist sogar die SWD trivial in das SKOS-Format übersetzbar.

Im Folgenden möchte ich auf die Struktur von SKOS eingehen und die Integration in den SWD-Explorer bzw. die Parallelität zur SWD skizzieren. Aufgrund der technischen Komplexität von RDF/S und SKOS muss ich dabei leider auf die detaillierte Skizzierung der Syntax verzichten bzw. setze elementare Kenntnisse in RDF voraus.

6.2.1 SKOS – Konzepte und Relationen

SKOS definiert als grundlegende Entität das **Konzept**. Konzepte werden über einen URI eindeutig identifiziert, die die Klasse `skos:Concept`[49] zugewiesen bekommt. Das SWD-Pendant zu einem Konzept ist ein Schlagwort (nicht seine Ansetzungsformen). In den Datenstrukturen des SWD-Explorers wäre die Entsprechung die Klasse `SWDdata.Entry`[50].

6.2.1.1 Labels SKOS differenziert zwischen drei sog. „Labels“: `skos:prefLabel`, `skos:altLabel` und `skos:hiddenLabel`. Label sind RDF-Eigenschaften des Typs `rdfs:label` und sind somit durch String-Datentypen repräsentiert. Sie stellen sie Namen des Konzeptes dar; das Pendant in der SWD sind die (Haupt-)Ansetzungsformen.

Obgleich `skos:prefLabel` mehrfach für ein Konzept definiert werden kann, wird geraten, davon abzusehen. Bei diesem Label handelt es sich um den präferierten Namen, der direkte Bezug zu der Hauptansetzungsform von Schlagworten der SWD ist klar erkennbar. `skos:altLabel` definiert alternative Namen, von denen beliebig viele vorhanden sein können und die z.B. zur vereinfachten Suche des Konzeptes dienen. Die SWD-Entsprechung sind die Nebenansetzungsfomen, die einen identischen Zweck erfüllen. Die Problematik bei der Darstellung von SKOS-definierten KOS im SWD-Explorer bestünde an dieser Stelle nur darin, dass es in SKOS nicht verboten ist, kein oder mehrere `skos:prefLabel` zu definieren. In diesem Fall könnte man aber trivialerweise ein künstlich erzeugtes oder aus den `skos:altLabel` gewonnenes Label verwenden bzw. einfach das erste `skos:altLabel` als repräsentierten Namen im SWD-Explorer anzeigen.

[49] Der XML-Namensraum skos entspricht dem Präfix `http://www.w3.org/2008/05/skos#`
[50] Für alle Klassen des SWD-Explorers: Vergleiche den Anhang

`skos:hiddenLabel` haben in der SWD keine Entsprechung, es handelt sich um „versteckte“ Namen, die nicht angezeigt werden sollen, aber zur Suche bzw. Indexierung zu Verfügung stehen sollen. Eine Gleichbehandlung mit den Nebenansetzungsformen der SWD im SWD-Explorer wäre eine mögliche Vorgehensweise.

6.2.1.2 Semantische Relationen In SKOS sind verschiedene semantische Relationen zwischen Konzepten definiert. Unterschieden wird dabei zwischen `skos:broader`, `skos:narrower` und `skos:related`. `skos:broader` soll verwendet werden, wenn das eine Konzept eine semantisch weitere Bedeutung hat (also z.B. ein Oberbegriff ist). `skos:narrower` wird dann verwendet, wenn das eine Konzept eine semantisch engere Bedeutung hat (also z.B. ein Unterbegriff ist). Die beiden Relationen implizieren sich invers, so dass man alle Relationen dieser Art durch eine der beiden Relationen darstellen kann. Ähnlich wurde in der PICA 3-Repräsentation der SWD verfahren, in der alle Unterbegriffs/Oberbegriff-Relationen durch die Kennzeichnung des Oberbegriffs im Datensatz geschehen.

Die Relationen sind nicht zwangsläufig transitiv, was für eine Darstellung im SWD-Explorer – die eine transitive Beziehung indirekt impliziert – ggf. problematisch ist. Auf der anderen Seite werden auch im SWD-Explorer nichttransitive pars-pro-toto-Relationen bzw. die nichttransitiven Relationen zwischen Unterbegriffen, mehrgliedrigen Oberbegriffen und deren Elementen der SWD visualisiert. Man muss sich bei der Darstellung eines SKOSifizierten Vokabulars nur über diesen Umstand im Klaren sein.

Der Unterschied zu der Definition von Unter- bzw. Oberbegriffen in der SWD liegt genau in diesem Punkt: Die pars-pro-toto-Relationen der SWD entsprechen nicht der strengen Definition eines semantisch engeren bzw. weiteren Begriffes. Auf der anderen Seite stellen diese auch mit Abstand die geringere Menge von Relationen in der SWD dar. Im SWD-Explorer werden diese Relationen durch die Klasse `SWDdata:Relation` repräsentiert.

Semantische Ähnlichkeit wird im SKOS in der Relation `skos:related` repräsentiert. In der SWD entspricht dieses der Ähnlichkeitsbeziehung zwischen Schlagworten. Wie oben geschildert, werden Ähnlichkeitsbeziehungen im SWD-Explorer z.Zt. nicht grafisch visualisiert.

Wie erkennbar ist, sollte sowohl eine (einschränkende) Repräsentation der SWD in SKOS, aber auch eine Repräsentation von SKOS-Datensätzen im SWD-Explorer trivial umsetzbar sein. Das einzige, was für letzteres benötigt würde, wäre die Implementation eines SKOS-XML-Parsers in den SWD-Explorer, der die jeweiligen Konzepte, Eigenschaften und Relationen in `SWDdata.Entry`- bzw. `SWDdata.Relation`-Objekte umsetzt.

7 Zusammenfassung

Im Rahmen dieser Arbeit habe ich ein praxisorientiertes Softwaretool zur Visualisierung der Schlagwortnormdatei erstellt und dargestellt. Der Nutzen einer solchen Software besteht in der starken Erleichterung der Prozesse der Verschlagwortung und der Datenpflege. So lassen sich strukturelle und oft auch inhaltliche Fehler in der Schlagwortnormdatei mit Hilfe des Tools auf einen Blick entdecken, durch die eingebauten Funktionen können Beispieldatensätze als Grundlage für die Ansetzung neuer Schlagworte einfach kopiert werden.

Ein besonderer Nutzen ergibt sich aus der Implementierung der Suche von Schlagworten mit Hilfe Regulärer Ausdrücke, die eine viel effizientere und genauere Recherche als die reine Suche mit Hilfe exakt übereinstimmender oder rechtstrunkierter Suchbegriffe ermöglichen.

Die entwickelte Software wurde bereits in der Praxis – in Fachreferaten der TIB/UB Hannover – eingesetzt und sehr positiv aufgenommen. Aufgrund der gewählten Strategie, das Tool vollständig auf Open-Source-Software basieren zu lassen, läßt es sich im Rahmen anderer Bibliotheken ohne zusätzliche Kosten zum Einsatz bringen.

Das Tool ist aufgrund der modularen Struktur erweiterbar und kann durch einfache Erweiterungen z.B. zur Repräsentation von SKOS-repräsentierten KOS verwendet werden. Selbst wenn das Tool aufgrund des Umstandes, dass es nicht Teil eines größeren Softwarepaketes oder eines Bibliotheksverwaltungssystems ist, auf Dauer in dieser Form nicht weiterentwickelt werden sollte, kann es als Machbarkeitsstudie gesehen werden, an der sich die Entwickler der verbreiteten Tools und Systeme orientieren können.

Die im Rahmen dieser Arbeit geschaffene Softwarebasis stellt ein sofort einsatzbereites Werkzeug dar, aber ebenso eine optimale Grundlage, um weitere Anwendungen darauf basierend zu entwickeln. Aus diesem Grund möchte ich die Arbeit mit einer Bemerkung beenden, bei der ich augenzwinkernd auf die eingangs zitierte Äußerung der Chesshire-Katze aus Carrols „Alice“ verweisen möchte: Der nächste Schritt ist letztendlich nur davon abhängig, wohin man gehen möchte.

Abbildungsverzeichnis

Literatur

[AACR 2002] *Anglo-american cataloguing rules.* 2002. – Second Edition, 2002 Revision: 2005 Update. Joint Steering Committee for Revision of AACR

[Bisig 1994] Bisig, Urs: OPAC und verbale Sacherschließung : ein Beitrag zur RSWK-Diskussion. In: *ABI-Technik* 14 (1994), S. 117–130

[Capellaro 2003] Capellaro, Christof: *Die Schlagwortnormdatei – ein zentrales Hilfsmittel der verbalen Sacherschließung : Versuch einer Einführung.* Berlin, Humboldt-Universität zu Berlin, Institut für Bibliothekswissenschaft, Diplomarbeit, 2003. – URL: http://www.ib.hu-berlin.de/texte/hausarbeiten/capellaro/swd-capellaro.htm, Version vom 4.11.2003

[Carroll 1982] Carroll, Lewis: *The Complete Illustrated Works.* New York : Gramercy Books, Random House Inc., 1982

[Coffman und Graham 1972] Coffman, E.G. ; Graham, R.L.: Optimal scheduling for two processor systems. In: *Acta Informatica* 1 (1972), S. 200–213

[Di Basstista u. a. 1999] Di Basstista, Giuseppe ; Eades, Peter ; Tamassia, Roberto ; Tollis, Ioannis G.: *Graph Drawing - Algorithms for the Visualization of Graphs.* New Jersey : Prentice-Hall Inc., 1999

[Hacker 2000] Hacker, Rupert: *Bibliothekarisches Grundwissen.* 7. Auflage. München : K. G. Saur, 2000

[Kunz u. a. 2007] Kunz, Martin (Hrsg.) ; Scheven, Esther (Hrsg.) ; Bellgardt, Sigrid (Hrsg.): *Regeln für den Schlagwortkatalog RSWK.* 3., überarbeitete und erweiterte Auflage auf dem Stand der 4. Ergänzungslieferung. Februar 2007. – Erarbeitet von der Expertengruppe RSWK des Deutschen Bibliotheksinstituts. URN: urn:nbn:de:1111-20040721235

[Mitchell 2003] Mitchell, Joan S. (Hrsg.): *Dewey decimal classification and relative index / devised by Melvil Dewey.* Dublin, Ohio : OCLC Online Computer Library Center, 2003 (22)

[Popst 1993] Popst, Hans (Hrsg.): *Regeln für die alphabetische Katalogisierung in wissenschaftlichen Bibliotheken : RAK-WB.* 2., überarb. Ausg. Berlin : Deutsches Bibliotheksinstitut, 1993. – ISBN 3-87068-436-4

[Recker-Kotulla 1992] Recker-Kotulla, Ingrid (Hrsg.): *Basisklassifikation für den Bibliotheksverbund Niedersachsen, Sachsen-Anhalt / PICA-Projekt Niedersachsen, Projektgruppe F: Sacherschließung.* Hannover : Nieders. Ministerium für Wissenschaft und Kultur [u.a.], November 1992

[Rieger 1989] Rieger, Burghard B.: *Unscharfe Semantik: die empirische Analyse, quantitative Beschreibung, formale Repräsentation und prozedurale Modellierung vager Wortbedeutungen in Texten.* Frankfurt am Main : Verlag Peter Lang, 1989

[Sugiyama u. a. 1981] Sugiyama, K. ; Tagawa, S. ; Toda, M.: Methods for visual understanding of hierarchical system structures. In: *IEEE Trans. on Systems, Man and Cybernetics* SMC-11 (1981), Nr. 2, S. 109–125

[Sugiyama 2002] Sugiyama, Kozo: *Series on Software Engineering and Knowledge Engineering*. Bd. 11: *Graph Drawing and Applications for Software and Knowledge Engineers*. Singapur u.a. : World Scientific, 2002

[Werner 1990] Werner, Stephan (Hrsg.) ; Kolloquium zur Schlagwortnormdatei (Veranst.): *Die Schlagwortnormdatei, Entwicklungsstand und Nutzungsmöglichkeiten : Vorträge eines Kolloqiums zur Schlagwortnormdatei (SWD) in Frankfurt a.M. am 5. und 6. Oktober 1989*. Berlin : Deutsches Bibliotheksinstitut, 1990

Index

A SWD-Explorer: Sourcecode

A.1 Hauptklasse

Die Hauptklasse `SWDexplorer`. Sie enthält die zum Starten einer Java-Applikation notwendige Main-Methode und initialisiert alle abhängigen Klassen und Methoden.

```
import SWDdata.Database;
import SWDgui.ExplorerGui;
import SWDio.SystemLogger;

class SWDexplorer
{
    public SWDexplorer(){};

    public static void main (String[] args)
    {
        SystemLogger.setOutfile("SWDlog.log"); //( Angabe absoluter Pfade
            moeglich))
        Database content = new Database();
        ExplorerGui gui = new ExplorerGui(content);
        gui.createGUI();
    }
}
```

A.2 Paket SWDdata – Datenverwaltung

Die Klassen dieses Paketes dienen der Datenverwaltung im Speicher.

A.2.1 Klasse SWDdata.Database

Die Klasse `SWDdata.Database`. Diese Klasse stellt Methoden zur Verfügung, die den gekapselten Zugriff auf die Schlagwortnormdaten erlauben.

```
package SWDdata;

import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Iterator;
import java.util.TreeSet;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.concurrent.locks.ReentrantLock;

import SWDio.SystemLogger;

//import SWDfilereader;

public class Database
{
    // Beschreibung
    // Verwaltungseinheit aller Eintraege

    // Notizen:
    /* Was tun, wenn auf einen Entry referenziert wird, der noch nicht
       existiert? Am Ende einen Cleanup machen, oder ein dynamisches
       Konstruieren zulassen?
       Problem Fall 1: Ggf. gibt es immer unverbundene Eintraege
       Problem Fall 2: Fake-Eintraege waeren die beste Alternative
       geplant: Die Algorithmen liefern einen Dummy zurueck, wenn ein
       nicht existierender Eintrag geliefert werden soll
     */

    // Datenfelder
    private HashMap<String,String> h_nameToName = new HashMap<String,
        String>();
    // Liefert fuer ALLE Bezeichner die PPN:
    private HashMap<String,Entry> h_nameToEntry = new HashMap<String,Entry
        >();
    private LinkedList<Entry> l_entrylist = new LinkedList<Entry>(); //
        Liste aller Entrys
    // Die jeweiligen Kinder:
    private HashMap<String,LinkedList<String>> h_nameToChildren = new
        HashMap<String,LinkedList<String>>();
    // Die jeweiligen Eltern
    private HashMap<String,LinkedList<String>> h_nameToParents = new
        HashMap<String,LinkedList<String>>();
```

```
// Variablen
final Pattern pattern = Pattern.compile("^\\d\\d\\d$");
ReentrantLock lock = new ReentrantLock();

// Konstruktoren
public Database () {};

public Database (File infile)
{
    l_entrylist = readFile(infile);
    updateHashes();
    String number = Integer.toString(l_entrylist.size());
    SystemLogger.info("Datei eingelesen. " + number + " Datensaetze");
}

// Methoden ##########################################################

// Externes Dazuladen von Dateien
public void read(File infile)
{
    SystemLogger.info("Lese Datei " + infile.getName() + ". Bitte
        warten.");
    // Aufruf der Einlesemethode
    LinkedList <Entry> entrylist = new LinkedList<Entry>();
    entrylist = readFile(infile);

    lock.lock();

    // Zusammenfuegen der globalen und der neuen Liste
    Iterator <Entry>iter = entrylist.iterator();
    while (iter.hasNext())
    {
        l_entrylist.add( (Entry) iter.next());
    }

    lock.unlock();

    updateHashes();

    String number = Integer.toString(l_entrylist.size());
    SystemLogger.info("Datei eingelesen. " + number + " Datensaetze");
    System.gc();
}

// Erzeugt eine LinkedList aller Eintraege aus einer Datei. Die
    Eintraege muessen
// in sich stimmig sein, aber nicht die Zusammenhaenge zwischen den
    Eintraegen
private LinkedList <Entry> readFile(File infile)
{
    lock.lock();

    LinkedList <Entry> l_elist = new LinkedList<Entry>(); // Liste
        aller Entrys

    try
```

```
{
    LinkedList <String>  l_linelist  = new LinkedList<String>();
    Entry entry = null;
    boolean c = true;
    boolean first = true;

    BufferedReader reader = new BufferedReader(new FileReader(
        infile));

    while (c)
    {
        String s_line = reader.readLine();
        if (s_line == null) { c = false; }
        else
        {
            String s_test = "";
            s_line = s_line.trim();
            if (! s_line.isEmpty())
            {
                String [] x = s_line.split("\\s",2);
                s_test = x[0];

                Matcher m = pattern.matcher(s_test);

                if ( m.matches() || s_test.equals("SET:"))
                {
                    if (s_test.equals("SET:"))
                    {
                        // Ist es der erste Eintrag?
                        if (!first)
                        {
                            entry = new Entry(l_linelist);
                            l_linelist = new LinkedList<String>();
                            l_linelist.add(s_line);
                            l_elist.add(entry);
                        }
                        else
                        {
                            first = false;
                            l_linelist.add(s_line);
                        }
                    }
                    else
                    {
                        l_linelist.add(s_line);
                    }
                }
            }
        }
    }
    if (l_linelist.size() > 0)
    {
        entry = new Entry(l_linelist);
        l_elist.add(entry);
    }
    reader.close();
}
```

```
        catch (IOException e)
        {
            System.err.println("Datei␣"+ infile.getName()  +"␣kann␣nicht␣
                gelesen␣werden!");
            System.exit(0);
        }

        lock.unlock();

        return(l_elist);
    }

    /* Methode zum Eintragen der "virtuellen" SWEntries, die auf
     * zusammengesetzten Oberbegriffen basieren
     * Wichtig dabei: Ggf. aeltere Eintraege loeschen
     */
    private void processCombinedEntries()
    {
        // aeltere Eintraege mit zusammengesetzten Begriffen loeschen
        {
            LinkedList <Entry> l_temp = new LinkedList <Entry>();

            for (Entry entry : l_entrylist)
            {
                if (!entry.getCombined())
                {
                    l_temp.add(entry);
                }
            }
            l_entrylist = l_temp;
        }

        HashMap<String,LinkedList<String>> keyToNames = new HashMap<String,
            LinkedList<String>>();

        /* Es wird ein auf den einzelnen Oberbegriffen basierender
            Schluessel erstellt,
         * anhand dessen bstimmt wird, ob zwei zusammengesetzte
            Oberbegriffe gleich sind.
         * Der Hauptname des zusammengesetzten Oberbegriffs ist jedoch der
         * erste Name, der ueberhaupt gefunden wird.
         */
        for (Entry entry : l_entrylist)
        {
            LinkedList<String> parents = entry.getParents();
            for (String parent : parents)
            {
                if (parent.contains("/"))
                {
                    String key = "";
                    TreeSet<String> sorter = new TreeSet<String>();
                    String[] s = parent.split("/");
                    for (String name : s)
                    {
                        name = name.trim();
                        sorter.add(name);
```

```
                    }
                    for (String name : sorter)
                    {
                        key = key.concat("_!_" + name);
                    }
                    if (!(keyToNames.containsKey(key)))
                    {
                        keyToNames.put(key,new LinkedList<String>());
                    }
                    (keyToNames.get(key)).add(parent);
                }
            }
        }
        // Eintragen der neuen Begriffe in die Datenbank
        // Aufpassen: keine Dubletten bei den Namen
        for (String key : keyToNames.keySet())
        {
            LinkedList<String> names = new LinkedList<String>();
            HashMap <String,Boolean> red = new HashMap<String,Boolean>();
            for (String name : keyToNames.get(key))
            {
                if (!(red.containsKey(name)))
                {
                    names.add(name);
                    red.put(name,false);
                }
            }
            Entry entry = new Entry(names,true);
            l_entrylist.add(entry);
        }
    }

    private void updateHashes()
    {
        // Zusammengesetzte Begriffe verarbeiten
        processCombinedEntries();

        /* Dublettencheck!
           Wenn ein Element desselben Namens gefunden wird, wird das
               aeltere Element
           ueberschrieben */

        LinkedList<Entry> t = new LinkedList<Entry>();
        HashMap<String,String> names = new HashMap<String,String>();

        Iterator<Entry> iter = l_entrylist.descendingIterator();
        while (iter.hasNext())
        {
            Entry entry = iter.next();

            String name = entry.getName();
            if (names.containsKey(name))
            {
                String ppnAlt = names.get(name);
                String ppnNeu = entry.getPPN();
                SystemLogger.warn("Dublette: " + name + ". PPN-alt: " +
                    ppnAlt +
```

```
                    "␣PPN-neu:␣" + ppnNeu +"␣Das␣aeltere␣Element␣wird␣
                        ueberschrieben.");
        }
        else
        {
            names.put(name,entry.getPPN());
            t.add(entry);
        }

        // Pruefung auf interne Doppeleintraege
        HashMap <String,String> u = new HashMap<String,String>();
        for (String s_name : entry.getAllNames())
        {
            if (u.containsKey (s_name))
            {
                SystemLogger.warn("Der␣Eintrag:␣" + s_name +
                        "␣spezifiziert␣denselben␣Namen␣mehrfach!");
            }
            u.put(s_name, "");
        }
    }

    l_entrylist = new LinkedList<Entry>();
    iter = t.descendingIterator();
    while (iter.hasNext())
    {
        l_entrylist.add(iter.next());
    }

    /* Aufbau der Hashes zur vereinfachten Suche
     *
     * private HashMap <String,LinkedList<SWEntry>> h_nameToChildren =
     * new HashMap <String,LinkedList<SWEntry>>(); // Die jeweiligen
         Kinder
       private HashMap <String,LinkedList<SWEntry>> h_nameToParents*/

    h_nameToEntry    = new HashMap <String,Entry>();
    h_nameToName     = new HashMap <String,String>();
    h_nameToChildren = new HashMap <String,LinkedList<String>>();
    h_nameToParents  = new HashMap <String,LinkedList<String>>();

    // h_nameToName, h_nameToEntry h_nameToParents
    for (Entry entry : l_entrylist)
    {
        LinkedList<String> content = entry.getAllNames();

        for (String name : content)
        {
            h_nameToName.put(name, entry.getName());
            h_nameToEntry.put(name, entry);
        }
    }

    /* Bitte beachten: h_nameToParents speichert nur die Hauptnamen
     * als Schluessel, aber ggf. die Spezialnamen als Value
     * Die Spezialnamen muessen aber auf jeden Fall von Dubletten
         bereinigt werden,
```

```
     * die durch zusammengesetzte Oberbegriffe entstehen.
     */
    for (Entry entry : l_entrylist)
    {
        LinkedList<String> parents = new LinkedList<String>();
        HashMap<String,Boolean> re = new HashMap<String,Boolean>();
        for (String parent : entry.getParents())
        {
            String f = h_nameToName.get(parent);
            if ( (!(re.containsKey(f))) && (f != null))
            {
                parents.add(f);
                re.put(f, false);
            }
        }
        h_nameToParents.put(entry.getName(),parents);
    }

    /* h_nameToChildren
     * Bitte beachten: Der Schluessel ist immer der Hauptname,
     * ebenso wie das Kind
     */
    for (Entry entry : l_entrylist)
    {
        String name = entry.getName();

        LinkedList <String> l_parents = entry.getParents();
        Iterator <String> parents = l_parents.iterator();
        while (parents.hasNext())
        {
            String parent = h_nameToName.get(parents.next());

            LinkedList <String> temp = new LinkedList<String>();
            if (h_nameToChildren.containsKey(parent))
            {
                temp = h_nameToChildren.get(parent);
            }
            temp.add(name);
            h_nameToChildren.put(parent, temp);
        }
    }

    /* Jetzt noch Dubletten aus h_nameToChildren entfernen, die durch
     * die zusammengesetzten Oberbegriffe ggf. entstanden sind
     */
    for (String name : h_nameToChildren.keySet())
    {
        LinkedList<String> rem = new LinkedList<String>();
        LinkedList<String> children = h_nameToChildren.get(name);
        HashMap <String,Boolean> u = new HashMap<String,Boolean>();
        for (String child : children)
        {
            u.put(child, false);
        }

        for (String child : u.keySet())
        {
```

```
                rem.add(child);
            }
            h_nameToChildren.put(name, rem);
        }
    }

    // Ausgabe aller Namen von Entries auf STDOUT
    public void printNames()
    {
        lock.lock();

        Iterator<Entry> iter = l_entrylist.iterator();

        while (iter.hasNext())
        {
            Entry one = (Entry) iter.next();
            String name = one.getName();
            System.out.println(name);

            Iterator<String> iterr = (one.getParents()).iterator();
            while (iterr.hasNext())
            {
                System.out.println("     " + (String) iterr.next());
            }
        }

        lock.unlock();
    }

    // Ausgabe aller (Haupt-)Namen in einem Stringarray
    public String[] getNames()
    {
        String[] result = new String[l_entrylist.size()];
        Iterator<Entry> iter = l_entrylist.iterator();

        int i = 0;
        while (iter.hasNext())
        {
            Entry x = iter.next();
            String name = x.getName();
            result[i] = name;
            i++;
        }
        return(result);
    }

    // Ausgabe aller Namen und Namensvarianten in einer LinkedList<String>
    public LinkedList<String> getAllNames()
    {
        LinkedList<String> result = new LinkedList<String>();

        for (Entry entry : l_entrylist)
        {
            /* Fuer die Generierung der Daten fuer den Machter
             * hier einkommentieren und die Entry-Kategorie anpassen

            String category = entry.getCategory();
```

```
            String mainForm = entry.getName();
            */

            LinkedList<String> names = entry.getAllNames();
            for (String name : names)
            {
                result.add(name);
                //result.add(name + ":::" + mainForm + "+++" + category);
            }
        }
        return(result);
    }

    // Ausgabe aller SWEntries in einem Array
    public Entry[] getEntries()
    {
        Entry[] result = new Entry[l_entrylist.size()];
        Iterator <Entry> iter = l_entrylist.iterator();

        int i = 0;
        while (iter.hasNext())
        {
            Entry x = iter.next();
            result[i] = x;
            i++;
        }
        return(result);
    }

    public LinkedList<String> getChildrenNames(String name)
    {
        LinkedList<String> result = new LinkedList<String>();

        name = getNameByName(name);

        if (h_nameToChildren.get(name) != null)
        {
            for (String s : h_nameToChildren.get(name))
            {
                result.add(s);
            }
        }
        return(result);
    }

    public boolean getCombined(String name)
    {
        Entry entry = getEntryByName(name);
        if (entry != null)
        {
            return(entry.getCombined());
        }
        return false;
    }

    public LinkedList<String> getParentNames(String name)
    {
```

```
        LinkedList<String> result = new LinkedList<String>();

        name = getNameByName(name);

        if (h_nameToParents.get(name) != null)
        {
            for (String s : h_nameToParents.get(name))
            {
                result.add(s);
            }
        }

        return(result);
    }

    public int getParentCount(String name)
    {
        name = getNameByName(name);
        if (h_nameToParents.get(name) != null)
        {
            return(h_nameToParents.get(name).size());
        }
        return(0);
    }


    public int getChildrenCount(String name)
    {
        name = getNameByName(name);
        if (h_nameToChildren.get(name) != null)
        {
            return(h_nameToChildren.get(name).size());
        }
        return(0);
    }

    // Ausgabe eines Eintrags anhand des Bezeichners (Namens)
    // null, wenn nicht vorhanden
    public Entry getEntryByName(String s)
    {
        s = s.replaceFirst("\\(.*\\)", "");
        Entry result = null;
        if (h_nameToEntry.containsKey(s))
        {
            result = h_nameToEntry.get(s);
        }
        return(result);
    }

    public String getPPNByName(String s)
    {
        Entry entry = getEntryByName(s);
        if (entry == null)
        {
            return(null);
        }
        return(entry.getPPN());
```

```
    }

    // Liefert den Hauptnamen eines Eintrags anhand eines Nebennamens
    public String getNameByName (String s)
    {
        s = s.replaceFirst("\\(.*/.*\\)", "");
        String r = "";
        if (h_nameToName.containsKey(s))
        {
            r = h_nameToName.get(s);
        }
        return(r);
    }

    // Liefert die Entitaetsbeschreibung eines Objektes
    // Benoetigt fuer das Detailausgabefenster

    public String getDescription(String s)
    {
        if (s == null)
        {
            return("");
        }
        Entry x = getEntryByName(s);

        if (x != null)
        {
            return("PPN: " + x.getPPN() + "\n" + x.getContent());
        }
        return("");
    }

    /* Liefert alle Eintraege, bei denen Kategorie 950 gesetzt wurde */
    public LinkedList<String> getAllEntriesWithChangeCommentary()
    {
        LinkedList<String> result = new LinkedList<String>();
        for ( Entry entry : l_entrylist )
        {
            String name = entry.getChangeCommentary();
            if (name != null)
            {
                result.add(entry.getName());
            }
        }
        return(result);
    }
}
```

A.2.2 Klasse SWDdata.Entry

Die Klasse `SWDdata.Entry`. Eine Instanz dieser Klasse speichert genau einen Schlagwortdatensatz. Die Instanzen werden von der Klasse `SWDdata.Database` verwaltet.

```
package SWDdata;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import SWDio.SystemLogger;

class Entry
{
    // # Beschreibung ######################################################

    // Notizen:
    /*
      005 Datencode. Interessant ist ggf, ob es sich um ein v oder x oder y
          handelt
      800 Bezeichnung, eingeleitet durch Schlagworttyp. Dieser kann ggf.
          extrahiert werden
      801-805 Mehrgliedriges Schlagwort
      Spitze Klammern zur Disambiguierung
      808 Quelle (|a|) und Beschreibung (|b|), mehrere moeglich
      810 Gebiet (Zahl)
      820 Alternative Ansetzungsform des Hauptsachtitels. Bei Personen:
          Nach Duden transkribiert, bei
              Verfasserwerken der Verfasser. Wird der Einfachheit halber
                  genau wie ein Synonym behandelt,
              da in diesem Explorer ja nur Sachschlagworte aus der SWD
                  bearbeitet werden sollen.
      830 Zusammengesetzte Oberbegriffe, CSV, Schlagworttyp am Anfang nur
          fuer das erste . Problem: Permutiert
      850 Oberbegriff
      860 Verwandte Begriffe
      950 Aenderungswunsch/Kommentar
     */

    /* Nicht beachtete Zahlcodes:
     *  008 - Sprachenkennung
        020 - Identifikationsnummer der GKD
        021 - PPN der SWD.
        811 - Laendercode (einmal "Muenchen"?)
        820 - uebersetzung eines Titels
        880 - ??? Kommt sehr selten vor
        900 - Besonderheiten (meist "In SWD geloescht")
        NN -
     */

    // # Datenfelder ######################################################
    // Ein Muster, das nur einmal kompiliert werden soll...
    // Pattern ppnPattern;
```

```
static Pattern ppnPattern = Pattern.compile("PPN:\\s\\d\\d*\\w?");

// Zusammengesetzter Begriff?
boolean b_combined = false;

/* Originaler Content in String-Repraesentation mit Newlines.
 * Wird benoetigt, um im Objektausgabefenster eine Ausgabe zu
    ermoeglichen.
 */
String s_content = "";

/* 005 Datencode. Interessant ist ggf., ob es sich um ein v oder x oder
   y handelt */
// private String s_code;

// 021
private String s_ppn;

/* 800 Bezeichnung, eingeleitet durch Schlagworttyp.
   Dieser kann ggf. extrahiert werden
   830 Weitere Bezeichnungen/Synonyme
   Die Position 0 in den beiden Listen steht fuer den eigentlichen
      Namen
 */
private ArrayList <String> l_namen = new ArrayList<String>();
private ArrayList <String> l_typen = new ArrayList<String>();

/* 808 Quelle (|a|) und Beschreibung (|b|), mehrere moeglich */
//private String s_source;
//private String s_description;

/* 810 Gebiet (Zahlenkette) */
//private String s_category;

/* 811 Herkunftsland */
//private String s_origin;

/* 845 Zusammengesetzte Oberbegriffe, CSV, Schlagworttyp am Anfang nur
   fuer das erste. Problem: Permutiert
   850 Einfache Oberbegriffe werden ebenfalls hier abgelegt*/
private HashMap <String,Boolean> h_oberbegriff = new HashMap<String,
   Boolean>();

/* 860 Verwandte Begriffe */
private LinkedList <String> l_verwandte = new LinkedList <String>();

/* 950 Bitte um aenderung */
private String changeCommentary = null;

/* Testvariablen */
private boolean b_code = false;
private boolean b_ppn = false;
private boolean b_name = false;

// # Konstruktor #########################################################
/* Der Entry kann per Hand spezifiziert werden, was in Faellen von
 * zusammengesetzten, also nicht in der Datenbank vorhandenen
```

```
     * Eintraegen notwendig ist
     */
    Entry (LinkedList <String> namen, boolean zusammengesetzt)
    {
        s_content = "Kein Datensatz vorhanden. Zusammengesetzter Begriff.";
        b_combined = true;

        for (String name : namen)
        {
            s_content = s_content.concat("\n" + "(Alternativer) Name: " +
                name);
        }

        HashMap<String,Boolean> parents = new HashMap<String,Boolean>();
        for (String name : namen)
        {
            name = name.trim();
            l_namen.add(name);
            String[] result = name.split("/");
            for (String s : result)
            {
                s = s.trim();
                parents.put(s,false);
            }
        };

        for (String parent : parents.keySet())
        {
            h_oberbegriff.put(parent, false);
            s_content = s_content.concat("\n" + "Virtueller Oberbegriff: "
                + parent);
        }

        s_ppn = "000000000";
    }

    /* Parameter des Konstruktors ist eine LinkedList aus Strings, die
       die einzelnen Zeilen der Datenbank enthalten.
       Dabei gilt: Jede Zeile muss von einer dreistelligen Zahl eingeleitet
       werden, die von dem Eintrag durch ein Space getrennt ist.
       Ausnahme ist nur die Titelzeile, in der die PPN spezifiziert wird
       und die mit SET: beginnen muss. */
    Entry (LinkedList <String> input)
    {
        Iterator <String> x = input.listIterator();
        while (x.hasNext())
        {
            String number = "";
            String a = "";

            boolean b_rest = true;

            String t = x.next();

            if (t.startsWith("SET:"))
            {
                process_ppn(t);
```

```
                b_rest=false;
            }
            else
            {
                s_content = s_content.concat(t);
                s_content = s_content.concat("\n");

                number = getNumber(t);
                a = getRest(t);
            }

            if (number.equals("005")) { process_code(a); b_rest=false; };
            if (number.equals("020")) { b_rest=false; };
            if (number.equals("021")) { b_rest=false; };
            if (number.equals("800")) { process_name(a); b_rest=false; };
            if (number.equals("801")) { process_multiple_names(a); b_rest=
                false; };
            if (number.equals("802")) { process_multiple_names(a); b_rest=
                false; };
            if (number.equals("803")) { process_multiple_names(a); b_rest=
                false; };
            if (number.equals("804")) { process_multiple_names(a); b_rest=
                false; };
            if (number.equals("805")) { process_multiple_names(a); b_rest=
                false; };
            if (number.equals("808")) { process_desc(a); b_rest=false; };
            if (number.equals("810")) { process_cate(a); b_rest=false; };
            if (number.equals("811")) { process_origin(a); b_rest=false; };
            if (number.equals("820")) { process_altname(a); b_rest=false;
                };
            if (number.equals("830")) { process_altname(a); b_rest=false;
                };
            if (number.equals("845")) { process_para(a); b_rest=false; };
            if (number.equals("850")) { process_para(a); b_rest=false; };
            if (number.equals("860")) { process_rel(a);  b_rest=false; };
            if (number.equals("900")) { b_rest=false; };
            if (number.equals("950")) { process_changeComment(a); b_rest=
                false; };
            if (number.equals("0"))   { b_rest = false; };

            if (b_rest)
            {
                String name = "␣";
                if (l_namen.size() > 0)
                {
                    name = name.concat(l_namen.get(0));
                }
                SystemLogger.warn("Nicht␣beruecksichtigter␣Zahlcode:␣" +
                    number + name);
            }
        }

        checkContent();
    }

    // # Methoden ##################################################
```

```
// ### Schnittstellen ###

// liefert den Hauptnamen des Eintrags
public String getName ()
{
    return(l_namen.get(0));
}

// liefert alle Namen des Eintrags
public LinkedList <String> getAllNames ()
{
    LinkedList <String> result = new LinkedList<String>();

    for ( String i : l_namen )
    {
        result.add(i);
    }
    return(result);
}

// Liefert die PPN des Eintrags
public String getPPN()
{
    return(s_ppn);
}

// Zusammengesetzter Begriff?
public boolean getCombined()
{
    return b_combined;
}

// Finde alle Oberbegriffe
public LinkedList <String> getParents()
{
    LinkedList <String> result = new LinkedList<String>();

    for (String t : h_oberbegriff.keySet())
    {
        result.add(t);
    }

    return(result);
}

// Liefert die Beschreibung des Entries fuer das Ausgabefenster.
public String getContent()
{
    return(s_content);
}

// ### Kontruktionsmethoden ###
//
   ##########################################################################################


// SET:
```

```
    private void process_ppn(String a)
    {
        String result = "";

        Matcher m = ppnPattern.matcher(a);
        if (m.find())
        {
            result = m.group(0).substring(4);
        }
        result = result.trim();
        s_ppn = result;
        b_ppn = true;
    }

    // 005
    private void process_code(String a)
    {
//      s_code = a;
        b_code = true;
    }

    /* 800 - Name
       Hier aufpassen, dass man ggf. den Typ (z.B. |s|) mit extrahiert: */
    private void process_name(String a)
    {
        l_typen.add(0,getIdentifyer(a));
        l_namen.add(0,stripIdentifyer(a));

        b_name = true;
    }

    /* 801-805 Mehrgliedriger Schlagwortname */
    private void process_multiple_names(String a)
    {

        String name = l_namen.get(0);
        name = name.concat(" / " + stripIdentifyer(a));
        l_namen.set(0,name);
    }

    /* 808 Quelle (|a|) und Beschreibung (|b|), mehrere moeglich */
    private void process_desc(String a)
    {
//      String test = getIdentifyer(a);
//      if (test.equals("a")) { s_source = stripIdentifyer(a); };
//      if (test.equals("b")) { s_description = stripIdentifyer(a); };
//
//      b_desc = true;
    }

    // 810 - Kategorie
    private void process_cate(String a)
    {
        //s_category = a;
    };

    // 811 - Laendercode
```

```
    private void process_origin(String a)
    {
//      s_origin = a;
    }

    // 830 Alternative Namen, i.e., Synonyme
    private void process_altname(String a)
    {
        if (!getIdentifyer(a).equals("v"))
        {
            l_typen.add(getIdentifyer(a));
            l_namen.add(stripIdentifyer(a));
        }
    }

    /* 845 Zusammengesetzte Oberbegriffe, CSV, Schlagworttyp am Anfang nur
       fuer das erste. Daher werden diese Typen geloescht.
       Problem: Permutiert
       850 Einfache Oberbegriffe werden ebenfalls hier abgelegt*/

    // private LinkedList <String> oberbegriffPer;
    private void process_para(String a)
    {
        String s_input = stripIdentifyer(a);
        s_input = s_input.trim();
        h_oberbegriff.put(s_input, true);
    }

    // 860 - Verwandte Begriffe
    private void process_rel(String a)
    {
        String s_result = stripIdentifyer(a);
        l_verwandte.add(s_result);
    }

    /* 950 - Bitte um aenderung */
    private void process_changeComment(String a)
    {
        changeCommentary = a;
    }

    private void checkContent()
    {
        if (!b_code) { SystemLogger.warn("Kritischer Fehler: Code: NN.");
            };
        if (!b_name) { SystemLogger.error("Kritischer Fehler: Name: NN.");
            };
        if (!b_ppn)  { SystemLogger.warn("PPN: NN. bei " + l_namen.get(0));
             };
    }

    // ### Hilfsmethoden ###

    // Liefert die Codenummer einer Textzeile
    private String getNumber(String a)
    {
        a = a.trim();
```

```
        if (a.isEmpty()) { return("0"); }

        String[] result = a.split("\\s",2);

        return(result[0]);
    }

    // Liefert den Inhalt einer Textzeile
    private String getRest(String a)
    {
        a = a.trim();
        if (a.isEmpty()) { return(""); }

        String[] result = a.split("\\s",2);

        return(result[1]);
    };

    // Methode zum Extrahieren eines anfaenglichen Identifyers (z.B. "|a|")
    private String getIdentifyer(String x)
    {
        char[] type = x.substring(0,3).toCharArray();
        String s_result = "";

        if ((type[0] == '|') && (type[2] == '|'))
        {
            s_result = String.valueOf(type[1]);
        }

        return(s_result);
    }

    // Methode zum Entfernen eines anfaenglichen Identifyers (z.B. "|a|")
    private String stripIdentifyer(String x)
    {
        char[] type = x.substring(0,3).toCharArray();
        String s_result = x;

        if ((type[0] == '|') && (type[2]== '|'))
        {
            s_result = x.substring(3);
        }
        s_result = s_result.trim();
        return(s_result);
    }

    // Liefert  Kategorie 950, sonst null
    public String getChangeCommentary()
    {
        return(changeCommentary);
    }

    // Liefert Kategorie 810
    /*
    public String getKate()
    {
        return(s_category);
```

```
    }
    */
}
```

A.2.3 Klasse SWDdata.Relation

Die Klasse `SWDrelation`. Eine Instanz dieser Klasse entspricht genau einer Ober-/Unterbegriffsbeziehung zweier Schlagworte. Die Instanzen werden von der Klasse `SWDdata.Database` verwaltet.

```
package SWDdata;

public class Relation
{
    String s_parent;
    String s_child;

    public Relation (String a, String b)
    {
        s_parent = a;
        s_child = b;
    }

    public String getChild()  { return(s_child); };
    public String getParent() { return(s_parent);};
}
```

A.3 Paket SWDio – Datenein- und Ausgabe

Dieses Paket dient der Datenausgabe. Eingabeklassen finden sich hier (zur Zeit) nicht, da das Einlesen von Daten aus Schlagwortdatensätzen über die Klasse SWDdata.Database geschieht. Wenn zukünftige Implementierungen des SWD-Explorers das Einlesen alternativer Formate (z.B. SKOS) umsetzen, wird an dieser Stelle eine bessere Kapselung erforderlich sein. In diesem Fall würden die entsprechenden Eingabemethoden und -klassen in diesem Paket annzusiedeln sein.

Zur Zeit wird in diesem Paket nur die Ausgabe von Systemnachrichten gehandhabt.

A.3.1 Klasse SWDlogger

Die Klasse `SWDio.Logger`. Das globale Logging-Modul, über das alle Systemnachrichten laufen.

```
package SWDio;

import java.io.*;
import java.util.concurrent.locks.ReentrantLock;

import javax.swing.JTextArea;

public class SystemLogger
{
    // Variablen
    private static ReentrantLock lock = new ReentrantLock();
    private static String outfile = "SWDlog.log";
    private static JTextArea outWindow = null;

    // Standard-Konstruktor
    // public SWDlogger () {};

    // Methoden
    public static void setOutfile(String file) { outfile = file; };

    public static void setOutWindow(JTextArea a)     { outWindow = a; };

    public static void info(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Info: " + message );
            System.out.println( "Info: " + message);
            out.newLine();
            out.close();

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
            System.err.println("Logfile IO-Fehler!");
            System.err.println(e);
```

```
            System.exit(1);
        }

        if (!(outWindow == null))
        {
            outWindow.setCaretPosition( outWindow.getDocument().getLength()
                );
            outWindow.append(message + "\n");
        }
    }

    public static void warn(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Warnung: " + message );
            out.newLine();
            out.close();

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
            System.err.println("Logfile IO-Fehler!");
            System.exit(1);
        }

        if (!(outWindow == null))
        {
            outWindow.setCaretPosition( outWindow.getDocument().getLength()
                );
            outWindow.append("Warnung: " + message + "\n");
        }
    }

    public static void error(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Error: " + message );
            System.out.println( "Error: " + message );
            out.newLine();
            out.close();
            System.exit(1);

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
```

```
                System.err.println("Logfile IO-Fehler!");
                System.exit(1);
            }
        }
}
```

A.4 Paket SWDgui – Grafische Oberfläche

Dieses Paket dient der Definition der grafischen Benutzeroberfläche (GUI, *Graphical User Interface*), aber auch der Visualisierung. Die Klassen und Methoden, die der Visualisierung dienen, sind an anderer Stelle (Paket SWDvisualization) verortet.

A.4.1 Klasse SWDgui.ExplorerGui

Die Klasse `SWDgui.ExplorerGui` definiert die grafische Benutzeroberfläche des SWD-Explorers.

```
package SWDgui;

import SWDdata.Database;
import SWDgui.requestHandlers.TextParser;
import SWDio.SystemLogger;
import SWDvisualization.VisualizationMain;

import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.Toolkit;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.StringSelection;
import java.awt.event.*;

import javax.swing.*;

import org.jgraph.JGraph;
import org.jgraph.graph.DefaultGraphCell;

import java.io.*;
import java.util.LinkedList;

public class ExplorerGui
{
    Database database = null;

    JPanel leftpanel = null;
    JPanel mainpanel = null;
    JMenu fileMenu = null;
    JFrame frame = null;
    JTextField textIn  = null; // Feld zur Eingabe von Befehlen
    JTextArea descriptionOut = null; // Feld mit der Beschreibung der
        Eintraege
    JTextArea textOut = null; // Feld zu allgemeinen Ausgabe von Text
    JGraph graphArea = null; // Graphenfeld

    RecentlyOpenedFilesManager recOpened = null;

    String currentPPN = "";
    String currentName = "";
    String currentPICA = "";

    LinkedList<String> history = new LinkedList<String>();
    int historyCounter = 0;
```

```
int childrenDepth = 1;
int parentsDepth = 4;
boolean showSiblings = false; //Noch nicht implementiert

int breaksize = 500;

// Welcher Suchmodus ist ausgewaehlt?
public final static int SHOW_REGEXP = 0;
public final static int SHOW_EXACT  = 1;
public final static int SHOW_TRUNC  = 2;
public final static int SHOW_OPAC   = 3;
public final static int SHOW_PPN    = 4;
int searchMode = SHOW_TRUNC;

// Konstuktor
public ExplorerGui (Database x)
{
    database = x;
}

public void createGUI()
{
    // ### Definition der Menues
       ##############################################
    JMenuBar menuBar = new JMenuBar();

    // # Datei-Menue ###
    recOpened = new RecentlyOpenedFilesManager();

    fileMenu = new JMenu("Datei");

    JMenuItem da = new JMenuItem("Datei oeffnen");
    JMenuItem dh = new JMenuItem("Speichere Hauptansetzungsformen in...
       ");
    JMenuItem dc = new JMenuItem("Speichere alle Ansetzungsformen in...
       ");
    JMenuItem db = new JMenuItem("Beenden");

    fileMenu.add( da );
    fileMenu.add( dh );
    fileMenu.add( dc );
    fileMenu.add( db );

    // Einfuegen der variablen Eintraege im Datei-Menu
    updateFileMenu();

    menuBar.add( fileMenu );

    // View-Menue
    JMenu viewMenu = new JMenu( "Bearbeiten" );

    JMenuItem vn1 = new JMenuItem("Kopiere PPN");
    ActionListener al = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            if (!currentPPN.equals(""))
```

```
            {
                Clipboard systemClip = Toolkit.getDefaultToolkit().
                    getSystemClipboard();
                systemClip.setContents(new StringSelection(currentPPN),
                     null);
                SystemLogger.info("Kopiere Text in Zwischenablage: " +
                    currentPPN);
            }
        }
    };
    vn1.addActionListener(al);

    JMenuItem vn2 = new JMenuItem("Kopiere Hauptansetzungsform");

    al = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            if (!currentName.equals(""))
            {
                Clipboard systemClip = Toolkit.getDefaultToolkit().
                    getSystemClipboard();
                systemClip.setContents(new StringSelection(currentName)
                    , null);
                SystemLogger.info("Kopiere Text in Zwischenablage: " +
                    currentName);
            }
        }
    };

    vn2.addActionListener(al);

    JMenuItem vn3 = new JMenuItem("Kopiere Eintrag");

    al = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            if (!currentName.equals(""))
            {
                Clipboard systemClip = Toolkit.getDefaultToolkit().
                    getSystemClipboard();
                systemClip.setContents(new StringSelection(currentPICA)
                    , null);
                SystemLogger.info("Kopiere Eintrag in Zwischenablage");
            }
        }
    };

    vn3.addActionListener(al);

    viewMenu.add( vn1 );
    viewMenu.add( vn2 );
    viewMenu.add( vn3 );

    menuBar.add( viewMenu );
```

```
// Hilfe-Menue
JMenu helpMenu = new JMenu( "Hilfe" );
JMenuItem h1 = new JMenuItem("Version");

al = new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        SystemLogger.info("SWD-Explorer Version 1.01");
    }
};

h1.addActionListener(al);
helpMenu.add( h1 );

menuBar.add( helpMenu );

// Vorwaerts-Rueckwaerts-Knoepfe
JButton b1 = new JButton("<-");
JButton b2 = new JButton("->");

al = new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        if (historyCounter > 0)
        {
            historyCounter--;
            processTextInput(history.get(historyCounter));
        }
    }
};

b1.addActionListener(al);

al = new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        if ((history.size()-1) > historyCounter)
        {
            historyCounter++;
            processTextInput(history.get(historyCounter));
        }
    }
};
b2.addActionListener(al);

menuBar.add(b1);
menuBar.add(b2);

// ### Anlegen des Main-Frames
   ##############################################

frame = new JFrame( "SWD-Explorer V1.0.1" );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
//JPanel mainpanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

// Das mainpanel ist der Container, der die gesamte Flaeche abdeckt
mainpanel = new JPanel();
mainpanel.setLayout( new BorderLayout() );

// Das Menupanel ist die Kontroll- und Eingabeleiste auf der oberen
    Seite
JPanel menupanel = new JPanel();
mainpanel.add(menupanel, BorderLayout.PAGE_START);

// Das leftpanel besteht aus den beiden Textfenstern auf der linken
    Seite
leftpanel = new JPanel();
leftpanel.setLayout(new BoxLayout(leftpanel, BoxLayout.Y_AXIS));
mainpanel.add(leftpanel, BorderLayout.LINE_START);

// Texteingabefeld definieren
textIn = new JTextField( "Suchbegriff", 25 );
menupanel.add(menuBar);
menupanel.add(textIn);

// Such-Button definieren
JButton b3 = new JButton("Suchen");

// Texteingabe, bestaetigt durch den "GO"-Button
al = new ActionListener()
{
    public void actionPerformed(ActionEvent evt)
    {
        String text = textIn.getText();
        textIn.selectAll();
        processTextInput(text);
        while ((history.size()-1) > historyCounter)
        {
            history.removeLast();
        }
        history.add(text);
        historyCounter = (history.size() - 1);
    }
};
b3.addActionListener(al);
menupanel.add(b3);

// Combo-Box zur Konfiguration der Texteingabe

JComboBox inputSelectorBox = new JComboBox();

inputSelectorBox.addItem("Exakte Uebereinstimmung");
inputSelectorBox.addItem("Trunkierung (rechts)");
inputSelectorBox.addItem("OPAC-Stil");
inputSelectorBox.addItem("Regulaerer Ausdruck");
inputSelectorBox.addItem("PPN");
inputSelectorBox.setSelectedItem("Trunkierung (rechts)");
menupanel.add(inputSelectorBox);

// Combo-Boxen zur Eingabe der Eltern/Kindertiefe
```

```
JComboBox childrenBox = new JComboBox();
JComboBox parentBox = new JComboBox();

childrenBox.addItem("0 Unterbegriffe");
childrenBox.addItem("1 Unterbegriff");
childrenBox.addItem("2 Unterbegriffe");
childrenBox.addItem("3 Unterbegriffe");
childrenBox.addItem("4 Unterbegriffe");
childrenBox.addItem("unendlich");
childrenBox.setSelectedItem("1 Unterbegriff");

parentBox.addItem("0 Oberbegriffe");
parentBox.addItem("1 Oberbegriff");
parentBox.addItem("2 Oberbegriffe");
parentBox.addItem("3 Oberbegriffe");
parentBox.addItem("4 Oberbegriffe");
parentBox.addItem("unendlich");
parentBox.setSelectedItem("4 Oberbegriffe");


/*
 * Optimierung fuer 800x600!
 */
/*
inputSelectorBox.addItem("Exakt");
inputSelectorBox.addItem("Trunkiert");
inputSelectorBox.addItem("OPAC-Stil");
inputSelectorBox.addItem("Reg. Exp.");
inputSelectorBox.addItem("PPN");
inputSelectorBox.setSelectedItem("Trunkiert");
menupanel.add(inputSelectorBox);

childrenBox.addItem("0 Unterb.");
childrenBox.addItem("1 Unterb.");
childrenBox.addItem("2 Unterb.");
childrenBox.addItem("3 Unterb.");
childrenBox.addItem("4 Unterb.");
childrenBox.addItem("unendlich");
childrenBox.setSelectedItem("1 Unterb.");

parentBox.addItem("0 Oberb.");
parentBox.addItem("1 Oberb.");
parentBox.addItem("2 Oberb.");
parentBox.addItem("3 Oberb.");
parentBox.addItem("4 Oberb.");
parentBox.addItem("unendlich");
parentBox.setSelectedItem("4 Oberb.");
*/

menupanel.add(childrenBox);
menupanel.add(parentBox);

// allgemeines Textausgabefeld definieren
textOut = new JTextArea(20,40);
textOut.setEditable(false);

textOut.append("Systeminformationen\n\n");
```

```
SystemLogger.setOutWindow(textOut);

leftpanel.add(new JScrollPane(textOut));

// Definition des Feldes zur Ausgabe der Objektinformation
descriptionOut = new JTextArea(20,40);
descriptionOut.setEditable(false);

descriptionOut.append("Objektinformationen\n");

leftpanel.add(new JScrollPane (descriptionOut));

// Anzeigen des Graphenfeldes, Implementierung des MouseListeners
{
    JScrollPane temp = VisualizationMain.helloSWD();
    final JGraph temp2 = (JGraph) temp.getViewport().getView();

    MouseListener ml = new MouseListener()
    {
        public void mouseClicked(MouseEvent e)
        {
            if (SwingUtilities.isRightMouseButton(e))
            {
                DefaultGraphCell cell = (DefaultGraphCell) temp2.
                    getSelectionCell();
                String name = (String) cell.getUserObject();
                descriptionOut.append("\n" + name);
            }
        }

        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseExited(MouseEvent e) {}
        @Override
        public void mousePressed(MouseEvent e) {}
        @Override
        public void mouseReleased(MouseEvent e) {}
    };

    temp2.addMouseListener(ml);
    mainpanel.add(temp, BorderLayout.CENTER);
}

// ### Definition der Aktionen
   ##########################################

// Combo-Boxen
inputSelectorBox.addActionListener( new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
      JComboBox selectedChoice = (JComboBox) e.getSource();
      if (selectedChoice.getSelectedItem().equals("Exakte␣
          Uebereinstimmung")) {searchMode = SHOW_EXACT;};
      if (selectedChoice.getSelectedItem().equals("Trunkierung␣(
          rechts)")) {searchMode = SHOW_TRUNC;};
```

```
        if (selectedChoice.getSelectedItem().equals("OPAC-Stil")) {
            searchMode = SHOW_OPAC;};
        if (selectedChoice.getSelectedItem().equals("Regulaerer␣
            Ausdruck")) {searchMode = SHOW_REGEXP;};
        if (selectedChoice.getSelectedItem().equals("PPN")) {
            searchMode = SHOW_PPN;};
      }
    } );

childrenBox.addActionListener( new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
      JComboBox selectedChoice = (JComboBox) e.getSource();
      if (selectedChoice.getSelectedItem().equals("0␣Unterbegriffe"
          )) {childrenDepth = 0;};
      if (selectedChoice.getSelectedItem().equals("1␣Unterbegriff")
          ) {childrenDepth = 1;};
      if (selectedChoice.getSelectedItem().equals("2␣Unterbegriffe"
          )) {childrenDepth = 2;};
      if (selectedChoice.getSelectedItem().equals("3␣Unterbegriffe"
          )) {childrenDepth = 3;};
      if (selectedChoice.getSelectedItem().equals("4␣Unterbegriffe"
          )) {childrenDepth = 4;};
      if (selectedChoice.getSelectedItem().equals("unendlich")) {
          childrenDepth = 99;};
    }
  } );

parentBox.addActionListener( new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
      JComboBox selectedChoice = (JComboBox) e.getSource();
      if (selectedChoice.getSelectedItem().equals("0␣Oberbegriffe")
          ) {parentsDepth = 0;};
      if (selectedChoice.getSelectedItem().equals("1␣Oberbegriff"))
           {parentsDepth = 1;};
      if (selectedChoice.getSelectedItem().equals("2␣Oberbegriffe")
          ) {parentsDepth = 2;};
      if (selectedChoice.getSelectedItem().equals("3␣Oberbegriffe")
          ) {parentsDepth = 3;};
      if (selectedChoice.getSelectedItem().equals("4␣Oberbegriffe")
          ) {parentsDepth = 4;};
      if (selectedChoice.getSelectedItem().equals("unendlich")) {
          parentsDepth = 99;};
    }
  } );

/* Texteingabe, bestaetigt durch "Return" im Eingabefeld.
   Texteingabe, die durch den "Suchen"-Button ausgeloest wird,
   wurde weiter oben als ActionListener definiert */
ActionListener textEingabe = new ActionListener()
{
    public void actionPerformed(ActionEvent evt)
    {
        String text = textIn.getText();
```

```
            textIn.selectAll();
            processTextInput(text);
            while ((history.size()-1) > historyCounter)
            {
                history.removeLast();
            }
            history.add(text);
            historyCounter = (history.size() - 1);
        }
    };
    textIn.addActionListener(textEingabe);

    // Aktionen im Datei-Menu

    // da = Datei Oeffnen
    // dh = Schlagwort-Hauptansetzungsformen schreiben
    // dc = alle Ansetzungsformen schreiben
    // db = Beenden

    ActionListener dateiBeenden = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            System.exit( 0 );
        }
    };
    db.addActionListener(dateiBeenden);

    ActionListener dateiOeffnen = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            JFileChooser chooser = new JFileChooser();
            int state = chooser.showOpenDialog( null );

            if ( state == JFileChooser.APPROVE_OPTION )
            {
                File file = chooser.getSelectedFile();
                SystemLogger.info( "Oeffnen der Datei: " + file.getName
                    () );
                recOpened.setRecentlyOpenedFile(file.getAbsolutePath())
                    ;
                updateFileMenu();
                database.read(file);
            }
            else
            {
                SystemLogger.info( "Auswahl abgebrochen" );
            }
        }
    };
    da.addActionListener(dateiOeffnen);

    ActionListener hauptformenSchreiben = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
```

```
            JFileChooser chooser = new JFileChooser();
            int state = chooser.showOpenDialog( null );

            if ( state == JFileChooser.APPROVE_OPTION )
            {
                File file = chooser.getSelectedFile();
                SystemLogger.info( "Schreibe aktuelle
                    Hauptansetzungsformen in Datei: " + file.getName() )
                    ;

                BufferedWriter bf = null;
                try
                {
                  bf = new BufferedWriter( new FileWriter( file ));
                  String[] names = database.getNames();
                  for (String name : names)
                  {
                      bf.append(name);
                      bf.newLine();
                  }
                }
                catch ( IOException erro )
                {
                  System.err.println( "Konnte Datei nicht erstellen" );
                }
                finally
                {
                    try { bf.close(); } catch ( IOException er ) {}
                }
            }
            else
            {
                SystemLogger.info( "Auswahl abgebrochen" );
            }
        }
    };
    dh.addActionListener(hauptformenSchreiben);

    ActionListener alleAnsetzungsformenSchreiben = new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            JFileChooser chooser = new JFileChooser();
            int state = chooser.showOpenDialog( null );

            if ( state == JFileChooser.APPROVE_OPTION )
            {
                File file = chooser.getSelectedFile();
                SystemLogger.info( "Schreibe alle aktuellen
                    Ansetzungsformen in Datei: " + file.getName() );

                BufferedWriter bf = null;
                try
                {
                  bf = new BufferedWriter( new FileWriter( file ));
                  LinkedList<String> names = database.getAllNames();
                  for (String name : names)
```

```
                    {
                        bf.append(name);
                        bf.newLine();
                    }
                }
                catch ( IOException erro )
                {
                  System.err.println( "Konnte Datei nicht erstellen" );
                }
                finally
                {
                    try { bf.close(); } catch ( IOException er ) {}
                }
            }
            else
            {
                SystemLogger.info( "Auswahl abgebrochen" );
            }
        }
    };
    dc.addActionListener(alleAnsetzungsformenSchreiben);

    // Aktionen im View-Menue

    // vn1 = Zeige Nachbarn
    // vn2 = zeige Namen

    // Ausgabe des Fensters
    frame.add(mainpanel);
    frame.pack();
    frame.setVisible( true );
}

// Schnittstelle zum Darstellen von Text
public void processTextInput(String text)
{
    SystemLogger.info("Auswahl: " + text);
    VisualizationMain visualisierung = new VisualizationMain();
    visualisierung.setDatabase(database);
    visualisierung.setBreaksize(breaksize);

    // Zur Steigerung der Usability wird das Texteingabefenster auf den
        neuen Wert gesetzt
    textIn.setText(text);

    TextParser parser = new TextParser(visualisierung,database,
       childrenDepth,parentsDepth,showSiblings,searchMode);
    visualisierung = parser.getGraph(text);

    if (visualisierung != null)
    {
        updateWindow(visualisierung);
    }
}

/* Die folgende Methode verwaltet die variablen Eintraege im Dateimenu,
 * also die Aufrufe der in der History gespeicherten Dateien.
```

```
 */
private void updateFileMenu()
{
    int itemCount = fileMenu.getMenuComponentCount();
    LinkedList<String> files = recOpened.getRecentlyOpenedFiles();

    /* Loeschen der alten Eintraege im Dateimenu */
    LinkedList <Integer> toDel = new LinkedList<Integer>();

    if (itemCount > 4)
    {
        for (int z = 4; z < itemCount; z++)
        {
            fileMenu.remove(4);
        }
    }

    if (files.size() > 0)
    {
        LinkedList<JMenuItem> entries = new LinkedList<JMenuItem>();
        fileMenu.addSeparator();
        int r = 0;
        for (String file : files)
        {
            r++;
            int a = 0;
            int a1 = file.lastIndexOf("\\");
            int a2 = file.lastIndexOf("/");
            if (a1 > a2)
            {
                a = a1;
            }
            else
            {
                a = a2;
            }
            file = file.substring(a+1);

            JMenuItem fileIns = new JMenuItem(Integer.toString(r) + "␣"
                + file);
            fileMenu.add(fileIns);
            entries.add(fileIns);
        }

        /* ActionListener anlegen */

        for (int q = 0; q < files.size(); q++)
        {
            final JMenuItem menuItem = entries.get(q);
            final String filename = files.get(q);

            ActionListener specOpen= new ActionListener()
            {
                public void actionPerformed( ActionEvent e )
                {
                        SystemLogger.info( "Oeffnen␣der␣Datei:␣" +
                            filename );
```

```
                        database.read(new File(filename));
                    }
                };
                menuItem.addActionListener(specOpen);
            }
        }
        fileMenu.updateUI();
    }

    // Modifikation des Fensters anhand eines neu generierten Graphmodells
    private void updateWindow(VisualizationMain graph)
    {
        // Testen, ob das Gesamtfenster maximiert ist
        boolean maximized = false;
        int state = frame.getExtendedState();
        if ((state & Frame.MAXIMIZED_BOTH) == Frame.MAXIMIZED_BOTH)
        {
            maximized = true;
        }

        mainpanel.remove(2);

        final JScrollPane temp = graph.getGraphScrollPane();
        final JGraph temp2 = (JGraph) temp.getViewport().getView();

        // Passenden MouseListener anlegen
        MouseListener ml = new MouseListener()
        {
            public void mouseClicked(MouseEvent e)
            {
                if (SwingUtilities.isRightMouseButton(e))
                {
                    try
                    {
                        // Einfachklick, also nur Details anzeigen
                        if (e.getClickCount() == 1)
                        {
                            DefaultGraphCell cell = (DefaultGraphCell)
                                temp2.getSelectionCell();
                            String name = (String) cell.getUserObject();
                            String insert = database.getDescription(name);
                            descriptionOut.setText("");
                            descriptionOut.append(insert);
                            currentName = database.getNameByName(name);
                            currentPPN = database.getPPNByName(name);
                            currentPICA = database.getDescription(
                                currentName);
                        }
                        // Mehrfachklick, also Selektion und erneuter
                            Aufbau
                        else
                        {
                            DefaultGraphCell cell = (DefaultGraphCell)
                                temp2.getSelectionCell();
                            String name = (String) cell.getUserObject();
                            name = database.getNameByName(name);
```

```
                            int currentSearchMode = searchMode;
                            searchMode = SHOW_EXACT;
                            processTextInput(name);
                            searchMode = currentSearchMode;

                            while ((history.size()-1) > historyCounter)
                            {
                                history.removeLast();
                            }
                            history.add(name);
                            historyCounter = (history.size() - 1);
                        }
                    }
                    catch (java.lang.NullPointerException x) {};
                }
            }

            @Override
            public void mouseEntered(MouseEvent e) {}
            @Override
            public void mouseExited(MouseEvent e) {}
            @Override
            public void mousePressed(MouseEvent e) {}
            @Override
            public void mouseReleased(MouseEvent e) {}
        };

        temp2.addMouseListener(ml);

        // Und: fertig!
        mainpanel.add(temp,BorderLayout.CENTER);
        frame.pack();
        frame.validate();
        for ( int i = 0; i<3; i++ )
        {
            try { Thread.sleep(500);}  catch (Exception e) { };
            if (maximized)
            {
                frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
            }
        }
    }
}
```

A.4.2 Klasse RecentlyOpenedFilesManager

Die Klasse `RecentlyOpenedFilesManager` verwaltet die Anzeige zu einem früheren Zeitpunkt geöffneter Dateien im Datei-Menu.

```
package SWDgui;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.LinkedList;

import SWDio.SystemLogger;

public class RecentlyOpenedFilesManager
{
    String configFileName = "recentlyOpened.txt";
    File configFile = new File(configFileName);

    LinkedList<String> files = new LinkedList<String>();

    public RecentlyOpenedFilesManager ()
    {
        if (! configFile.canWrite())
        {
            SystemLogger.warn("Kann Konfigurationsdatei nicht schreiben!");
            return;
        }

        if (! configFile.exists())
        {
            SystemLogger.info("Erzeuge neue Konfigurationsdatei.");
            try
            {
                configFile.createNewFile();
            }
            catch (IOException e)
            {
                SystemLogger.warn("Kann Konfigurationsdatei nicht schreiben
                    !");
                return;
            }
        }
    }

    public LinkedList<String> getRecentlyOpenedFiles()
    {
        files = new LinkedList<String>();
        try
        {
            BufferedReader reader = new BufferedReader(new FileReader(
                configFile));
            boolean c = true;
            while (c)
            {
```

```
                String line = reader.readLine();
                if (line == null)
                {
                    c = false;
                }
                else
                {
                    files.add(line);
                }
            }
            reader.close();

        }
        catch (Exception e)
        {
            SystemLogger.warn("Fehler beim Lesen der Konfigurationsdatei");
        }
        return files;
    }

    public void setRecentlyOpenedFile(String filename)
    {
        int isAlready = 0;
        int i = 0;
        for (String list : files)
        {
            if (filename.equals(list))
            {
                isAlready = i;
            }
            i++;
        }

        if (isAlready > 0)
        {
            files.remove(isAlready);
        }

        files.addFirst(filename);
        while (files.size() > 4)
        {
            files.removeLast();
        }
        writeFile();
    }

    public void clearRecentlyOpenedFiles()
    {
        files = new LinkedList<String>();
        writeFile();
    }

    private void writeFile()
    {
        try
        {
            PrintWriter out = new PrintWriter( configFileName );
```

```
            for (String line : files)
            {
                out.println(line);
            }
            out.close();
        }
        catch (Exception e) { SystemLogger.warn("Fehler beim Schreiben der
            Konfigurationsdatei"); }
    }
}
```

A.4.3 Klasse CellMouseListener

Die Klasse `CellMouseListener` ist die Listener-Klasse, die die Maus-Interaktion mit den Schlagwortknoten kontrolliert.

```
package SWDgui;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import org.jgraph.*;
import org.jgraph.graph.DefaultGraphCell;
import javax.swing.SwingUtilities;

public class CellMouseListener implements MouseListener {

    JGraph graph = null;

    public CellMouseListener(JGraph j) { graph = j; }

    @Override
    public void mouseClicked(MouseEvent e) {
        if (SwingUtilities.isRightMouseButton(e))
        {
            // Einfachklick, also nur Details anzeigen
            if (e.getClickCount() == 1)
            {
                System.out.println("Ein Klick genuegt!");
                DefaultGraphCell cell = (DefaultGraphCell) graph.
                    getSelectionCell();
                String name = (String) cell.getUserObject();
                System.out.println(name);
            }
            // Mehrfachklick, also Selektion
            else
            {
                System.out.println("Mehr Klicks!");
            }
        }
    }

    @Override
    public void mouseEntered(MouseEvent arg0) {}

    @Override
    public void mouseExited(MouseEvent arg0) {}

    @Override
    public void mousePressed(MouseEvent arg0) {}

    @Override
    public void mouseReleased(MouseEvent arg0) {}
}
```

A.5 Paket SWDgui.requestHandlers – intelligente Suche

Dieses Paket enthält die benötigten Klassen zum Interpretieren der Suchanfragen. Für die Suchmodi (z.B. Exakte Übereinstimmung oder Suche mit Regulären Ausdrücken) existieren in diesem Paket RequestHandler, die die relevanten Knoten ausgeben und an die Visualisierungsmethoden weiterleiten.

A.5.1 Klasse SWDgui.requestHandlers.Textparser

Diese Klasse ist sowohl für die Suche nach exakt mit dem Suchbegriff übereinstimmenden Ansetzungsformen und nach Ansetzungsformen, die mit dem Anfang des Suchbegriffs übereinstimmen gedacht, ebenso wie für die Suche mit Hilfe von Regulären Ausdrücken.

```
package SWDgui.requestHandlers;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import SWDdata.Database;
import SWDgui.ExplorerGui;
import SWDio.SystemLogger;
import SWDvisualization.VisualizationMain;

/* Diese Klasse verarbeitet textuelle Eingaben des Benutzers
 *
 * Erste Funktionalitaet:
 * Der Parser zeigt den Begriff, die Kinder und die Eltern
 * Wie tief jeweils in die Ebenen der Kinder und Eltern exploriert
 * wird, wird ueber einen Parametersatz bei Klassenkonstruktion festgelegt.
 * Die Exploration erfolgt dann ueber Rekursion.
 */

public class TextParser
{
    private VisualizationMain graph;
    private Database database;

    int childrenDepth = 0;
    int parentsDepth = 0;
    boolean showSiblings = false; // Noch nicht implementiert
    int searchMode = 0;

    public TextParser(VisualizationMain g, Database d, int c, int p,
        boolean s, int sm)
    {
        graph = g;
        database = d;
        childrenDepth = c;
        parentsDepth = p;
        showSiblings = s;
        searchMode = sm;
    }
```

```
public VisualizationMain getGraph (String input)
{
    LinkedList<String>queriedElements = new LinkedList<String>();
    // Aufloesen der Anfrage entsprechend des Suchmodus
    if (searchMode == ExplorerGui.SHOW_REGEXP)
    {
        queriedElements = getNamesByRegExp(input);
    }

    if (searchMode == ExplorerGui.SHOW_EXACT)
    {
        LinkedList <String> names = database.getAllNames();
        for (String s : names)
        {
            if (s.equalsIgnoreCase(input))
            {
                if ((s != null) && (! s.equals("")))
                queriedElements.add(database.getNameByName(s));
            }
        }
    }

    if (searchMode == ExplorerGui.SHOW_TRUNC)
    {
        String smallinput = input.toLowerCase();
        LinkedList <String> names = database.getAllNames();
        for (String s : names)
        {
            String smalls = s.toLowerCase();

            if (smalls.startsWith(smallinput))
            {
                if ((s != null) && (! s.equals("")))
                queriedElements.add(database.getNameByName(s));
            }
        }
    }

    if (searchMode == ExplorerGui.SHOW_OPAC)
    {
        queriedElements = getNamesByOPAC(input);
    }

    if (searchMode == ExplorerGui.SHOW_PPN)
    {
        queriedElements = getNamesByPPN(input);
    }

    if (queriedElements.size() == 0)
    {
        SystemLogger.info("Es␣wurden␣keine␣matchenden␣AusdrÃ¼cke␣im␣
            Datensatz␣gefunden.");
        return(null);
    }

    // Farbe der primaeren Elemente setzen
```

```
        for (String name : queriedElements)
        {
            graph.setEntry(name,"red");
        }

        /* Diese Methode ist notwendig um festzustellen, ob ggf. die
            Begriffe
         * innerhalb dieser Liste Eltern/Kind-Beziehungen haben, die
            aufgrund der
         * auf 0 gesetzten Suchweite nicht angezeigt werden wuerden
         */
        if ((childrenDepth < 1) || (parentsDepth < 1))
        {
            getRelationsInCaseOfDepthEqualsZero(queriedElements);
        }

        // Jetzt die Rekursionen, um die entsprechenden Kinder und Eltern
            zu finden
        processRelatives(queriedElements, childrenDepth, parentsDepth);

        return(graph);
    }

    public VisualizationMain getGraphFromNameList (LinkedList<String> input
        )
    {
        LinkedList<String> queriedElements = new LinkedList<String>();

        for (String name : input)
        {
            String q = database.getNameByName(name);
            if (q != null)
            {
                queriedElements.add(q);
            }
        }

        if (queriedElements.size() == 0)
        {
            SystemLogger.info("Es wurden keine matchenden AusdrÃ¼cke im
                Datensatz gefunden.");
            return(null);
        }

        // Farbe der primaeren Elemente setzen
        for (String name : queriedElements)
        {
            graph.setEntry(name,"red");
        }

        /* Diese Methode ist notwendig um festzustellen, ob ggf. die
            Begriffe
         * innerhalb dieser Liste Eltern/Kind-Beziehungen haben, die
            aufgrund der
         * auf 0 gesetzten Suchweite nicht angezeigt werden wuerden
         */
        if ((childrenDepth < 1) || (parentsDepth < 1))
```

```
        {
            getRelationsInCaseOfDepthEqualsZero(queriedElements);
        }

        // Jetzt die Rekursionen, um die entsprechenden Kinder und Eltern
            zu finden
        processRelatives(queriedElements, childrenDepth, parentsDepth);

        return(graph);
    }

    // Methode, um alle Kindelemente einer Liste von Namen zu erhalten und
    // in den Graphen einzutragen
    private void processRelatives(LinkedList<String> names, int
        l_childrenDepth, int l_parentsDepth)
    {
        HashMap<String,Boolean>entriesToShow = new HashMap<String,Boolean
            >();

        entriesToShow.putAll(processChildren(names, l_childrenDepth));
        entriesToShow.putAll(processParents(names, l_parentsDepth));

        /* Jetzt, nachdem alle anzuzeigenden Elemente gefunden wurden, die
         * Relationen im Graphen setzen
         */
        for (String name : entriesToShow.keySet())
        {
            graph.setEntry(name);

            // Zusammengesetztes Element? Farbe!
            if (database.getCombined(name))
            {
                graph.setEntry(name,"green");
            }

            for (String child : database.getChildrenNames(name))
            {
                if (entriesToShow.containsKey(child))
                {
                    graph.setPCRelation(name, child);
                }
            }
        }
    }

    private HashMap<String,Boolean> processChildren(LinkedList<String>
        names, int l_depth)
    {
        HashMap<String,Boolean> result = new HashMap<String,Boolean>();
        if (l_depth < 1) { return result;};
        l_depth--;

        for (String name : names)
        {
            LinkedList<String> l_children = database.getChildrenNames(name)
                ;
            // Eintragen in die Graphstruktur
```

```
            for (String child : l_children)
            {
                result.put(name,true);
                result.put(child,true);
            }

            if ((l_depth > 0) && (l_children.size() > 0))
            {
                result.putAll(processChildren(l_children,l_depth));
            }
        }
        return(result);
    }

    // Methode, um alle Elternemente einer Liste von Namen zu erhalten und
    // in den Graphen einzutragen
    private HashMap<String,Boolean> processParents(LinkedList<String> names
        , int depth)
    {
        HashMap<String,Boolean> result = new HashMap<String,Boolean>();
        if (depth < 1) { return result; };
        depth--;
        for (String name : names)
        {
            LinkedList<String> l_parents = database.getParentNames(name);
            // Eintragen in die Graphstruktur
            for (String parent : l_parents)
            {
                result.put(name,true);
                result.put(parent,true);
            }

            if ((depth > 0) && (l_parents.size() > 0))
            {
                result.putAll(processParents(l_parents,depth));
            }
        }
        return(result);
    }

    /* Liefert alle matchenden Namen zurueck, als ob die Anfrage in einem
       OPAC
     * gestellt wurde. Soll heissen: AND, OR und Klammern sind als
         Suchangaben moeglich.
     */
    private LinkedList<String> getNamesByOPAC(String input)
    {
        OPACParser parser = new OPACParser (database);
        parser.setInput(input);
        return(parser.parseInput());
    }

    /* Liefert das Schlagwort zurueck, welches der PPN entspricht */
    private LinkedList<String> getNamesByPPN(String input)
    {
        LinkedList<String> result = new LinkedList<String>();
        String[] names = database.getNames();
```

```
        for (String name : names)
        {
            String ppn = database.getPPNByName(name);
            if ((!(ppn == null)) && ppn.equals(input))
            {
                result.add(name);
            }
        }
        return(result);
    }


//  Liefert alle matchenden Namen zurueck, normalisiert auf den
//  Standardnamen
//  Deshalb aus database ausgelagert, damit die Implementierung einer
//  Relationalen Datenbank einfacher moeglich ist
    private LinkedList<String> getNamesByRegExp(String input)
    {
        LinkedList<String> result = new LinkedList<String>();
        LinkedList<String> allNames = database.getAllNames();

        if (! (input.equals("") || input == null))
        {
            try
            {
                Pattern p = Pattern.compile(input, Pattern.CASE_INSENSITIVE
                    + Pattern.UNICODE_CASE);

                for (String name : allNames)
                {
                    Matcher m = p.matcher(name);
                    boolean b = m.matches();

                    if (b)
                    {
                        result.add(name);
                    }
                }
            }

            catch (java.util.regex.PatternSyntaxException e)
            {
                SystemLogger.warn("Nicht zulaessiger Suchbegriff: Kein
                    regulaerer Ausdruck!");
                return(new LinkedList<String>());
            }

            // Bereinigen der Liste
            HashMap<String,Boolean> trzt = new HashMap<String,Boolean>();
            Boolean blah = new Boolean(false);
            for (String name : result)
            {
                String g = database.getNameByName(name);
                trzt.put(g,blah);
            }
            result = new LinkedList<String>();
```

```
            for (String key : trzt.keySet())
            {
                result.add(key);
            }
        }
        return(result);
    }

    /* Diese Methode ist notwendig um festzustellen, ob ggf. die Begriffe
     * innerhalb dieser Liste Eltern/Kind-Beziehungen haben, die aufgrund
        der
     * auf 0 gesetzten Suchweite nicht angezeigt werden wuerden
     */
    private void getRelationsInCaseOfDepthEqualsZero (LinkedList<String>
        names)
    {
        HashMap<String,String> contained = new HashMap<String,String>();
        for (String name : names)
        {
            contained.put(name, "␣");
        }

        for (String name : names)
        {
            LinkedList<String> l_children = database.getChildrenNames(name)
                ;
            for (String element : l_children)
            {
                if (contained.containsKey(element))
                {
                    graph.setPCRelation(name, element);
                }
            }
        }
    }
}
```

A.5.2 Klasse SWDgui.requestHandlers.OPACParser

Mit Hilfe dieser Klasse lassen sich die OPAC-ähnlichen Suchanfragen behandeln, die mit Hilfe von UND und ODER verkettet wurden.

```
package SWDgui.requestHandlers;

import java.util.HashMap;
import java.util.LinkedList;

import SWDdata.Database;

public class OPACParser {

    Database database;
    String input = "";

    public OPACParser (Database d)
    {
        database = d;
    }

    public void setInput(String i)
    {
        input = i;
    }

    public LinkedList<String> parseInput()
    {
        //checkInput();
        LinkedList<String> result = processInput(input);

        System.out.println(result);

        return (result);
    }

    /* Idee: Erst einmal alles in die geklammerten Terme zerlegen, dann die
         einzelnen Dinge an eine
     * weitere Subtroutine geben, bei der die Konjunktionen zerlegt werden.
         Schliesslich
     * die einzelnen Suchanfragen auswerten
     * Problem: Bisher keine Ueberpruefung, ob der Term syntaktisch korrekt
         ist
     */

    private LinkedList <String> processInput(String input)
    {
        // Tokenisieren
        LinkedList <String> eingabe = new LinkedList<String>();
        input = input.replaceAll("\\(", " ( ");
        input = input.replaceAll("\\)", " ) ");
        String[] result = input.split("\\s");
          for (int x=0; x<result.length; x++)
          {
              String t = result[x];
              t.trim();
```

```
            if (!(result[x].equals("")))
            {
                eingabe.add(result[x]);
            }
        }
        return(processTerm(eingabe));
    }

    private LinkedList <String> processTerm(LinkedList<String> eingabe)
    {
        LinkedList<String> result = new LinkedList<String>();
        String first = eingabe.removeFirst();
        if (first.equals("("))
        {
            int x = 1;
            LinkedList <String> part = new LinkedList<String>();
            while (x > 0)
            {
                String pt = eingabe.removeFirst();
                if (pt.equals("("))
                {
                    x++;
                }
                if (pt.equals(")"))
                {
                    x--;
                }
                part.add(pt);
            }
            result = processTerm(part);
        }
        else
        {
            // Hier noch weitermachen, bis ein AND oder ein OR oder das
               Ende erreicht wurde
            boolean t = true;
            while (t && (eingabe.size() > 0))
            {
                if (eingabe.getFirst().equals("AND") || eingabe.getFirst().
                    equals("OR"))
                {
                    t = false;
                }
                else
                {
                    first = first.concat( "␣" + eingabe.removeFirst());
                }
            }
            result = search(first);
        }

        // Liegen noch weitere Verschachtelungen vor, oder sind wir fertig?

        if (eingabe.size() == 0)
        {
            return (result);
        }
```

```
        // OK, eine Verschachtelung. dann also weiter im Text
        String connector = eingabe.removeFirst();

        if (connector.equals("AND"))
        {
            LinkedList<String> rest = processTerm(eingabe);
            HashMap<String,String> check = new HashMap<String,String>();

            for (String h : rest)
            {
                check.put(h, "_");
            }
            LinkedList <String> re = new LinkedList<String>();
            for (String h : result)
            {
                if (check.containsKey(h))
                {
                    re.add(h);
                }
            }
            result = re;
        }

        if (connector.equals("OR"))
        {
            result.addAll(processTerm(eingabe));
        }
        return result;
    }

    private LinkedList<String> search(String e)
    {
        LinkedList<String> result = new LinkedList<String> ();
        boolean truncated = false;
        LinkedList<String> allNames = database.getAllNames();
        System.out.println(">" + e + "<" );
        if (e.endsWith("$"))
        {
            e = e.replaceAll("\\$", "");
            truncated = true;
        }
        System.out.println(e);

        for (String name : allNames)
        {
            if (truncated)
            {
                String n1 = name.toLowerCase();
                String e1 = e.toLowerCase();
                if (n1.startsWith(e1))
                {
                    result.add(name);
                }
            }
            else
            {
```

```
                if (name.equalsIgnoreCase(e))
                {
                    result.add(name);
                }
            }
        }

        HashMap<String, String> sdl = new HashMap<String,String>();
        for (String name : result)
        {
            sdl.put(database.getNameByName(name), "␣");
        }
        result = new LinkedList<String>();
        result.addAll(sdl.keySet());
        return(result);
    }
}
```

A.6 Paket SWDvisualization – Visualisierung

Dieses Paket dient der Erzeugung der Baumstrukturen, anhand derer die Schlagworteinträge grafisch dargestellt werden. Die Interaktion mit den dargestellten Strukturen wird über die Klassen des Paketes SWDgui gehandhabt.

A.6.1 Klasse SWDvisualization.VisualizationMain

Die Klasse `SWDvisualization.VisualizationMain` ist die Hauptklasse, die die in einer Suchanfrage gefundenen Schlagworte anhand der herschenden Unter- und Oberbegriffsrelationen visualisiert.

```
package SWDvisualization;

import org.jgraph.*;
import org.jgraph.graph.*;
import javax.swing.*;
import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.HashMap;
import java.util.Iterator;

import SWDdata.Database;
import SWDdata.Relation;
import SWDio.SystemLogger;

public class VisualizationMain
{
    /* SWDvis: Diese Klasse konstruiert aus den uebergebenen Entitaeten und
       Relationen eine
       Visualisierung, die mittels der print()-Methode uebergeben werden
          kann. */

    /* Basisidee: Die Entitaeten und Relationen werden erst in einen
       grossen Beutel gepackt,
       dann werden die Entitaeten anhand der Unterbegriff/Oberbegriff-
          Relationen auf Ebenen
       verteilt. Aufpassen auf Sonderfaelle: Kinder auf mehr als einer
          Ebene.
       Anschliessend werden die Groessen der Rechtecke bestimmt und die
          Ebenen v.l.n.r. Feder
       aufgebaut (Sortierung?).
       Schliesslich erfolgt die Erzeugung der Pfeile zwischen den Ebenen.
          */

    /* Offene Frage: Wie werden Verwandtschaften dargestellt? Ueber Farben?
       */

    /* Problem: Da ein Kind mehrere Eltern und ein Elternteil mehrere
       Kinder
     * haben kann, wurde die ineffiziente Implementierung ueber SWDrelation
     * gewaehlt
     */
```

```
LinkedList <Relation> pcRelations ;
HashMap <String , String> toShow ; // Alle Eintraege, die noch angezeigt
    werden sollen
HashMap <String ,String> isChild ;
HashMap <String ,String> colorOf; // Farbe!
int breaksize = 400; // DEFAULT, Minimum-Anzahl von Pixeln, nach denen
   umgebrochen wird
Database database = null;

public VisualizationMain ()
{
    pcRelations = new LinkedList<Relation>();
    toShow = new HashMap<String , String>();
    isChild = new HashMap<String ,String>();
    colorOf = new HashMap<String ,String>();
}

public void setDatabase (Database d)
{
    database = d;
}

public void setBreaksize(int i)
{
    breaksize = i;
}

public JScrollPane getGraphScrollPane()
{
    cleanup();

    // Definition der Basiselemente
    GraphModel model = new DefaultGraphModel();
    GraphLayoutCache view = new GraphLayoutCache(model, new
        DefaultCellViewFactory());
    JGraph graph = new JGraph(model, view);
    graph.setAutoResizeGraph(true);

    // Definieren der Ebenen, Kaestchen, Pfeile
    LinkedList <LinkedList<String>> layers = getLayers();
    //layers = splitLayers(layers); // Methode zum Aufteilen zu langer
        Ebenen
    // Name -> Nummer in dem cell-Array; Wichtig fuer Kantenbestimmung:
    HashMap <String ,Integer> NameToPos = getNumOrder(layers);
    LinkedList <DefaultGraphCell> l_cells = createRectangles(layers);
    /* Bestimmen der Anzahl der Kanten: Es werden nur die Kanten
        angezeigt, die
     * auch eine Entsprechung in den dargestellten Kaestchen haben.
     * Dies kann einfach ueber das NameToPos-Hash festgestellt werden
     * Die Anzahl der Kanten ist notwendig fuer die Bestimmung der
         Groesse des
     * cell-Arrays.
     */
    int i_edgeCount = 0;

    for ( Relation i : pcRelations )
```

```
        {
            String s_child = i.getChild();
            String s_parent = i.getParent();
            if (NameToPos.containsKey(s_child) && NameToPos.containsKey(
                s_parent))
            {
                i_edgeCount++;
            }
        }

        /* Definieren des einzutragenden DefaultGraphCell-Arrays */

        DefaultGraphCell cells[] = new DefaultGraphCell[l_cells.size() +
            i_edgeCount];

        Iterator <DefaultGraphCell> iter = l_cells.iterator();
        int c = 0;
        while (iter.hasNext())
        {
            cells[c] = iter.next();
            c++;
        }

        int k = l_cells.size();
        Iterator <Relation> iters = pcRelations.iterator();
        while (iters.hasNext())
        {
            Relation i = iters.next();
            String s_child = i.getChild();
            String s_parent = i.getParent();
            if (NameToPos.containsKey(s_child) && NameToPos.containsKey(
                s_parent))
            {
                int i_source = NameToPos.get(s_parent);
                int i_target = NameToPos.get(s_child);

                DefaultEdge edge = new DefaultEdge();
                edge.setSource(cells[i_source].getChildAt(0));
                edge.setTarget(cells[i_target].getChildAt(0));
                cells[k] = edge;
                int arrow = GraphConstants.ARROW_CLASSIC;
                GraphConstants.setLineEnd(edge.getAttributes(), arrow);
                GraphConstants.setEndFill(edge.getAttributes(), true);
                k++;
            }
        }

        graph.getGraphLayoutCache().insert(cells);

        JScrollPane scroller = new JScrollPane(graph);
        scroller.setWheelScrollingEnabled(true);
        return(scroller);
    }

    // Methode zum Aufteilen der Ebenen
    private LinkedList <LinkedList<String>> splitLayers (LinkedList<
        LinkedList<String>> ebenen)
```

```
{
    LinkedList<LinkedList<String>> result = new LinkedList<LinkedList<
        String>>();

    // Zuerst jede Ebene nach der Anzahl der Kinder in der Ebene
    // sortieren : Je mehr Kinder, desto tiefer
    HashMap <String,Integer> nameToChildren = new HashMap<String,
        Integer>();
    for (String name : toShow.keySet())
    {
        nameToChildren.put(name, new Integer(0));
    }

    int max = 0;
    for (Relation r : pcRelations)
    {
        String child = r.getChild();
        String parent = r.getParent();
        if (toShow.containsKey(child) && toShow.containsKey(parent))
        {
            nameToChildren.put(parent, (nameToChildren.get(parent)+1));
            if (max < nameToChildren.get(parent))
            {
                max++;
            }
        }
    }

    // Sortieren
    LinkedList<LinkedList<String>> res = new LinkedList<LinkedList<
        String>>();
    for (LinkedList<String> ebene : ebenen)
    {
        ArrayList<LinkedList<String>> w = new ArrayList<LinkedList<
            String>>(max);
        for ( int t = 0; t < (max+1); t++)
        {
            w.add(new LinkedList<String>());
        }

        for (String name : ebene)
        {
            LinkedList<String> temp = w.get(nameToChildren.get(name));
            temp.add(name);
            w.set(nameToChildren.get(name),temp);
        }

        LinkedList<String >o = new LinkedList<String>();
        for (LinkedList<String> d : w)
        {
            for (String name : d)
            {
                o.add(name);
            }
        }
        res.add(o);
    }
```

```
        ebenen = res;

        // Umbrechen
        for (LinkedList<String> ebene : ebenen)
        {
            LinkedList<String> i = new LinkedList<String>();
            int space = 20;
            for (String entry : ebene)
            {
                // 10 Pixel je Buchstabe, 20 linken Rand, 20 zwischen den
                    Rechtecken
                space += (entry.length() * 10 + 20);
                i.add(entry);

                //if (space > breaksize)
                //{
                //  result.add(i);
                //  i = new LinkedList<String>();
                //  space = 20;
                //}
            }
            result.add(i);
        }

        return(result);
    }

    private LinkedList <DefaultGraphCell> createRectangles(LinkedList <
        LinkedList<String>> layers)
    {
        // Und jetzt fuer jedes Layer die Kaestchen bestimmen
        /* Annahmen:
         * 10 Pixel je Buchstabe
         * Abstand von 40 Einheiten zwischen den Ebenen
         * Abstand von 20 Einheiten zwischen den Rechtecken
         * Abstand von 20 Einheiten zum linken Rand
         */

        int offset = 20;
        LinkedList <DefaultGraphCell> result = new LinkedList <
            DefaultGraphCell>();

        // Iteration ueber den Ebenen
        Iterator <LinkedList<String>>iter = layers.iterator();
        while (iter.hasNext())
        {
            LinkedList <String> layer = iter.next();
            LinkedList <DefaultGraphCell> l_temp = (calculateRecLayer(layer
                ,offset));
            Iterator<DefaultGraphCell> c = l_temp.iterator();
            while (c.hasNext())
            {
                result.add(c.next());
            }
            offset += 60;
        }
```

```
        return(result);
    }

    // Methode, die aus den einzelnen Ebenen die Rechtecke erzeugt
    private LinkedList <DefaultGraphCell> calculateRecLayer(LinkedList <
        String> layer, int offset)
    {
        LinkedList <DefaultGraphCell> result = new LinkedList<
            DefaultGraphCell>();
        int distance = 20;

        // An dieser Stelle werden die Kinder und Elterncounts in die Namen
            eingefuegt
        if (database != null)
        {
            LinkedList<String> t = new LinkedList<String>();
            for (String content : layer)
            {
                content = content + ("␣(" + database.getParentCount(content
                    ) +
                        "/" + database.getChildrenCount(content) + ")" );
                t.add(content);
            }
            layer = t;
        }

        for (String content : layer)
        {
            DefaultGraphCell cell = new DefaultGraphCell(content);
            GraphConstants.setBounds(cell.getAttributes(),
                    new Rectangle2D.Double(distance,offset,(content.length
                        ()*10),20));
            // Farbe setzen
            if ( colorOf.get(content.replaceFirst("␣\\(.*\\)", "")) == null
                )
            {
                GraphConstants.setGradientColor( cell.getAttributes(),
                    Color.orange );
            }
            else
            {
                if (colorOf.get(content.replaceFirst("␣\\(.*\\)", "")).
                    equals("red"))
                {
                    GraphConstants.setGradientColor( cell.getAttributes(),
                        Color.red );
                }
                else
                {
                    GraphConstants.setGradientColor( cell.getAttributes(),
                        Color.green );
                }
            }
            GraphConstants.setOpaque(cell.getAttributes(), true);
            DefaultPort port = new DefaultPort();
            cell.add(port);
            result.add(cell);
```

```
            distance += (content.length()*10);
            distance += 20;
        }
        return(result);
    }

    /* Methode, die aus den eingespeicherten Daten die Ebenenstruktur
        erzeugt.
     * Sie liefert eine LinkedList von LinkedList zurueck, die die
        jeweiligen Ebenen
     * repraesentiert
     *
     * globale Datenstrukturen, die in dieser Methode verwendet werden:
     * - toShow, Hashstruktur, die alle anzuzeigenden Elemente als Key
        besitzt
     * - isChild
     * - pcRelations
     */
    private LinkedList <LinkedList<String>> getLayers()
    {
        // Datenstrukturen festlegen
        LinkedList <LinkedList<String>>result = new LinkedList<LinkedList<
            String>>();

        // Eliminierung der autozyklischen Elemente
        LinkedList <Relation> temp = pcRelations;
        pcRelations = new LinkedList<Relation>();
        LinkedList <Relation> autoZ = new LinkedList<Relation>();
        for (Relation rel : temp)
        {
            if ((rel.getChild()).equals(rel.getParent()))
            {
                autoZ.add(rel);
                SystemLogger.warn("SWD-Fehler: Element " + rel.getParent()
                    + " ist autozyklisch!");
            }
            else
            {
                pcRelations.add(rel);
            }
        }
        temp = null;

        /* Trennung aller Graphen in Teilgraphen durchfuehren.
         * Die unverbundenen Elemente kommen in die erste Reihe. */
        LinkedList <LinkedList<String>>subgraphs = getSubgraphs();

        /* Rekursiver Aufruf der originalen Methode zum Graphenaufbau.
         */
        for (LinkedList<String> subgraph : subgraphs)
        {
            if (subgraph.size() > 0)
            {
                result.addAll(align(subgraph));
            }
            else
            {
```

```
                // Rettung der leeren Graphen zur Untergliederung
                result.add(subgraph);
            }
        }

        // "Rettung" der autozyklischen Elemente
        for (Relation x : autoZ)
        {
            pcRelations.add(x);
        }

        return(result);
    }

    private LinkedList<LinkedList<String>> align (LinkedList<String> part)
    {
        if (part.size() == 1)
        {
            LinkedList<LinkedList<String>> result = new LinkedList<
                LinkedList<String>>();
            result.add(part);
            return result;
        };

        Sugiyama t = new Sugiyama(breaksize);
        return(t.performSugiyamaLayering(part, pcRelations, isChild));
    }

    /*
     * Methode liefert aus den angegebenen globalen Variablen die
     * einzelnen (unverbundenen) Subgraphen, wobei alle komplett
        unverbundenen
     * Knoten in die erste Reihe kommen.
     * - toShow, Hashstruktur, die alle anzuzeigenden Elemente als Key
        besitzt
     * - isChild
     * - pcRelations
     * Diese Methode legt weiterhin die Farben fuer die jeweiligen
        Rechtecke fest.
     */

    private LinkedList <LinkedList<String>> getSubgraphs()
    {
        LinkedList<LinkedList<String>> result = new LinkedList<LinkedList<
            String>>();

        LinkedList<HashMap<String,String>> temp = new LinkedList<HashMap<
            String,String>>();
        for (String name : toShow.keySet())
        {
            HashMap<String,String> tmp = new HashMap<String,String>();
            tmp.put(name, "␣");
            temp.add(tmp);
        }

        for (Relation relation : pcRelations)
        {
```

```
        String child = relation.getChild();
        String parent = relation.getParent();
        if (toShow.containsKey(child) && toShow.containsKey(parent))
        {
            int i = 0;
            int r1 = 0;
            int r2 = 0;
            int a = 0;
            for (HashMap<String,String> map : temp)
            {
                if (map.containsKey(child))
                {
                    r1 = i;
                    a++;
                }
                else
                {
                    if (map.containsKey(parent))
                    {
                        r2 = i;
                        a++;
                    }
                }
                i++;
            }

            if (a > 1)
            {
                HashMap<String,String> t1 = temp.get(r1);
                HashMap<String,String> t2 = temp.get(r2);
                if (r2 < r1)
                {
                    temp.remove(r1);
                    temp.remove(r2);
                }
                else
                {
                    temp.remove(r2);
                    temp.remove(r1);
                }

                for (String t : t2.keySet())
                {
                    t1.put(t, "␣");
                }

                temp.add(t1);
            }
        }
    }

    // Einsammeln der einzelnen Elemente
    // Spezielle Beachtung der autozyklischen Elemente
    LinkedList<String> one = new LinkedList<String>();

    for (HashMap<String,String> map : temp)
    {
```

```
            if (map.size() == 1)
            {
                for (String z : map.keySet())
                {
                    one.add(z); // Sollte nur eines sein
                }
            }
        }

        if (one.size() > 0)
        {
            result.add(one);
            // Leerzeile einfuegen
            result.add(new LinkedList<String>());
        }

        // Einsammeln der anderen Elemente
        for (HashMap<String,String> map : temp)
        {
            LinkedList<String> more = new LinkedList<String>();
            if (map.size() > 1)
            {
                for (String z : map.keySet())
                {
                    more.add(z); // Sollten mehrere sein
                }
                result.add(more);
                // Leerzeile einfuegen
                result.add(new LinkedList<String>());
                more = new LinkedList<String>();
            }
        }
        return(result);
    }

    /* Sortiert Ebenendarstellung der Strings um, so dass die Nummer in dem
     * cell-Array direkt bekannt ist */
    private HashMap <String,Integer> getNumOrder(LinkedList <LinkedList<
        String>> layers)
    {
        HashMap <String,Integer> result = new HashMap<String,Integer>();
        int i = 0;
        Iterator <LinkedList<String>> iter = layers.iterator();
        while (iter.hasNext())
        {
            LinkedList <String> z = iter.next();
            Iterator <String>iter2 =  z.iterator();
            while (iter2.hasNext())
            {
                String name = iter2.next();
                result.put(name, i);
                i++;
            }
        }
        return(result);
    }
```

```
// Diese Methode raeumt ein letztes Mal vor der Anzeige die Relations
    auf
private void cleanup()
{
    HashMap <String,Boolean> childParent = new HashMap<String,Boolean
        >();
    LinkedList<Relation> temp = pcRelations;
    pcRelations = new LinkedList<Relation>();

    for (Relation element : temp)
    {
        String child =  element.getChild();
        String parent = element.getParent();
        if(!(childParent.containsKey(child + "_!_" + parent)))
        {
            childParent.put(child + "_!_" + parent,true);
            pcRelations.add(element);
        }
    }
}

/* ### Oeffentliche Methoden ##############################################
    */

/* Methode zum Setzen von Relationen.
 * Eine Eintragung mit dieser Methode fuehrt nicht zu einer
automatischen Darstellung der Schlagwoerter! */
public void setPCRelation (String a, String b)
{
    Relation entry = new Relation(a,b);
    isChild.put(b, "␣");
    pcRelations.add(entry);
}

/* Hier werden die darzustellenden Schlagwoerter definiert
 * Wahlweise mit Angabe einer Farbe
 * Die einmalige Setzung von Rot kann nicht ueberschrieben werden
 */
public void setEntry(String a)
{
    toShow.put(a,"␣");
}
public void setEntry(String a, String color)
{
    if (!(color == null))
    {
        toShow.put(a,"␣");
        if ((colorOf.get(a) == null))
        {
            colorOf.put(a, color);
        }
        if (!(colorOf.get(a).equals("red")))
        {
            colorOf.put(a, color);
        }
    }
}
```

```
    // Methode zum Anzeigen des Startup-Graphen "Hallo User"
    public static JScrollPane helloSWD()
    {
        GraphModel model = new DefaultGraphModel();
        GraphLayoutCache view = new GraphLayoutCache(model, new
            DefaultCellViewFactory());

        JGraph graph = new JGraph(model, view);
        graph.setAutoResizeGraph(true);

        DefaultGraphCell[] cells = new DefaultGraphCell[3];
        cells[0] = new DefaultGraphCell(new String("Hello"));
        GraphConstants.setBounds(cells[0].getAttributes(), new Rectangle2D.
            Double(20,20,40,20));
        GraphConstants.setGradientColor( cells[0].getAttributes(), Color.
            orange);
        GraphConstants.setOpaque(cells[0].getAttributes(), true);
        DefaultPort port0 = new DefaultPort();
        cells[0].add(port0);
        cells[1] = new DefaultGraphCell(new String("User"));
        GraphConstants.setBounds(cells[1].getAttributes(), new Rectangle2D.
            Double(140,140,40,20));
        GraphConstants.setGradientColor(cells[1].getAttributes(), Color.red
            );
        GraphConstants.setOpaque(cells[1].getAttributes(), true);
        DefaultPort port1 = new DefaultPort();
        cells[1].add(port1);
        DefaultEdge edge = new DefaultEdge();
        edge.setSource(cells[0].getChildAt(0));
        edge.setTarget(cells[1].getChildAt(0));
        cells[2] = edge;

        int arrow = GraphConstants.ARROW_CLASSIC;
        GraphConstants.setLineEnd(edge.getAttributes(), arrow);
        GraphConstants.setEndFill(edge.getAttributes(), true);
        graph.getGraphLayoutCache().insert(cells);

        JScrollPane scroller = new JScrollPane(graph);

        return(scroller);
    }
}
```

A.6.2 Klasse SWDvisualization.Sugiyama

Die Klasse `SWDvisualization.Sugiyama` implementiert den Sugiyama-Algorithmus zur Darstellung gerichter Graphen.

```
package SWDvisualization;

import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.TreeSet;

import SWDdata.Relation;
import SWDio.SystemLogger;

/* Revidierte Version von align nach Sugiyama,
basierend auf dem Layering-Algorithmus von Coffman und Grayham (72),
Greedy Cycle Removal und Median Crossing Reduction
*/

public class Sugiyama
{
    int breaksize = 750;
    int currentSize = 20;

    /* Notiz: Es muessen bei der Einfuehrung der Dummy-Nodes
     * und dem Cycle-Removal keine Aenderungen an den pcRelations
         rueckgaengig gemacht werden,
     * da die originalen Informationen in dem Hauptprogramm bleiben.
     * Allerdings muessen alle Dummy-Nodes wieder aus den Listen entfernt
         werden!
     */

    LinkedList <Relation> pcRelations = new LinkedList<Relation>();
    LinkedList<String> v = null;
    HashMap <String,String> isChild = null;
    LinkedList <LinkedList<String>>layers = new LinkedList<LinkedList<
        String>>();

    // Konstruktor
    public Sugiyama(int breakSize)
    {
        breaksize = breakSize;
        if (breaksize <= 20)
        {
            breaksize = 100;
        }
    };

    public LinkedList<LinkedList<String>> performSugiyamaLayering (
        LinkedList<String> vinput,
            LinkedList <Relation> rt,
            HashMap <String,String> isChildInput)
    {
        v = vinput;
        isChild = isChildInput;
```

```
// Backup von pcRelations und
// Reduktion der pcRelations auf die hier relevanten
{
    HashMap<String,Boolean> g = new HashMap<String,Boolean>();
    for (String vs : vinput)
    {
        g.put(vs, false);
    }
    for (Relation rel : rt)
    {
        if ( g.containsKey(rel.getChild()) || g.containsKey(rel.
            getParent()))
        {
            pcRelations.add(rel);
        }
    }
}
eliminateCircles();

int nodecount = v.size();

//Phase 0: Abfangen der trivialen Faelle

if (v.size() == 0) { return(layers); };
if (v.size() == 1)
{
    layers.add(v);
    return(layers);
}
if (v.size() == 2)
{
    LinkedList<String> a = new LinkedList<String>();
    LinkedList<String> b = new LinkedList<String>();

    for (Relation rel : rt)
    {
        if ((rel.getChild().equals(v.get(0))) && rel.getParent().
            equals(v.get(1)))
        {
            a.add(v.get(1));
            b.add(v.get(0));
            layers.add(a);
            layers.add(b);
            return layers;
        }
        if ((rel.getChild().equals(v.get(1))) && rel.getParent().
            equals(v.get(0)))
        {
            a.add(v.get(0));
            b.add(v.get(1));
            layers.add(a);
            layers.add(b);
            return layers;
        }
    }
}
```

```
        // Nicht trivial? Coffman Graham anwenden!

        coffmanGraham();
        insertDummies();

        medianCrossingReduction();
        removeDummies();

        // Abschliessende Sicherheitstests
        int nodecount2 = 0;
        for (LinkedList<String> r : layers)
        {
            nodecount2 += r.size();
        }

        if (nodecount != nodecount2)
        {
            SystemLogger.error("Element count mismatch error in
                CoffmanGraham");
        }
        return(layers);
    }

    // Methode zum Finden von Zirkularitaeten
    private void eliminateCircles()
    {
        /* Eliminieren der autozyklischen Elemente
         * wurde eigentlich schon durchgefuehrt, erfolgt aber in linearer
         * Komplexitaet und wird daher vor Vollstaendigkeit noch einmal
             getan
         */

        LinkedList<Relation> temp1 = pcRelations;
        pcRelations = new LinkedList<Relation>();

        for (Relation rel : temp1)
        {
            if (!(rel.getChild().equals(rel.getParent())))
            {
                pcRelations.add(rel);
            }
        }
        temp1 = pcRelations;
        pcRelations = new LinkedList<Relation>();

        /* Eliminieren der Zweikreise
         * Einfache Heuristik: Das Element mit dem kuerzeren Namen ist oben
             !
         */

        HashMap <String, Boolean> temp2 = new HashMap<String,Boolean>();
        HashMap <String, String > toDel = new HashMap<String,String>();

        for (Relation rel : temp1)
        {
            String parent = rel.getParent();
            String child = rel.getChild();
```

```
            if (! ((temp2.containsKey(parent + "_!_" + child))
                    || temp2.containsKey(child + "_!_" + parent)))
            {
                temp2.put(parent + "_!_" + child,true);
                temp2.put(child + "_!_" + parent,true);
            }
            else
            {
                if (parent.length() > child.length())
                {
                    toDel.put(parent,child);
                }
                else
                {
                    toDel.put(child,parent);
                }
            }
        }

        for (Relation rel : temp1)
        {
            if (!(toDel.containsKey(rel.getParent())
                    && (toDel.get(rel.getParent()).equals(rel.getChild()))))
                    )
            {
                pcRelations.add(rel);
            }
            else
            {
                SystemLogger.warn("SWD-Fehler: Zweikreis gefunden! " +
                        rel.getChild() + " und " + rel.getParent());
            }
        }

        /* Eliminieren komplexer Kreise nach dem Greedy-Cycle-Removal-
           Algorithmus, Umdrehung der jeweiligen
         * Kanten wuerde an dieser Stelle eingefuegt werden
         */
    }

    private void coffmanGraham()
    {
        // Phase 1:
        HashMap <String,Integer> pi = new HashMap<String,Integer>();

        // Bestimmung eines Quellknotens
        boolean b = true;
        for (String element : v)
        {
            if (b)
            {
                if (! (isChild.containsKey(element)))
                {
                    pi.put(element, 1);
                    b = false;
                }
            }
```

```
}

for (int i = 2; i <= v.size(); i++)
{
    LinkedList <String> candidates = new LinkedList<String>();
    for (String element : v)
    {
        // Label wurde noch nicht vergeben, V besitzt noch kein
           Label
        if (!pi.containsKey(element))
        {
            // Alle Knoten u mit (u,v) elem E besitzen bereits ein
               Label
            boolean label = true;
            for (Relation relation: pcRelations)
            {
                if (relation.getChild().equals(element))
                {
                    if (!pi.containsKey(relation.getParent()))
                    {
                        label = false;
                    }
                }
            }

            // die Menge der Kandidaten bilden
            if (label)
            {
                candidates.add(element);
            }
        }
    }

    // die Menge der direkten Vorgaenger ist lexikographisch
       minimal
    // Vorgaenger-Mengen bilden
    HashMap <String,TreeSet<Integer>> pre = new HashMap<String,
       TreeSet<Integer>>();
    for (String element : candidates)
    {
        TreeSet<Integer> vmenge = new TreeSet<Integer>();
        for (Relation relation: pcRelations)
        {
            if (relation.getChild().equals(element))
            {
                vmenge.add(pi.get(relation.getParent()));
            }
        }
        pre.put(element, vmenge);
    }

    // Groessenvergleich
    LexicographicCompare compare = new LexicographicCompare(pre);
    TreeSet<String> sorter = new TreeSet<String>(compare);
    for (String element : candidates)
    {
        sorter.add(element);
```

```
        }
        pi.put(sorter.first(), i);
    }

    // Phase 2:
    layers.add(new LinkedList<String>());
    int k = 0;
    HashMap<String,String> u = new HashMap<String,String>();

    while (u.size() != v.size())
    {
        /* Wenn die maximale Groesse des Layers erreicht ist, ein
        neues aufmachen! */
        // if (layers.get(k).size() >= w)
        if (currentSize > breaksize)
        {
            k ++;
            layers.add(new LinkedList<String>());
            currentSize = 20;
        }

        // Sammeln moeglicher Kandidaten fuer u
        LinkedList <String>candidates = new LinkedList<String>();

        /* Finde die Kandidaten aus V\U, bei denen alle Kinder schon in
            einem
         * tieferen Layer enthalten sind
         */
        HashMap<String, Boolean> contentOfDeeperLayers = new HashMap<
            String,Boolean>();
        for (int i = 0; i < (layers.size()-1); i++)
        {
            for (String s : layers.get(i))
            {
                contentOfDeeperLayers.put(s,true);
            }
        }

        for (String element : v)
        {
            if (! u.containsKey(element))
            {
                boolean candidate = true;
                for (Relation rel : pcRelations)
                {
                    if (rel.getParent().equals(element))
                    {
                        if (!contentOfDeeperLayers.containsKey(rel.
                            getChild()))
                        {
                            candidate = false;
                        }
                    }
                }
                if (candidate)
                {
                    candidates.add(element);
```

```
            }
        }
    }

    /* Gibt es ueberhaupt einen Kandidaten?
     * Wenn nicht, neues Layer aufmachen und die Suche
     * neu beginnen!
     */
    if (candidates.size() == 0)
    {
        k ++;
        layers.add(new LinkedList<String>());
        currentSize = 20;
    }
    else
    {
        /* Jetzt aus den Kandidaten denjenigen herausfischen, bei
           dem
         * Pi maximiert ist
         */
        // Pi(cand) is maximized
        HashMap<Integer,String> f = new HashMap<Integer,String>();
        TreeSet <Integer> t = new TreeSet<Integer>();

        for (String cand : candidates)
        {
            f.put(pi.get(cand),cand);
            t.add(pi.get(cand));
        }
        String the_u = f.get(t.last());

        // Naechste Bedingung
        boolean allIncluded = true;
        HashMap <String,String> lowerLayers = new HashMap<String,
            String>();
        int z = layers.size();
        for (int i = 0; i < z; i++)
            //for (int i = 0; i < (z-1); i++)
        {
            for (String element : layers.get(i))
            {
                lowerLayers.put(element, "␣");
            }
        }

        for (Relation rel : pcRelations)
        {
            if (allIncluded)
            {
                if (rel.getParent().equals(the_u))
                {
                    if (!(lowerLayers.containsKey(rel.getChild())))
                    {
                        allIncluded = false;
                    }
                }
            }
```

```
                }

                if (allIncluded)
                {
                    layers.getLast().add(the_u);
                    currentSize += (the_u.length()*10);
                }
                else
                {
                    k ++;
                    LinkedList <String> newl = new LinkedList<String>();
                    newl.add(the_u);
                    layers.add(newl);
                    currentSize += (the_u.length()*10);
                }
                // add u to U
                u.put(the_u, "␣");
            }
        }

        // layers noch umdrehen...
        LinkedList<LinkedList<String>>result = new LinkedList<LinkedList<
            String>>();
        Iterator<LinkedList<String>> iter = layers.descendingIterator();

        while (iter.hasNext())
        {
            LinkedList<String> r = iter.next();
            result.add(r);
        }
        layers = result;
    }

    /* Diese Methode fuegt als Vorbereitung fuer den Crossing-Reduction-
        Schritt
     * Dummy-Knoten in den Graphen ein
     */

    private void insertDummies()
    {
        HashMap<String,Boolean> dontCopyBack = new HashMap<String,Boolean
            >();
        LinkedList <Relation> localRelations = new LinkedList<Relation>();
        HashMap<String,Boolean> content = new HashMap<String,Boolean>();
        int dummyCounter = 0;

        // Kopieren der Liste der pcRelations
        for (Relation rel : pcRelations)
        {
            localRelations.add(rel);
        }

        // Aufbauen der Liste aller Elemente
        for (LinkedList<String> einzel : layers)
        {
            for (String node : einzel)
            {
```

```
            content.put(node, false);
        }
    }

    // Loeschen der ersten Ebene
    LinkedList<String> f = layers.get(0);
    for (String s : f)
    {
        content.remove(s);
    }

    // Iteration ueber den Doppellayern
    for (int i = 0; i < (layers.size() - 1); i++)
    {
        // Loeschen der naechsten Ebene
        LinkedList<String> fi = layers.get(i+1);
        for (String s : fi)
        {
            content.remove(s);
        }

        // Notieren der aktuellen Ebene
        HashMap <String,Boolean> lc = new HashMap<String,Boolean>();
        f = layers.get(i);
        for (String s : f)
        {
            lc.put(s,false);
        }

        // Anlegen neuer Knoten, wenn noetig
        LinkedList<Relation> toInsert = new LinkedList<Relation>();
        for (Relation rel :localRelations)
        {
            if (lc.containsKey(rel.getParent()) && content.containsKey(
                rel.getChild()))
            {
                String dummyName = ("!_Dummy" + Integer.toString(
                    dummyCounter));
                layers.get(i+1).add(dummyName);
                dummyCounter ++;
                toInsert.add(new Relation(rel.getParent(),dummyName));
                toInsert.add(new Relation(dummyName, rel.getChild()));
                if (rel.getParent().startsWith("!_Dummy"))
                {
                    dontCopyBack.put(rel.getParent(), false);
                }
            }
        }
        localRelations.addAll(toInsert);

        // Erstellen der korrekten Relations-Liste
        LinkedList <Relation> result = new LinkedList<Relation>();
        for (Relation rel : localRelations)
        {
            if (!(dontCopyBack.containsKey((rel.getParent()))
                    && (!rel.getChild().startsWith("!_Dummy"))))
            {
```

```
                result.add(rel);
            }
        }
        pcRelations = result;
    }
}

/* Diese Methode laesst von unten nach oben eine Crossing-Reduction
   durchlaufen.
 * Dabei wird nur eine einzelne Iteration durchgefuehrt.
 */

private void medianCrossingReduction()
{
    for (int i = (layers.size() - 1); i > 0; i--)
    {
        twoLayerMedianCrossingReduction(i,i-1);
    }

    for (int i = 0; i < (layers.size() - 1); i++)
    {
        twoLayerMedianCrossingReduction(i,i+1);
    }
}

private void twoLayerMedianCrossingReduction(int i, int i2)
{
    LinkedList<String> lone = layers.get(i);
    LinkedList<String> ltosort = layers.get(i2);

    HashMap <String, Integer> nameToNumber = new HashMap<String,
        Integer>();
    for (int y = 0; y < lone.size(); y++)
    {
        nameToNumber.put(lone.get(y),y);
    }
    HashMap <String,TreeSet<Integer>> row = new HashMap<String,TreeSet<
        Integer>>();
    for (String node : ltosort)
    {
        row.put(node, new TreeSet<Integer> ());
    }

    for (Relation rel : pcRelations)
    {
        String parent = rel.getParent();
        String child = rel.getChild();
        if ((nameToNumber.containsKey(child) && row.containsKey(parent)
            ))
        {
            row.get(parent).add(nameToNumber.get(child));
        }

        if ((nameToNumber.containsKey(parent) && row.containsKey(child)
            ))
        {
            row.get(child).add(nameToNumber.get(parent));
```

```
        }
    }

    // Im Zweifelsfall Tausenden einfuegen
    for (String name : row.keySet())
    {
        if (row.get(name).size() == 0)
        {
            TreeSet <Integer> insert = new TreeSet<Integer>();
            insert.add(1000);
            row.put(name, insert);
        }
    }

    // Median berechnen
    HashMap <Double, String> medianToName = new HashMap<Double, String>()
        ;
    HashMap<String, Boolean> h_values = new HashMap<String, Boolean>();
    for (String name : row.keySet())
    {
        TreeSet<Integer> values = row.get(name);
        double med = 0.5 * ((double) values.size());
        med += 0.5;

        int last = 0;
        int h = 0;
        boolean cont = true;
        for ( Integer ui : values )
        {
            if (cont)
            {
                h++;
                if (med == (double) h)
                {
                    double value = (double) ui;
                    while (h_values.containsKey(Double.toString(value))
                        )
                    {
                        value += 0.001;
                    }
                    medianToName.put(value, name);
                    h_values.put(Double.toString(value),false);
                    cont = false;
                }
                if (med < (double) h)
                {
                    double value = ((double) ui + (double) last) / 2;
                    while (h_values.containsKey(Double.toString(value))
                        )
                    {
                        value += 0.001;
                    }
                    medianToName.put(value,name);
                    h_values.put(Double.toString(value),false);
                    cont = false;
                }
                last = ui;
```

```
                }
            }
        }

        LinkedList<String> result = new LinkedList<String>();
        TreeSet<Double> sorter = new TreeSet<Double>();
        sorter.addAll(medianToName.keySet());
        for (double d : sorter)
        {
            result.add(medianToName.get(d));
        }

        layers.set(i2, result);
    }

    /* Diese Methode fuegt als Vorbereitung fuer den Crossing-Reduction-
       Schritt
     * Dummy-Knoten in den Graphen ein
     */

    private void removeDummies()
    {
        LinkedList <LinkedList<String>>result = new LinkedList<LinkedList<
            String>>();

        for (LinkedList<String> einzel : layers)
        {
            LinkedList <String> t = new LinkedList<String>();
            for (String node : einzel)
            {
                if (!node.startsWith("!_Dummy"))
                {
                    t.add(node);
                }
            }
            result.add(t);
        }
        layers = result;

        // Loeschen der entsprechenden Eintraege aus den Relations
        LinkedList <Relation> result2 = new LinkedList<Relation>();
        for (Relation rel : pcRelations)
        {
            if (!(rel.getParent().startsWith("!_Dummy") || (rel.getChild().
                startsWith("!_Dummy"))))
            {
                result2.add(rel);
            }
        }
        pcRelations = result2;
    }
}
```

A.6.3 Klasse SWDvisualization.LexicographicCompare

Die Klasse `LexicographicCompare` liefert eine spezielle Vergleichsoperation, die für den Coffman-Graham-Algorithmus benötigt wird.

```
package SWDvisualization;

import java.util.Comparator;
import java.util.HashMap;
import java.util.TreeSet;

public class LexicographicCompare implements Comparator {

    HashMap <String, TreeSet<Integer>> pre;

    public LexicographicCompare(HashMap <String, TreeSet<Integer>> p)
    {
        pre = p;
    }

    @Override
    public int compare(Object o, Object t)
    {
        TreeSet <Integer> one = pre.get((String) o);
        TreeSet <Integer> two = pre.get((String) t);

        /* a negative integer, zero, or a positive integer as the first
           argument
          is less than, equal to, or greater than the second. */

        if (one.size() == 0)
        {
            if (two.size() == 0)
            {
                return 0;
            }
            return -1;
        }

        if (two.size() == 0)
        {
            return 1;
        }

        if (one.last() < two.last()) {return -1;};
        if (two.last() < one.last()) {return  1;};

        // der harte Fall, eine Rekursion

        // Maximum erhalten
        int max = one.last();

        // Neues pre aufbauen
        HashMap <String, TreeSet<Integer>> np = new HashMap <String, TreeSet<
            Integer>>();

        TreeSet<Integer> orev = new TreeSet<Integer>();
```

```
        TreeSet<Integer> trev = new TreeSet<Integer>();

        for (int i : one)
        {
            if (i != max)
            {
                orev.add(i);
            }
        }

        for (int i : two)
        {
            if (i != max)
            {
                trev.add(i);
            }
        }

        np.put((String) o, orev);
        np.put((String) t, trev);

        LexicographicCompare compare = new LexicographicCompare(np);
        return(compare.compare(o, t));
    }
}
```

A.7 Paket SWDio – allgemeine Input/Outputklassen

Dieses Paket dient der Ausgabe von Systemnachrichten.

A.7.1 Klasse SWDio.SystemLogger

Die Klasse `SWDio.SystemLogger`. Das globale Logging-Modul, über das alle Systemnachrichten laufen.

```
package SWDio;

import java.io.*;
import java.util.concurrent.locks.ReentrantLock;

import javax.swing.JTextArea;

public class SystemLogger
{
    // Variablen
    private static ReentrantLock lock = new ReentrantLock();
    private static String outfile = "SWDlog.log";
    private static JTextArea outWindow = null;

    // Standard-Konstruktor
    // public SWDlogger () {};

    // Methoden
    public static void setOutfile(String file) { outfile = file; };

    public static void setOutWindow(JTextArea a)     { outWindow = a; };

    public static void info(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Info: " + message );
            System.out.println( "Info: " + message);
            out.newLine();
            out.close();

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
            System.err.println("Logfile IO-Fehler!");
            System.err.println(e);
            System.exit(1);
        }

        if (! (outWindow == null))
        {
```

```
            outWindow.setCaretPosition( outWindow.getDocument().getLength()
                );
            outWindow.append(message + "\n");
        }
    }

    public static void warn(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Warnung: " + message );
            out.newLine();
            out.close();

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
            System.err.println("Logfile IO-Fehler!");
            System.exit(1);
        }

        if (! (outWindow == null))
        {
            outWindow.setCaretPosition( outWindow.getDocument().getLength()
                );
            outWindow.append("Warnung: " + message + "\n");
        }
    }

    public static void error(String message)
    {
        try
        {
            lock.lock();

            BufferedWriter out = new BufferedWriter(new FileWriter(outfile,
                true));
            out.write( "Error: " + message );
            System.out.println( "Error: " + message );
            out.newLine();
            out.close();
            System.exit(1);

            lock.unlock();
        }
        catch (java.io.IOException e)
        {
            System.err.println("Logfile IO-Fehler!");
            System.exit(1);
        }
    }
}
```