Endbericht

ChemBioSpace Explorer

PG 504

Impressum

Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Algorithm Engineering (LS11)
Otto-Hahn-Str. 14
44227 Dortmund

Betreuer

Prof. Petra Mutzel, Karsten Klein,
Stefan Wetzel (FB Chemie LS 4)

Mitglieder der Projektgruppe 504

Adalbert Gorecki, Anke Arndt, Arbia Ben Ahmed,
Andre Wiesniewski, Cengizhan Yücel, Gebhard Schrader,
Henning Wagner, Michael Rex, Nils Kriege,
Philipp Büderbender, Sergej Rakov, Vanessa Bembenek.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Vorwort

Der vorliegende Bericht dokumentiert die Arbeit und die Ergebnisse der einjährigen Projektgrupppe 504, im weiteren Verlauf mit PG504 abgekürzt.

Projektgruppen sind ein fester Bestandteil des Studiengangs Informatik an der Universität Dortmund und werden schon seit 1972 durchgeführt. Eine PG besteht in der Regel aus 12 Studierenden und soll einen Vorgeschmack geben, auf das, was einen Informatiker im späteren Berufsleben erwartet. Sie sollen die Organisation und die Umsetzung eines grösseren Software-Projekts über einen längeren Zeitraum üben. Diese Aufgabe hat die PG504 voll und ganz erfüllt. Über das gesamte Jahr hin war die Arbeit in und um die PG504 geprägt von Motivation, Produktivität und einem angenehmen Arbeitsklima, was nicht zu letzt auch der interessanten Aufgabenstellung zu verdanken ist.

Die Verfasser bedanken sich beim Lehrstuhl 11, insbesondere bei den Betreuern Karsten Klein, Stefan Wetzel und Professorin Petra Mutzel für den reibungslosen Ablauf und die hervorragende Betreuung der Projektarbeit.

2 Projektplanung

2.1 Allgemeine Ziele und Struktur der PG

Die so genannte „Projektgruppe“ (PG) ist eine Wahlpflichtveranstaltung des Studienganges Informatik an der Universität Dortmund in der 8 bis 12 Studierende und zwei Betreuer über einen Zeitraum von zwei Semestern an einem umfangreichen forschungsnahen Thema arbeiten. Themen werden im Vorfeld in einer gemeinsamen Veranstaltung vorgestellt. Interessenten können dabei die Voraussetzungen für die Teilnahme klären und eine Projektgruppe belegen. Ziele der Projektgruppenarbeit sind dabei die Vertiefung fachlicher Kenntnisse und das Einüben nichtfachlicher Kompetenzen wie Präsentationstechniken, Teamarbeit, Projektmanagement, das Einarbeiten in neue Problemkreise und die selbstständige Bewältigung umfangreicher Aufgaben.

Die PG504 setzte sich aus 12 teilnehmenden Studenten, sowie den Betreuern zusammen. Bei der Auswahl wurde berücksichtigt, dass die Teilnehmer die formalen und inhaltlichen Voraussetzungen erfüllten, um das Projekt bearbeiten zu können.

In der vorlesungsfreien Zeit vor Beginn des ersten Semesters arbeiteten sich die Studenten zunächst einzeln in bestimmte Bereiche der Thematik ein, indem jeder für ein von den Betreuern zugewiesenes Thema einen Vortrag vorbereitete. Zu Beginn des ersten Semesters wurden die Themen dann in einer Seminarphase der gesamten Gruppe präsentiert. In der anschliessenden Gesamtplanung des Projektes wurden die zu bearbeitenden Aufgaben festgelegt und eine Zuordnung in Kleingruppen vorgenommen. Auf die Aufteilung der Gruppe, ihre Aufgabenbereiche und Organisation wird später detailierter eingegangen. Während der zwei Semester fand einmal pro Woche eine teilgruppenübergreifende Besprechung statt. Dort stellte jeder Student den aktuellen Stand seiner Arbeit vor. Die Betreuer der Projektgruppe 504 besprachen Ziele und Probleme mit den Teilnehmern und überprüften, ob vorherige Ziele schon erreicht worden waren. Es wurde auf eine konsequente Verteilung der Aufgaben und Verantwortlichkeiten innerhalb der Gruppe geachtet und beachtet, dass alle Teilnehmer nach Möglichkeit in allen auftretenden Tätigkeitsprofilen (z.B. Programmierung, Dokumentation, Berichtserstellung, Arbeitsorganisation) herangezogen wurden.

Am Ende des ersten Semesters erstellten die Teilnehmer einen Zwischenbericht, um den Stand des Projekts festzuhalten und die Planung für das zweite Semester anzupassen.

Die Ergebnisse der Projektgruppe, die Darstellung des bearbeiteten Problems, der zeitliche Aufwand und die eingeschlagenen Lösungswege sind in diesem Abschlussbericht zusammengefasst.

2.2 Das Projekt

2.2.1 Motivation für CBSE

Aufgabe der PG504 war die Erstellung einer Applikation zur Visualisierung von Daten aus der Cheminformatik. Diese sollte Daten aus der Chemie verarbeiten und bei der Wirkstoffsuche eingesetzt werden, bei der Substanzen auf ihre biologische Wirksamkeit für ein bestimmtes Problem untersucht werden, wie etwa die Aktivierung eines Proteins. Diese Substanzen können dann beispielsweise als Grundsubstanzen von Arzneimitteln genutzt werden. Da es eine sehr große Zahl von testbaren Substanzen gibt, aber nur wenige davon wirklich relevant sind, muss eine Vorauswahl getroffen werden. Dies schränkt den Aufwand für die Synthetisierung der Substanzen und den Test im Labor ein. Für diesen Zweck wurde 2005 in der Gruppe von Prof. Waldmann an der Universität Dortmund/Max-Planck-Institut für molekulare Physiologie eine Klassifikation der Substanzen hinsichtlich ihrer chemischen Struktur vorgeschlagen. Dazu analysierten die Forscher etwa 200.000 Naturstoffe und entwickelten auf Basis dieser Analyse die SCONP (Structural Classification Of Natural Products). Mit Hilfe dieses Periodensystems konnten die Wissenschaftler auch bereits eine neuartige Strukturklasse von Hemmstoffen entwickeln. Durch die Klassifikation nach der Struktur ergibt sich eine Baumhierarchie aus Verwandtschaftsbeziehungen, die aus dem Enthaltensein von Teilstrukturen entstehen - enthält eine Struktur eine kleinere Teilstruktur, so ist sie ein Nachfahre von ihr. Diese Baumhierarchie läßt sich als Diagramm darstellen, welches aber nur einen sehr kleinen Teil der möglichen Strukturen umfassen kann, ohne unübersichtlich zu werden. Man benötigt also eine Navigation in den Daten, um von einer Datenauswahl zur nächsten zu gelangen. Diese Aufgabe sollte der CBSE erfüllen.

2.2.2 Anforderungen an den CBSE

Stefan Wetzel vom Fachbereich Chemie und vom Max-Planck Institut für Molekulare Physiologie fungierte hier als Kunde und stellte zu Beginn der Entwicklungsphase einen Katalog von Anforderungen vor, die das endgültige Programm erfüllen sollte. Dieser beinhaltete folgende Punkte:

Darstellung der Datenstruktur:

- **Gerüst Ebene:** SMILES, Gerüststruktur, gemittelte Eigenschaften, sowie Kinder jedes Scaffolds anzeigen.
- **Molekül Ebene:** SMILES, Gerüststruktur, Eigenschaften des Moleküls anzeigen.

Datenbank:

- Unterstützung von MySQL (Oracle) (Standard SQL Queries)
- Abspeichern von Nutzerdaten in der Datenbank: Historie, Compoundsets,...
- Graphischer Query Builder

Import/Export:

- Export von Bilddateien (svg, png)
- Export von Daten als SDF File
- Import von SDF / csv Dateien
- Export von user defined sets
- (Update von Daten)

Benutzerinteraktion:

- automatisches Zeichnen des Baumes
- Scaffold-Ausrichtung: Immer ähnlich (über Substruktur...) evtl. auch vorher schon
- Zoom, 2D Navigation
- Optionen (Einstellen von Thresholds, Angezeigten Daten usw.)
- Selektion der angezeigten Datenmenge (SQL Query Builder)
- Highlighting oder Selektion
- Selektion über definierte Eigenschaften (Range slider...)
- Kontextfunktionen (Mouse-over und Kontextmenü)
- Eventuell Smartssuche im Tree und Center auf Scaffold
- Undo-Funktion
- Home-Button zur Standardansicht

Kontextfunktionen:

Mouse-over:

- Anzeige von Node Eigenschaften (Zahl der Moleküle, Zahl der Kinder, Zahl der direkten Nachkommen usw.)
- u.U. Anzeige des Scaffolds für den Node in grösserer Darstellung

Kontextmenü:

- Aus-/Einblenden aller Kinder
- Einblenden der nächsten Kindergeneration
- Anordnung der Äste nach Ähnlichkeit (frei wählbares Kriterium)
- Farbverlauf nach wählbaren Eigenschaften
- Einfärbung nach Selektionskriterien
- Subtree: Erstellt Unterbaum ab diesem Node in neuem Fenster
- Zeige alle Moleküle zu einem Node in neuem Fenster

2.3 Organisation

2.3.1 Regelmässige Abläufe

Die PG504 traf sich einmal wöchentlich mit ihren Betreuern. Hier wurden die aktuellen Entwicklungen und Geschehnisse präsentiert und gruppenübergreifende Entscheidungen getroffen.

Die insgesamt 12 Teilnehmer teilten sich schon recht früh in 3 Gruppen auf, jede mit einem besonderen Fokus auf Teilprobleme der Entwicklung des CBSE:

- Visualisierung (VIS): Es sollte eine performante und ästhetische Darstellung des biochemischen Strukturraumen mitsamt praktikablen Interaktionsmöglichkeiten entwickelt werden.
- Graphisches User Interface (GUI): Nutzerverwaltung, Personalisierung, Einbettung verschiedener Funktionalitäten wie Druckdialogen, Hilfestellungen usw. wurden in dieser Gruppe erarbeitet.
- Datenbankanbindung (DB): Hier galt es, eine bereits bestehende mySQL-Datenbank in den CBSE zu integrieren und an dessen Strukturelle Bedürfnisse anzupassen.

Um den Austausch zwischen den verschiedenen Kleingruppen zu verbessern, wurden Gruppensprecher bzw. Ansprechpartner ausgewählt, die in unterschiedlichen Rhythmen wechselten.

Zusätzlich zu den wöchentlichen Treffen, bei denen die Anwesenheit für alle Teilnehmer verpflichtend war und die aktuellen Entwicklungen protokolliert wurden, trafen sich die Mitglieder der jeweiligen Kleingruppen nach Bedarf zu gesonderten und variierenden Zeiten. Dank der vielfältigen zur Verfügung gestellten Kommunikationsmöglichkeiten war ein persönliches Zusammentreffen nicht immer nötig, der Austausch via Email, Forum, Wiki, usw. erleichterte den Projektteilnehmern die Entwicklung des CBSE.

Zum Ende des ersten Semesters wurde ein Zwischenbericht erstellt, in dem die Entwicklungen am CBSE festgehalten wurden. Viele der darin verfassten Texte konnte für den Endbericht übernommen oder weitergeführt werden.

2.3.2 Interdisziplinäre Zusammenarbeit

Die Organisation und Durchführung der PG504 war eine Zusammenarbeit zwischen dem Lehrstuhl 11 „Algorithm Engineering“ der Fakultät Informatik der Universität Dortmund und dem Max-Planck-Institut für molekulare Physiologie.

Betreuer während der gesamten Projektgruppe waren Karsten Klein, Mitarbeiter am Lehrstuhl 11 „Algorithm Engineering“ und Stefan Wetzel, Doktorant am Max-Planck-Institut für molekulare Physiologie und Mitarbeiter am Lehrbereich „Chemische Biologie“ am Fachbereich Chemie der Universität Dortmund.

Karsten Klein und Stefan Wetzel arbeiteten bei der Erstellung des Programmkonzeptes mit und waren Ansprechpartner für Fragen bezüglich der Informatik bzw. der Biochemie, die während des Projektes auftraten. Ausserdem testeten die beiden regelmässig die aktuelle Version des CBSE und gaben Feedback. Stefan Wetzel betrachtete das Programm dabei besonders aus Sicht eines Chemikers und konnte so zielgruppenbezogene Eindrücke weitergeben, um mögliche Arbeitsabläufe der späteren Benutzer zu verbessern und vereinfachen.

Dieser Informationsaustausch fand im Rahmen des wöchentlichen Treffens statt, aber auch im Forum der PG504.

Die Teilnehmer der PG504 wurden im ersten Semester von einem Mitarbeiter des Pharmaunternehmens Novartis besucht. Novartis zeigte schon während des Entwicklungsprozesses großes Interesse am CBSE. Der Mitarbeiter hatte vor seinem Besuch die aktuellste Version des CBSE getestet und konnte weitere Anregungen aus der Sicht eines Chemikers geben, um das Programm zu verbessern.
Im weiteren Verlauf war Stefan Wetzel Ansprechpartner des Unternehmens Novartis und gab deren Feedback an die Teilnehmer der PG504 weiter.

2.3.3 Verwendete Programme

Im CBSE werden verschiedene Third-Party-Libraries verwendet. Die beiden zentralen Bibliotheken, Piccolo und Batik, werden in den folgenden Unterabschnitten näher beschrieben.

Darüber hinaus wurden noch folgende Bibliotheken verwendet:

- Common Components[1], die einige in Swing nicht vorhandene Komponenten bereitstellt, wie z.B. die `JTaskPane`, die für die Seitenleiste des CBSE verwendet wird.
- JDOM[2], eine API zur Verarbeitung von XML[3]-Dateien, die einfacher zu benutzen ist als die Java-eigene DOM-API.
- JGoodies Forms[4], ein Layoutmanager, der besonders für formularartig aufgebaute Layouts geeignet ist.
- JGoodies Looks[5], ein alternatives Look And Feel.

2.3.3.1 Arbeitsumgebung

Zur Entwicklung des CBSE wurden Eclipse[6] bzw. Together[7] verwendet. Zur Versionskontrolle und um die gemeinsame Arbeit am Code zu ermöglichen, kam das Versionskontrollsystem Subversion[8] zum Einsatz. Das Subversion-Repository wurde auf den Rechnern des PG-Pools angelegt. Zusätzlich wurde das Eclipse-Plugin Subclipse[9] verwendet, das es ermöglicht, die Arbeit mit dem Repository, z.B. Updates, direkt aus Together heraus zu erledigen.

2.3.3.2 Toolkits

Im Grafikbereich gibt es eine Vielzahl von Toolkits, die Standardfunktionen zur Verfügung stellen und die Entwicklung von Anwendung so erleichtern. Zur Darstellung von SVGs ist ein Toolkit auf Grund der Komplexität des Formats unverzichtbar. Der CBSE benutzt dazu das umfangreiche Toolkit Batik. Das Zeichnen des Scaffoldbaums wird mit Piccolo, einem Toolkit für *Skalierbare Benutzeroberflächen*, realisiert.

Piccolo

Piccolo ist ein Framework für skalierbare Benutzeroberflächen und wurde am Human-Computer Interaction Lab, University of Maryland als Nachfolger des ähnlichen Toolkits Jazz entwickelt. Es unterstützt Benutzerinteraktionen wie Zoom und Panning direkt und lässt sich in Java Swing Applikationen integrieren. Über Aktivitäten lassen sich zeitabhängige Veränderungen durchführen wie beispielsweise Animationen.

Das Tookit ist ausführlich dokumentiert und kann den anwendungsspezifischen Anforderungen weitgehend angepasst werden. Piccolo ist open-source und steht unter der *BSD Public License*, die die freie Verwendung des Toolkits erlaubt. Piccolo kommt bereits in einigen - auch kommerziellen - Applikationen zum Einsatz, was für die Qualität des Toolkits spricht.

[1] http://common.l2fprod.com/
[2] http://www.jdom.org/
[3] Extensible Markup Language
[4] http://www.jgoodies.com/freeware/forms/
[5] http://www.jgoodies.com/freeware/looks/
[6] http://www.eclipse.org/
[7] http://www.borland.com/us/products/together/
[8] http://subversion.tigris.org/
[9] http://subclipse.tigris.org/

Batik

Die Bilder der Scaffolds, die an verschiedenen Stellen im Programm verwendet werden, liegen im SVG[10]-Format vor. Zur Anzeige und Verarbeitung der Bilder war daher eine weitere Bibliothek nötig, das Batik SVG Toolkit[11]. Batik stellt unter anderem GUI-Elemente zur Anzeige von SVG-Dokumenten zur Verfügung. Diese Elemente werden überall dort verwendet, wo ausserhalb des Graphen SVGs angezeigt werden müssen. Darüber hinaus bietet es die Möglichkeit, SVGs in andere Bildformate umzuwandeln. Von dieser Funktionalität wird beim Export Gebrauch gemacht. Die im CBSE verwendete Version von Batik ist 1.6.

2.3.4 Verwendete Kommunikationsmedien

Für die Kommunikation und Koordinierung innerhalb der PG504 standen ein Forum und ein Wiki zur Verfügung.

Forum

Das Forum enthielt mehrere Unterforen: Eins für jede Teilgruppe und weitere für verschiedene gruppenübergreifende Themenbereiche, wie beispielweise Organisatorisches oder Fragen zur Chemie. In den Foren konnten sich die Teilgruppen jederzeit untereinander, mit den anderen Gruppen oder den Betreuern beraten, Fragen klären und ihre Arbeit koordinieren. Es gab auch die Möglichkeit, Abstimmungen durchzuführen.

Wiki

Das Wiki diente hauptsächlich der Dokumentation. Hier wurden die Ziele des Projekts sowie die Aufgaben der Gruppen und deren momentaner Status festgehalten. Das Wiki spiegelt die gesamte Planung und Entwicklung des Projekts wieder, von der Seminarphase mit ihren Vorträgen zur Einarbeitung in die Themen über die Planungsphase, die gesetzten Deadlines und Ziele für das Gesamptprojekt, bis hin zur Implementierung. Unter anderem wurde hier auch eine Liste der Schnittstellen gepflegt, die die einzelnen Gruppen zur Verfügung stellten. Desweiteren sammelte hier jede Gruppe die zu erledigenden Aufgaben, dokumentierte die dafür Verantwortlichen, angestrebte Deadlines und den Arbeitsfortschritt.

[10] Scalable Vector Graphics
[11] http://xmlgraphics.apache.org/batik/

2.4 Zeitlicher Ablauf

2.4.1 Übersicht

Tabelle 1 gibt eine Übersicht über die wichtigsten Aktivitäten im Verlauf der beiden Semester:

Datum	Aktivität
1. Semester	
16.11.06	Klärung der Deadline für Methoden und Schnittstellen
21.11.06	Deadline für Teilaufgaben der einzelnen Gruppen
23.11.06	Vorstellung Teilaufgaben der Teilgruppen
30.11.06	Deadline für Logoabgabe im Forum
01.12.06	Beginn der Abstimmung über Logo
08.12.06	Deadline für Präsentation des Vor-Prototypen
15.12.06	Deadline Abstimmung Logo
23.12.06	Beginn Testphase, Deadline für Klassendiagramm
19.01.07	Beginn Debugging
09.02.07	Deadline für Prototyp
2.Semester	
05.04.07	Start des 2. Semesters
26.04.07	Onlinefragebogen fertig
12.07.07	Aufteilung der DB-Gruppe
28.05.07	Deadline stabile Version
06.07.07	Deutsches Handbuch abgeschlossen
12.07.07	Code-Freeze
07.09.07	Deadline für Texte des Endberichts
13.09.07	Besprechung Endbericht und Endpräsentation
18.10.07	Endpräsentation

Tabelle 1: Zeitplan

2.4.2 Seminarphase

Die Seminarphase fand in den ersten zwei Wochen des ersten Semesters der PG504 statt. Jeder der zwölf Teilnehmer hielt einen ca. 30 minütigen Vortrag mit anschliessender Diskussion zu einem bestimmten Thema. Diese Phase diente dazu, allen Teilnehmern einen allgemeinen Überblick über die verschiedenen Themengebiete zu verschaffen, die zum erfolgreichen Abschluss der Projektes nötig sein würden. Eine Übersicht der behandelten Themen und Vortragenden liefert die folgende Tabelle:

Datum	Vortragender	Thema
17.10.	Gebhard Schrader	Graph Visualization and Navigation in Information Visualization
	Sergej Rakov	On Balloon Drawings of Rooted Trees
	Philipp Büdenbender	A parent-centered radial layout algorithm for interactive graph visualization and animation
19.10.	Vanessa Bembenek	A Focus+Context Technique based on Hyperbolic Geometry for Visualizing large Hierarchies
	Arbia Ben Ahmed	EncCon: an approach to constructing interactive visualization of large hierarchical data
	Nils Kriege	Treeplus: Interactive Exploration of Networks with Enhanced Tree Layouts
24.10.	Henning Wagner	Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams
	Anke Arndt	Toolkits
	Adalbert Gorecki	Strukturelle Ähnlichkeit
26.10.	Michael Rex	SVG
	Andre Wiesniewski	MDL
	Cengizhan Yücel	SMILES

Tabelle 2: Seminarvorträge

Es folgt eine Zusammenfassung der Inhalte der einzelnen Seminarvorträge:

- **Graph Visualization and Navigation in Information Visualization:** In dem Artikel von Ivan Herman, Guy Melancon und M. Scott Marshall gehen die Autoren auf viele unterschiedliche Darstellungsarten für grosse Datenmengen ein, z.B.: *H-Tree Layout*, *Radial View*, *Balloon View* und *Tree-Map*. Darüber hinaus berichten sie über verschiedene Formen der Repräsentation, wie kubische oder hyperbole Darstellung. Weitere Aspekte des Papers sind die Navigation innerhalb solcher Graphen und die Interaktionen mit ihnen, sowie Möglichkeiten für Focus/Context-Wechsel. Zum Schluss des Artikels werden Möglichkeiten der Gruppierung von Datensätzen näher untersucht.

- **On Balloon Drawings of Rooted Trees:** Bei der Darstellung mittels *balloon drawing* werden Kinder-Knoten auf einen Kreis angeordnet, dessen Mittelpunkt auf dem Elter-Knoten liegt. Kantenüberschneidungen sollen auf diese Art vermieden werden. Die Länge der Kanten nimmt in Beziehung zum Abstand zum Wurzel-Knoten ab. Dadurch verringert sich der Platzbedarf für die Visualisierung zum Rand der Darstellung. Die Autoren des Papers haben einen Algorithmus entwickelt, der diese Art der Darstellung berechnet. Er weist Kanten einen Spielraum in ihrer Länge und Knoten eine Variabilität in Form einer „magnetischen Ladung“ zu. Über diese Herangehensweise pendelt sich die Visualisierung mit jeder Iteration im Kontext einer ästhetischen Gesamtdarstellung näherungsweise ein, was nach wenigen Iterationsschritten zu einem sowohl schnell berechenbaren, als auch ansehnlichen Ergebnis führt.

- **A parent-centered radial layout algorithm for interactive graph visualization and animation:** Das Paper von Andrew Pavlo, Christopher Homan und Jonathan Schull behandelt eine Art des radialen Layouts, wie es Yee et. al verwenden. Ihr Algorithmus benutzt dabei eine leicht veränderte Darstellungsart, welche die Kreuzung von Kanten minimiert. Jeder Knoten ist auf einem eigenen Koordinatensystem, basierend auf seinem Elter-Knoten, positioniert - im Gegensatz zu Yees Herangehensweise, alle Knoten in einem einzigen Koordinatensystem zu platzieren. Sie vergleichen ihre Arbeit mit der von Yee et. al und kommen zu dem Schluss, dass ihr Ansatz für ein interaktives Darstellungssystem mit wenigen Kantenkreuzungen sehr gut und in diversen Anwendungsfällen sogar besser geeignet ist.

- **A Focus+Context Technique based on Hyperbolic Geometry for Visualizing large Hierarchies:** Die Autoren des Artikels haben eine Möglichkeit entwickelt, die Darstellung von Baumstrukturen mittels radialem Layout zu verbessern. Sie erreichen dies, indem sie in Anlehung an die *Fisheye*-Technik dem Focus der Visualisierung mehr Platz einräumen; Objekte, die in der Hierarchie weiter entfernt liegen, rücken verstärkt in den an den Rand der Darstellung. Darüber hinaus kommen Routinen zum Einsatz, die bei einem Kontextwechsel für eine elegante Animation der Fokusverschiebung sorgen. In einer abschliessenden Zusammenfassung kommen die Autoren zu dem Schluss, dass diese Darstellungsart einen vielversprechenden Ansatz darstellt.

- **EncCon - an approach to constructing interactive visualization of large hierarchical data:** Bei der Darstellung grosser Datenmengen gibt es in Abhängigkeit zur Darstellungsform meist verschiedene Ansätze, diese zu verwirklichen. Die Autoren dieses Artikels, Quang Vinh Nguyen und Mao Lin Huang, beschäftigen sich mit einer Darstellung von Informationen mittels Treemaps, die sowohl kompakt als auch übersichtlich sein soll. Sie vergleichen Ihre Arbeit u.a. mit bereits entwickelten Algorithmen wie *Squarified Treep-Maps*, *Strip Tree-Maps* und *Slice and Dice Tree-Maps*.

- **Treeplus - Interactive Exploration of Networks with Enhanced Tree Layouts:** Im Paper von Bongshin Lee et. al geht es um die Darstellbarkeit grosser Graphen. Die Autoren verfolgen hierbei einen Ansatz, den sie mit „pflanze ein Samenkorn und lass es gedeihen„ beschreiben. Der User startet mit einem beliebigen Knoten im Graphen und vergrössert interaktiv den sichtbaren Bereich, um so Zusammenhänge darzustellen. Unterstützt wird er dabei durch ein Kontext-Fenster, welches ihm die Beziehung von aktuellem Knoten und zur Verfügung stehender Nachfolger bzw. Vorgänger anzeigt. Mit Hilfe des anfangs kleinen Darstellungsbereiches verbessert die Arbeitsgruppe vom Lee sowohl die Orientierung als auch Navigation in grossen Graphen.

- **Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams:** In diesem Paper geht es (wie der Titel bereits vermuten lässt) um eine verbesserte Darstellung grosser Datenmengen durch die Kombination von *Treemaps* und *Node-Link*-Darstellungen. *Treemaps* stellen Informationen platzsparend dar, dies geht wiederum zu Lasten der Übersichtlichkeit. In Bezug auf *Node-Link Diagrams* sind diese positiven bzw. negativen Attribute genau umgekehr: eine gut überschaubare Darstellung benötigt viel Platz. Zhao, McGuffin und Chignell untersuchen die Möglichkeiten, die sich in der Kombination dieser beiden Visualisierungen ergeben. Ein geschickter Einsatz schafft den Spagat zwischen guter Lesbarkeit und kompakter Darstellung.

- **Toolkits:** Dieser Vortrag gab einen umfassenden Überblick über verfügbare Entwicklungswerkzeuge für Programme im biochemischen Bereich. Den Anfang machte *JoeLib*, eine Weiterentwicklung von *OeLib*. Es weist viele interessante Möglichkeiten auf, durch eine instabile Lauffähigkeit kam es aber für das Projekt nicht in Frage. Es folgte das *chemical development kit*, eine Java-Bibliothek, die alle grundlegenden Klassen und Werkzeuge für chemische Softwaretools unter der GNU General Public License zur Verfügung stellt. *Jmol* wurde als freies, open source Molekülbetrachtertool vergestellt, gefolgt von *JchemPaint* und *RDKit*. Letztere zeigten diverse Probleme mit der Integration in unser Projekt auf und ihr Einsatz wurde nicht empfohlen.

- **Strukturelle Ähnlichkeit:** Der Beitrag befasste sich mit der Suche und Findung von Parallelen sowie Gemeinsamkeiten chemischer Strukturen. Es wurde der Frage nachgegangen, wie man den Ähnlichkeitsbegriff zweier Strukturen definieren kann. Dabei wurden die beiden Formeln von Tanimoto und

Tversky vorgestellt. Die Erste nimmt das Verhältnis der grössten Gemeinsamkeit zweier Verbindungen zur Gemeinsamkeit der kleinsten Verbindung als Mass. Die zweite Formel ist ähnlich aufgebaut, jedoch wird hierbei die grösste Gemeinsamkeit in Bezug zur Differenz der Strukturen zueinander betrachtet. Der Vorteil bei dieser Herangehensweise ist, dass der Bezug von Vergleichs- zu Referenzelement stärker gewichtet wird: die Suche einer kleinen Struktur innerhalb einer Grösseren fördert aussagekräftigere Ergebnisse zutage.

- **SVG:** Das *scalable vector graphics* Format benutzt eine Vektordarstellungform für Grafiken. Üblicherweise benutzt man bei Bildern die Rasterdarstellung. Bei dieser ergeben sich Einschränkungen in der Weiterverarbeitung u.a. durch die schlechte Skalierbarkeit. Vektorgrafiken werden aus geometrischen Figuren wie Striche, Kreise und Rechtecke zusammengesetzt; eine Vergrösserung der Bildinhalte ist damit in viel höherer Qualität möglich. Das *SVG*-Format ist ein solches Vektorgrafik-Format, es ist ein offener Standard des W3C basierend auf *XML*. Es wurden in diesem Zusammenhang zwei Toolkits vorgestellt (*CDK* und *Batik*), bei dem *Batik* mit seinen ausgereifteren Methoden, *SVG*s zu erzeugen, anzuzeigen und umzuwandeln, überzeugen konnte.

- **Darstellung von Molekülen mittels SDF:** Die Firma MDL (Molecular Design Limited) entwickelte Mitte der 80er Jahre die sog. *chemical table files*, eine Darstellungsform für molekulare Strukturen. Nicht nur einzelne Strukturen, sondern Informationen zu mehreren Verbindungen können gespeichert werden. Das Format ist sehr weit verbreitet, u.a. dank der einfachen ASCII-Darstellung. Es werden viele wichtige Eigenschaften festgehalten: Strukturelle Beziehungen und Ausprägungen einer Menge von Atomen, z.B. Moleküle, ihre Bindungen (einfache/doppel), Atomkoordinaten, Massendifferenz, Wasserstoffanzahl und vieles mehr.

- **SMILES:** David Weininger entwickelte Ende der 80er Jahre das *SMILES*-Format, eine weitere Darstellungsform chemischer Strukturen. Inzwischen wird es kommerziell weiterentwickelt. Mit Hilfe einfacher Spezifikationsregeln lassen sich komplexe Strukturen beschreiben. Mittels Erweiterungen können darüber hinaus chemische Reaktionen festgelegt werden. Eine zusätzliche Weiterentwicklung sind *SMARTS*: Während *SMILES* zur Beschreibung von Molekülen verwendet wird, so benutzt man *SMARTS* bei der Kennzeichnung von Mustern. Dadurch können Substrukturen in *SMILES*-Daten gesucht werden, welches einen sehr bedeutenden Vorteil dieser Darstellung gegenüber anderen ausmacht.

2.4.3 Planungsphase

Die Planungsphase begann direkt mit dem Abschluss der Seminarphase am 26.10.2006 und ging bis zum 23.11.2006. Die Mitglieder der PG504 mussten erst einmal lernen sich zu organisieren. Da kaum jemand Erfahrung in Projektmanagement hatte, erwies sich dieses zeitweise als recht schwierig. Geplant wurde anfänglich noch wie im Software-Praktikum gelernt vorzugehen. Es sollten also die folgenden Diagramme erstellt werden:

- Klassendiagramm
- Problembereichsmodell
- Sequenzdiagramme
- Strukturmodell

Dieses Vorgehen wurde dann aber später fallen gelassen. Es wurden in den Teilgruppen lediglich die Klassendiagramm und die Sequenzdiagramme erstellt. Die PG504 entschloss sich dann nur noch die Schnittstellen abzuklären, um dann mit der Implementierung des CBSE zu beginnen.

Aufteilung in Gruppen

In einem der ersten Treffen wurde entschieden, das Projekt in 3 Bereiche aufzuteilen. Die Aufgaben der einzelnen Gruppen und deren Aufteilung sind im Folgenden zusammengefasst.

Datenbank (DB)

Diese Gruppe hatte die Aufgabe sich mit der Verbindung aus Java zu einer Datenbank zu kümmern. Sie sollten jegliche Anfragen aus dem Programm an die Datenbank heraus bedienen. Als Datenbank wurde die freie Software-Datenbank MySQL vom Kunden zur Verfügung gestellt. Mit dem JDBC Datenbank Treiber sollte die Verbindung zur Datenbank hergestellt werden und die benötigten Daten mittels SQL Befehlen von dort geholt werden. Dieses sollten in der Datenbank als Strings gespeichert werden. Des Weiteren wurden während des Planungsphase auch eine erste Version der Schnittstellen der Datenbank erstellt. Es wurde die folgenden Mitglieder dieser Gruppe zugeordnet:

- Henning Wagner
- Philipp Büdenbender
- Cengiz Yücel
- Sergej Rakov

Benutzeroberfläche (GUI)

Die Aufgabe der GUI Gruppe war es, die graphische Benutzer Oberfläche und das Rahmenprogramm zu entwerfen und zu implementieren. Die Gruppe lieferte sehr früh einen ersten Entwurf für das spätere Aussehen der Graphischen Oberfläche. Ferner kümmerte sich die Gruppe um die Druckfunktion und die Personalisierung des CBSE. Die Gruppe bestand aus den folgenden Mitgliedern:

- Arbia Ben Ahmed
- Anke Arndt
- Vanessa Bembenek
- Michael Rex
- Andre Wiesniewski (bis 7.11.2006)

Visualisierung (VIS)

In dieser Gruppe wurde die Visualisierung des Graphen entwickelt. Dazu wurde während der Planungsphase einige Toolkits betrachtet und verglichen. Es wurde beschlossen, ein radiales Layout zu implementieren. Etwaige Erweiterungen wurden auf die zweite Projektphase verschoben. Anfänglich bestand die Gruppe nur aus 3 Mitgliedern. Diese wurde aber frühzeitig dann auf 4 erweitert, da der Aufwand hier doch höher war, als noch zu Beginn angenommen. Anfänglich plante die Gruppe noch mit dem Tool Prefuse zu arbeiten. Auf den ersten Blick hatte dieses einen sehr guten Eindruck bei Mitgliedern dieser Gruppe hinterlassen. Es zeigte selbst bei 7000 Knoten eine gute Performance. Das grosse Manko bei diesem Toolkit war aber die fehlende Dokumentation und der Status einer Beta-Version. Die Mitglieder dieser Gruppe waren anfänglich:

- Adalbert Wojtek Gorecki
- Nils Kriege
- Gebhard Schrader
- Andre Wiesniewski (ab 7.11.2006)

Weiteres Vorgehen

Während der Planungsphase wurde sich auf die Entwicklungsumgebung Borland Together Architect geeinigt. Dieses wurde zur Implementierung des CBSE benutzt. Die PG504 bekam durch die Uni einen PG Pool zugewiesen, den sie dann zur weiteren Entwicklung nutzen konnten.

Die Gruppen trafen sich von da an regelmäßig und organisierten sich völlig eigenständig. In den wöchentlichen Treffen wurde der Fortschritt der Gruppen dem Rest des Projektteams mitgeteilt, so dass alle immer auf dem aktuellen Stand waren.

2.4.4 Entwicklungsphase

2.4.4.1 1. Semester

Um effektiv arbeiten zu können, wurde der Aufgabenbereich der Gruppe in weitere kleinere Teilaufgaben unterteilt, die von maximal zwei Teilnehmern bearbeitet werden konnten. Da zwischen vielen Aufgaben enge Abhängigkeiten bestanden, wurde ein Zeitplan erarbeitet, der diese berücksichtigte und das parallele Bearbeiten unabhängiger Teilaufgaben ermöglichte. Dieser musste während der Entwicklungsphase wiederholt angepasst werden, da der Zeitaufwand für einzelne Aufgaben teilweise schwierig abzuschätzen war. Ausserdem mussten neue Ideen, die erst während der Entwicklung entstanden, eingeplant werden. Durch regelmässige Treffen und eine gut funktionierende Kommunikation in der Gruppe konnte der Zeitplan dennoch gut eingehalten werden.

Aufteilung in Gruppen

Nach Abschluss der Seminarphase galt es das weitere Vorgehen zu planen, zu strukturieren und die anstehenden Aufgaben zu verteilen. Im Hinblick auf Einarbeitung und Zurechtfinden in den bereits vorhandenen Anforderungen an die Software, erschien es den Teilnehmern der PG504 als sinnvoll, die Aufgaben in drei Bereiche zu unterteilen:

- das Rahmenprogramm
- Visualisierung des Graphen und Benutzerinteraktion
- die Datenbank

Die Gruppe wurde in drei Kleingruppen mit jeweils vier Teilnehmern aufgespaltet, so dass jedes der drei Aufgabengebiete parallel bearbeitet werden konnte.

Zur Verteilung der Teilnehmer auf die einzelnen Gruppen wurde zunächst ermittelt, wer an welchem Thema besonders interessiert war. Danach wurden die Anwesenden gemäß ihres Interesses eingeteilt.

GUI	VIS	DB
Anke Arndt	Gebhard Schrader	Henning Wagner
Arbia Ben Ahmed	Adalbert Gorecki	Philipp Büdenbender
Michael Rex	Nils Kriege	Sergej Rakov
Vanessa Bembenek	Andre Wiesniewski	Cengizhan Yücel

Aufgabenbereich

Die Gruppe GUI

Zu den Aufgaben der Gruppe gehörten die Erstellung der grafischen Benutzeroberfläche. Die grafische Benutzeroberfläche (Graphical User Interface) ist eine Schnittstelle zwischen Mensch und Maschine. Dabei soll dem Benutzer mit Hilfe grafischer Gestaltung Orientierung, Navigation und Aufnahme von Inhalten erleichtert werden. Ausserdem soll eine einfache Bedienung mit der Maus und ein einheitliches Look & Feel realisiert werden. Zu den Hauptaufgaben der Gruppe gehörten die Erstellung des Rahmenprogrammes, die Funktionalität der Komponenten und das Handbuch zu implementieren, sowie typische Elemente von GUIs zum Einsatz zu bringen: Menüs, Symbolleisten (toolbar), Schaltflächen, Auswahlfelder (Button, Radio-Button, Checkbox, pull-down menu), Texteingabefelder, Abfangen und Weiterleitung von Events.

Die Teilaufgaben der GUI waren:

- Klassendiagramm aufbauen: Das Klassendiagramm dient in erster Linie dazu, die grafische Darstellung von Klassen sowie die Abhängigkeiten zwischen einzelnen Klassen darzustellen. Mit Hilfe von UML-Werkzeugen haben wir das Klassendiagramm gezeichnet und daraus ein grobe Darstellung des Codes erzeugt. (Siehe Anhang: Grobes Klassendiagramm)

- Favoriten verwalten: Der CBSE enthält eine Funktion, damit der User auf häufig besuchte Molekülen leicht zugreifen kann. So kann der Benutzer mühelos zu einem bestimmten Molekül zurückkehren, sie in Gruppen verwalten und bearbeiten, wenn er die Moleküle der Favoritenliste hinzugefügt hat. (Siehe Abschnitt 3.1.5 Bookmarks)
- Profil verwalten und Filtern (Siehe Abschnitte 3.1.3 und 3.1.7)
- Mouseover realisieren
- Detailansicht der Scaffolds im Hauptfenster: In diesen Bereich des Hauptfensters werden Detailinformationen zu gewählten Molekülen angezeigt.
- Eigenschaften der Scaffolds in eigenem Fenster darstellen: Der Benutzer kann hier Kommentare hinzufügen und speichern und die Eigenschaften der einzelnen Moleküle auslesen und drucken.
- Auflistung der Moleküle hinter einem Scaffold in einem neuen Tab: Das Programm bietet die Möglichkeit eine Molekülgruppe oder einen Unterbaum in einem neuen Tab zu öffnen.

Die Gruppe DB

Allgemeine Aufgaben der Gruppe waren:

- Datenbankanbindung durch Zufügen der benötigten Java-Bibliotheken/Treiber (JDBC-mySQL Treiber). In diesem Zusammenhang auch die Implementierung einer Verbindungsklasse, die Methoden bereitstellt, um eine Verbindung zur Datenbank aufzubauen.
- Optimierung der Datenbank und Ihrer Struktur, so dass die Daten hinsichtlich der erwarteten Nutzerzahl zu einem Zeitpunkt und deren Zugriffshäufigkeit möglichst effizient bereitgestellt werden können. Dabei spielen u.a. Entscheidungen über die Auswahl der MySQL Datenbank-Engine, wie auch der in der Datenbank verwendeten Datentypen und die Optimierung der Datenbankstruktur bezüglich besonders häufig auftretender Datenbankanfragen eine Rolle.
- Bereitstellung von Suchfunktionen für spätere Anfragen.
- Entwicklung einer Datenstruktur und Schnittstelle, die möglichst effizient arbeitet und möglichst keine weitere Speicherlast produziert. Hierbei gab es im Verlauf des 1. Projektsemesters im Grossen und Ganzen 2 verschiedene Ansätze:
  - Die Idee beim ersten Ansatz der Schnittstelle war, auf eine Anfrage seitens der beiden anderen Gruppen hin alle relevanten Daten zurückzugeben, um die Zahl der Anfragen an die Datenbank zu minimieren. Vorhandene Daten zu Scaffolds wurden also vollständig verarbeitet und weitergegeben, so dass sie bei der VIS und GUI für entsprechende Pop-Up Funktionen, o.ä. permanent vorhanden sein würden.
  - Im weiteren Verlauf wurde es dann aber in einem zweiten Ansatz aus Performanzgründen erforderlich zu bestimmen, welche der relevanten Daten für eine Anfrage zwingend notwendig sind, also für die Darstellung benötigt werden. Die übrigen relevanten Daten aus einem Anfrageresultat wurden dann erst einmal nicht verarbeitet, nicht gespeichert und nicht weitergegeben. Motiviert wurde das Ganze durch Anfrageresultate mit immensem Umfang und folglich auch immensen Bearbeitungszeiten nach dem ersten Ansatz.

Ausserdem war die Wahrscheinlichkeit im ersten Ansatz viel zu gering, dass im Nachhinein bei Anfragen mit grossem Umfang auch wirklich alle Daten des Anfrageresultats vom Nutzer berücksichtigt würden. Der zweite Ansatz hat wesentlich geringere Bearbeitungszeiten pro Anfrage zur Folge, allerdings müssen dafür weitere Anfragemöglichkeiten bereitgestellt werden, um den Abruf der Übrigen relevanten Daten zu ermöglichen, wodurch die Zahl der Anfragen an die Datenbank wiederum steigt. Insgesamt ein Vorteil, da davon ausgegangen wird, dass in den meisten Fällen im Nachhinein nur ein geringer Anteil weiterer relevanter Daten vom Nutzer nachgefragt wird.

Aufgaben bezüglich der Schnittstelle waren:

- Schnittstelle zur Visualisierungsgruppe: Implementierung von Schnittstellenklassen gemäss der Interface-Klassen DTree und DNode.
- DTree: Stellt den Graphen dar, also einen Wald. Die implementierende Klasse verwaltet Referenzen zu sämtlichen DNodes (Scaffolds) und bietet u.a. eine Methode, um zu einem gegebenen SMILES, bzw. einer Scaffold-ID, eine Referenz auf das entsprechende DNode zurückzugeben, sofern vorhanden. Dieses Interface wurde in der Klasse DBController.java implementiert.
- DNode: Stellt ein einzelnes Scaffold dar. Die implementierende Klasse verwaltet einen String, der das SVG beschreibt, und bietet u.a. Methoden um die Kennungen der Kinder oder des Elter des Scaffolds zurückzugeben. Implementiert in der Klasse Scaffold.java.
- Schnittstelle zur GUI-Gruppe: Implementierung der geforderten Methoden in einer Schnittstellenklasse.
- Sämtliche Schnittstellen zur GUI wurden ebenfalls in der Klasse DBController.java implementiert.

Dabei handelt es sich u.a. um Methoden, die alle verfügbaren Eigenschaften zu den Scaffolds identifizieren oder zu den Scaffolds die Charakteristika bzgl. der dafür verfügbaren Eigenschaften zurückgeben. Ausserdem sind ebenso auch Methoden verfügbar, die die Moleküle betreffen, die unter einem Scaffold liegen.

Zu realisieren waren somit zwei Schnittstellenklassen. Eine, die die von der GUI benötigten Methoden enthält und das Interface DTree implementiert, und eine weitere, die die Klasse DNode implementiert.

Die Gruppe VIS

Die Aufgabe der Gruppe Visualisierung war die Darstellung bzw. Zeichnen des Graphen. Nach weiteren Überlegungen wurde diese Aufgabe ein wenig genauer spezifiziert.

- Datenstruktur: Um die Daten aus der Datenbank intern überhaupt vernünftig verwalten zu können, muss eine Datenstruktur erstellt werden. Diese Datenstruktur soll einen sinnvollen Umgang mit den Daten ermöglichen.
- Speichermanagement: Da der verfügbare Arbeitsspeicher nur begrenzt ist, und weitaus kleiner als der Inhalt der Datenbank, muss eine Möglichkeit geschaffen werden, mit dem Arbeitsspeicher Haus zu halten. Der Rechner darf nicht zu schnell, oder am besten gar nicht, an seine Grenzen stossen und so unerwünschte Verzögerungen erzeugen.
- Layout: Eine für den Nutzer am deutlichsten erkennbare Aufgabe der VIS-Gruppe ist es, ein strukturiertes und übersichtliches Layout der Scaffolds zu erzeugen. Dabei ist es wichtig, so viele Informationen wie möglich darzustellen, ohne den Benutzer zu überfordern. Der Graph soll übersichtlich die Beziehungen zwischen den Scaffolds vermitteln.
- Interface: Das darzustellende Volumen an Daten ist in den meisten Fällen so groß, dass nicht alles auf einen Blick gut erkennbar auf dem Bildschirm darzustellen ist. Es kann meistens nur ein kleiner Teil des Graphen so angezeigt werden, dass man die Scaffolds und die Beziehung zu den umliegenden Scaffolds gut erkennen kann. Dem Nutzer muss die Möglichkeit gegeben werden, sich die Detailinformationen anschauen zu können, die ein Scaffold in seiner Struktur enthält, ebenso wie die Beziehungen zwischen den Scaffolds im grossen Überblick. Es muss daher eine Möglichkeit geschaffen werden, durch den Graphen zu navigieren, ohne die Gesamtübersicht zu verlieren. Der Nutzer muss mit der Darstellung interagieren, um sie verändern zu können. Dazu muss ein intuitives Benutzerinterface erstellt werden.
- Zeichnen: Zu guter letzt müssen die Scaffolds noch auf den Bildschirm gebracht werden. Die Scaffolds sind in der Datenbank im XML-Dateien gespeichert, sie müssen also ausgelesen und an einer durch das Layout vorgegebenen Position als SVG gezeichnet werden.

Organisation

Ausser dem Termin am Donnerstag für die Teilnehmer der PG504, haben sich die einzelnen Kleingruppen zu abgesprochenen Zeiten getroffen. Sie haben sich intern organisiert, Aufgaben auf die einzelnen Teilnehmer aufgeteilt und sich auf Deadlines geeinigt.

Die Organisation innerhalb der einzelnen Gruppe ist in den folgenden Teilabschnitten beschrieben.

Die Gruppe GUI

Die GUI Gruppe traf sich regelmässig am Montag, um die Aufgaben zu besprechen und weitere Planungen vorzunehmen. Es wurden organisatorische Fragen geklärt, Termine abgesprochen, Meilensteine gelegt und Probleme gemeinsam gelöst.

Im ersten Teil des Semesters befasste sich die GUI mit Modellierung, Klassendiagramm und Implementierung des Hauptfensters. Diese Aufgaben haben die Teilnehmer gemeinsam gelöst. Im zweiten Teil des Semesters teilte sich die Gruppe die weitere Implementierung auf.

Den folgenden Zeitplan hat sich die GUI Gruppe intern aufgestellt, nachdem in der grossen Runde beschlossen wurde, die Schnittstellen abzusprechen, so dass dann jede einzelne Gruppe für sich einen Zeitplan aufstellen kann:

Aufgaben	Deadlines
Klassendiagramm	23.10.2006
GUI implementieren	08.12.2006
Rahmenprogramm	12.01.2007
Handbuch	09.02.2007
Schnittstellen zur DB und VIS	Permanent

Die Gruppe DB

Die DB Gruppe hat sich folgenden Zeitplan aufgestellt:

Aufgaben	Deadlines
Klassendiagramm	23.11.2006
Klassen implementieren	07.12.2006
Datenstruktur für den Graphen	07.12.2006
Aus DB lesen	07.12.2006
Schnittstellen zur GUI und VIS	Permanent

Nachdem der Zeitplan aufgestellt wurde, konnte dies auch anfangs eingehalten werden. Ab dem 23.11.2006 wurden die Klassen implementiert und versucht eine Anbindung an eine Datenbank herzustellen.

Die Datenbank konnte nicht sofort zur Verfügung gestellt werden, weil sie noch bearbeitet werden musste. Aufgrund dessen hat die DB-Gruppe am Anfang mit einer Test-MySQL-Datenbank gearbeitet, welche wenig Datensätze enthielt. Deswegen musste erstmal die Gruppe zusehen, wie die Datenbank mit Testdaten gefüllt wurde. Dieses hat einige Zeit in Anspruch genommen. Als die Datenbank mit Dummy Daten gefüllt war, sind starke Performance Probleme aufgetreten, die dann aufgehoben wurden.

Danach bekam die Gruppe neue Datensätze und musste eine Anbindung am Programm implementieren. Dabei musste das ursprüngliche Vorgehen mit dem Klassendiagramm verworfen werden und die Probleme der Klassen aufgeteilt und bearbeitet werden.

Aufgaben	Deadlines
Implementierung: Graph zeichnen	19.12.2006
Implementierung: Knoten als SVG dargestellen	19.12.2006
Implementierung: Auf- und Zuklappen von Kindern	19.12.2006
Layout, Mini Map	23.01.2007
Puffer und zur Behebung einiger Fehler	bis Abgabe des Prototyp
Schnittstellen zur GUI und DB	Permanent

Um einen Zeitplan für die spätere Implementierung aufstellen zu können, wurde zunächst eine Planungsphase bis zum 30. November 2006 angesetzt. Während dieser Zeit wurde innerhalb der Gruppe VIS eng zusammengearbeitet, einzelne Toolkits wurden getrennt untersucht und in der Gruppe verglichen.

Da sich die Toolkits Piccolo und Prefuse, die in die engere Auswahl gezogen wurden, stark in ihrem Umfang unterschieden, beeinflusste die Entscheidung für das Toolkit Piccolo auch die spätere Zeitplanung. Als neue Teilaufgabe ergab sich damit der Entwurf eines Layouts, der erst einmal zurückgestellt wurde.

Eine Schwierigkeit bei der Einhaltung der Deadline bestand darin, dass zu diesem Zeitpunkt noch nicht mit einer Datenbank gearbeitet werden konnte und zusätzlich eine Testumgebung geschaffen werden musste, die das Paket DB vorläufig ersetzen sollte. Während dieser Zeit wurden die Teilaufgaben „Darstellung von SVGs (Toolkits Batik und Piccolo)“, „Navigation“ sowie „Erzeugen und Darstellen eines Graphen und Klappmechanismus“ weitgehend parallel bearbeitet.

Aufgrund der Verzögerung mit der Datenbank musste ausserdem in dieser Phase die Integration des Pakets db durchgeführt werden. Hierbei traten Schwierigkeiten bei der Speicherverwaltung auf, die gelöst werden konnten. Weitere Aufgaben ergaben sich aus Anregungen in den wöchentlichen Treffen.

Begleitend wurden während dieser Phase zahlreiche Verbesserungen bei der Darstellung vorgenommen, wie z.B. die Animation von Layoutveränderungen oder die Steigung der Performance. Die Zeit bis zur Abgabe des Prototypen wurden als Puffer und zur Behebung einiger Fehler verwendet.

Prototyp

Das Ziel des ersten Semester war eine erste lauffähige Version des CBSE: Der Prototyp. Diese Version sollte alle grundlegenden Funktionen, die im Pflichtenheft spezifiziert worden waren, enthalten. Die Funktionen ließen sich in fünf Bereiche aufteilen:

1. Benutzeroberfläche: Der Prototyp bot eine Benutzeroberfläche, mit der Filtereinstellungen ausgewählt und Favoriten verwaltet werden konnten. Ausserdem stellte die Benutzeroberfläche den Bereich zur Verfügung, indem der Graph gezeichnet werden konnte.
2. Navigation: Der Prototyp bot Funktionen an, mit denen im Graph stufenlos navigiert werden konnte. Dies beinhaltete zooming und paning. Vorher festgelegte Favoriten und das ausgewählte Home-Molekül ließen sich fokussieren.
3. Interaktion: Der Prototyp bot bereits vielfältige Möglichkeiten der Interaktion mit dem Graphen. Im ausgewählten Datensatz liessen sich beliebige Äste ein- und ausklappen. Durch Selektion eines Knoten wurden die Eigenschaften der Struktur angezeigt. Ausserdem bestand die Möglichkeit sich weitere Informationen einer Struktur anzeigen zu lassen, indem man mit dem Mauszeiger einige Millisekunden auf der Struktur blieb (Mouseover). Die Molekülgruppe einer Struktur und ein Teilbaum ab einem ausgewählten Knoten liessen sich im neuen Tab öffnen. Der Prototyp enthielt auch die Möglichkeit den aktuellen Graphen nach Molekülen zu durchsuchen, d.h. es konnte nach SMILES-Strings gesucht werden. Nach Absprache mit dem Kunden wurde diese Funktion aber in der finalen Version wieder herausgenommen.
4. Datenbank: Die Anbindung an eine SQL-Datenbank wurde im Prototypen schon fast vollständig implementiert. Im Laufe des zweiten Semesters wurden noch einige Optimierungen vorgenommen.

5. Layout: Im ersten Semester wurde an der Umsetzung des radialen Layout gearbeitet. Nur in einigen Sonderfällen gab es mit der Version die im Prototypen implementiert war noch Probleme. Diese lagen vor allem an der Reihenfolge der ausgeführten Threads. Weitere Layouts wurden erst im zweiten Semester implementiert.

Im Prototypen gab es noch keine Fehlerbehandlung und es wurden nur die bei der Implementierung bemerkten Bugs korrigiert. Diese erste Version des CBSE wurde von Informatikern und Chemikern getestet, so dass deren Vorschläge und Kritikpunkte bei der Weiterentwicklung im zweiten Semester berücksichtigt wurden.

Pflichtenheft

Nach den ersten Wochen, in denen die grundlegenden Entscheidungen getroffen worden waren, wurde ein Pflichtenheft verfasst.

Dieses Pflichtenheft diente als Orientierung für die Entwicklung des Prototypen des CBSE, der am Ende des ersten Semesters fertig sein musste.

Im Pflichtenheft wurden die Musskriterien festgelegt, d.h. Funktionen die auf jeden Fall im Prototypen funktionieren mussten. Aber auch schon ein paar Wunschkriterien, die eventuell im zweiten Semester implementiert werden sollten. Das Pflichtenheft bot eine Übersicht über alle Anwendungsfälle mit einer kurzen Beschreibung der dazugehörigen Funktionen. Desweiteren wurden die Systemvoraussetzungen, sowie der Anwendungsbereich und die Zielgruppe festgelegt. Damit das Pflichtenheft jederzeit von allen Teilnehmern der PG504 und den Betreuern einsehbar war, wurde es in die Wiki gestellt.

Das Pflichtenheft ist im Anhang dieses Endberichtes einsehbar.

Zwischenbericht

Zu den Anforderungen einer Projektgruppe gehört neben dem vorliegenden Endbericht ein Zwischenbericht. Dieser muss nach dem ersten Semester erstellt werden. Er gibt einen Überblick über das erste Semester und einen Ausblick auf das zweite Semester. Im Zwischenbericht wird der bis zu diesem Zeitpunkt entwickelte Prototyp, die aufgetretenen Probleme und Entscheidungsfindungen, z.B. die Auswahl der Toolkits, und alle weiteren Aspekte der Entwicklung beschrieben.

Zum einen dient er den Mitgliedern einer Projektgruppe als Übung für den späteren Endbericht, zum anderen werden die wichtigsten Punkte aus dem ersten Semester schriftlich festgehalten, damit sie am Ende der Projektgruppe nicht in Vergessenheit geraten.

Bei der Erstellung des Endberichtes wurden die Themengebiete auf die entsprechenden Teilgruppen GUI, Visualisierung und Datenbank aufgeteilt. Allgemeine Abschnitte wie z.B. die Einleitung wurden unter allen Teilnehmern verteilt.

2.4.4.2 2. Semester

Nachdem die Teilnehmer der PG im ersten Semester die Möglichkeit hatten sich an den Ablauf und die Arbeitsweisen der Projektarbeit zu gewöhnen, war der Ablauf des zweiten Semester viel routinierter. Das zweite Semester begann mit einem Resümee des ersten Semesters und dem genauen Festlegen der Aufgaben für das neue Semester. Die restlichen Anforderungen an den CBSE wurden zusammengefasst und die daraus resultierenden Aufgaben auf die Teilnehmer und Teilgruppen verteilt. Die Aufteilung der PG-Teilnehmer auf die Teilgruppen wurde ebenfalls neu überdacht und an die anstehenden Aufgaben angepasst. Des Weiteren stellte jede Teilgruppe einen Zeitplan für das zweite Semester auf. Beim Zeitplan musste berücksichtigt werden, dass eine Deadline für den Einbau neuer Features gesetzt werden sollte. Dies sollte einen ausreichend großen Zeitraum für eine Testphase und das nötige Debugging bieten.

Änderung der Gruppenaufteilung

Zu Beginn der Projektgruppe wurden drei Teilgruppen mit folgenden Mitgliedern gebildet:

- Visualisierung (VIS): Gebhard Schrader, Wojtek Gorecki, Nils Kriege, Andre Wiesniewski
- Graphisches User Interface (GUI): Anke Arndt, Arbia Ben Ahmed, Michael Rex, Vanessa Bembenek
- Datenbankanbindung (DB): Cengizhan Yücel, Henning Wagner, Philipp Büdenbender, Sergej Rakov

Diese drei Gruppen blieben bis in das zweite Semester hinein bestehen. Die Datenbank wurde zu Beginn des zweiten Semesters nochmals überarbeitet. Durch die entstandenen Veränderungen sowie Weiterentwicklungen wurden die Methoden der Datenbank-Schnittstellen angepasst. Nach einem Zeitraum von drei Wochen konnte diese Umstellung abgeschlossen werden, so dass die DB-Gruppe in ihrer Form aufgelöst werden konnte. Die Mitglieder verteilten sich auf die übrigen Kleingruppen. Sergej Rakov und Philipp Büdenbender waren zudem dafür zuständig weitere Änderungen an der Datenbank-Schnittstelle bei Bedarf vorzunehmen.

Cengizhan Yücel und Philipp Büdenbender schlossen sich der GUI-Gruppe an. Sergej Rakov und Henning Wagner wechselten zur VIS-Gruppe.

Rückkopplung und Fragebogen

Nach der Erstellung des Prototyps war eine Testphase mit mehreren Mitarbeitern des MPI und der Firma Novartis geplant. Dafür wurde eine Onlineumfrage eingerichtet, um das Ergebnis dieser Testphase statistisch auszuwerten. Aus organisatorischen Gründen wurde diese Testphase jedoch nicht durchgeführt, so dass die Teilnehmer der PG504 und die Betreuer intensiv mit dem Prototypen arbeiteten, um mögliche Verbesserungen und Fehler zu entdecken.

Der Informationsaustausch über den aktuellen Stand der Entwicklung fand wie im Kapitel „Interdisziplinäre Zusammenarbeit“ bereits geschildert im Rahmen unseres wöchentlichen PG-Treffens statt, an dem alle Studenten der PG504 und die Betreuer teilnahmen.

Falls bei der Benutzung des Programms Fehler aufgefallen oder weitere Ideen oder Fragen entstanden sind, wurden diese meistens direkt in das eingerichtete Forum gestellt, so dass die zuständige Teilgruppe auch zwischen den wöchentlichen Treffen davon erfuhr und darauf reagieren konnte.

Debugging

Damit ein Programm sinnvoll in der Praxis eingesetzt werden kann, sollte es fehlerfrei funktionieren. Darum wurde vor der Veröffentlichung des Prototypen und vor der Abgaben der Endversion eine Phase eingeplant, während der Fehler entfernt werden sollten. Aufgrund der engen Abhängigkeiten zwischen den einzelnen Programmpaketen, war dabei eine gute Kommunikation zwischen den Gruppen sehr wichtig. Über gefundene Fehler wurde die gesamte Gruppe informiert und der Fehler wurde von der verantwortlichen Person korrigiert.

Als hilfreich erwiesen sich dabei die Tools, die von der Entwicklungsumgebung Eclipse zur Verfügung gestellt werden. Neben dem integrierten Debugger steht mit der Erweiterung TPTP (Test & Performance Tools Platform) ein Profiler zur Verfügung. Hiermit konnte der Speicherverbrauch analysiert werden und Optimierungen vorgenommen werden. Bei Grafikanwendungen ist die Anzahl der Frames pro Sekunde ein Maßstab dafür wie flüssig die Grafikdarstellung wirkt. Mit der TPTP kann die Rechenzeit, die bei einem Testlauf in den einzelnen Methoden „verbracht“ wurde, gemessen werden. Dadurch lässt sich feststellen, durch welche Methoden das Zeichnen eines Frames verlangsamt wird und was die Ursache für langsame Reaktion auf Benutzereingaben oder stockende Animationen sein kann.

Endbericht

Zum Abschluss des Projektes erarbeitete die PG504 den hier vorliegenden Endbericht. Der schon nach dem ersten Semester erstellte Zwischenbericht diente als Grundlage, hinzugefügt wurden die für das Projekt relevanten Ereignisse des 2. Semesters.

Die PG504 entschied sich dazu, die einzelnen Themengebiete auf die entsprechenden Teilgruppen aufzuteilen. Abschnitte, die die gesamte Projektgruppe betrafen, wurden unter allen Teilnehmern aufgeteilt. Nach mehreren Überarbeitungsschleifen wurde dieser Endbericht verabschiedet.

Dem Abschlussbericht folgte noch ein Fachgespräch, in dem die Projektgruppe ihre Arbeit dem Fachbereich und allen Interessierten vorstellte. Die PG504 war hiermit beendet.

3 Der CBSE

3.1 Aus Sicht der GUI

3.1.1 Handbuch

Da das Programm nicht selbsterklärend ist, entschlossen die Teilnehmer der PG504 am Anfang des 1. Semesters ein Handbuch zu entwerfen. Zweck des Handbuches ist, dass die Kunden jederzeit nachlesen können, wenn sie ein Problem mit dem Programm haben oder nicht wissen, wie man bestimmte Funktionen bedient. Dieses Handbuch wurde im 1. Semester in das Programm integriert. Damals war es aber nur als Pdf-Datei verfügbar. Die PG504 plante später auch eine Suchfunktion nach Stichwörtern zu integrieren. Im 2. Semester haben sich die Teilnehmer der PG504 dazu entschlossen, dass das Handbuch als html-Datei eingebunden werden sollte und über den Hilfe-Button in den verschiedenen Dialogfenstern aufgerufen wird. Somit gelangen die Nutzer zu den einzelnen Kapiteln, welche sie über die betreffenden Probleme informieren. Weiterhin kann man aber auch das gesammte Handbuch aufrufen. Die PG504 hat am Ende des 1. Semester ebenfalls geplant, dass das Handbuch in die englische Sprache übersetzt wird. Dies wurde aber aus organisatorischen und fachlichen Gründen verschoben. Die Betreuer haben sich dieses Themas angenommen.

Arbeitsumgebung

Als Arbeitsumgebung haben sich die Teilnehmer der PG504 für LaTex entschieden, da es sehr praktisch ist, die ganzen Kapitel in kleine Dateien zu packen, die man dann über einen „include“-Befehl einbinden kann. Der Vorteil war, dass man gleichzeitig an dem Handbuch arbeiten konnte, ohne die anderen zu behindern. Ausserdem konnten die Teilnehmer mit LaTex professioneller arbeiten als es das Programm Word zulassen würde.

Systemvorraussetzungen

Bei den Systemvoraussetzungen hat die PG504 folgende Anforderungen festgelegt, denn der Aufbau des Graphen erfordert einige Ressourcen vom Computer und da sie nicht wollten, dass der Benutzer am Ende lange darauf warten muss, bis sich der Graph mit der eingestellten Anzahl von Scaffolds aufbaut, haben die Teilnehmer der PG504 diese Anforderungen etwas höher angesetzt als notwendig war.

Hardware

Mindestanforderungen

Folgende Mindestanforderungen hat die PG504 zu Anfang festgelegt:

- Pentium 400 MHz-Prozessor
- Mindestens 128 MB RAM
- Mindestens 20 MB verfügbarer Speicherplatz auf der Festplatte
- Tastatur und Microsoft Mouse oder ein kompatibles Zeigegerät
- Videoadapter und Monitor mit Super VGA (800 x 600) oder höherer Auflösung

Im Laufe der Zeit veränderten sich diese Mindestanforderungen aber noch, da das Programm komplexer wurde und die PG504 daher höhere Mindestanforderungen setzen müsste. Um das Programm stabil laufen zu lassen, benötigten die Teilnehmer der PG504 mindestens folgende Eigenschaften eines Computers:

- Pentium 1-GHz-Prozessor

- Mindestens 512 MB RAM

Empfohlen
Auch hier hat die PG504 anfangs die empfohlenen Anforderungen definiert:

- Pentium 1-GHz-Prozessor
- 512 MB RAM
- Grafikkarte und Monitor mit einer Auflösung von 1024 x 768

Diese Anforderungen mussten gegen Ende des 2. Semesters ebenfalls von den Teilnehmern der PG504 an das fertige Programm CBSE angepasst werden.

- Pentium 2-GHz-Prozessor
- 1 GB RAM
- Grafikkarte und Monitor mit einer Auflösung von 1024 x 768

Software

Wie bei der Hardware mussten die Teilnehmer der PG504 auch bei der Software Anforderungen an ihr Programm stellen. Zu Beginn des 1. Semesters haben sie folgende Anforderungen festgelegt.

- Windows 98/2000/XP
- Java JRE 1.5 / 5.0
- Internetbrowser

Desweiteren benötigen die Benutzer noch eine SQL-Datenbank und einen Internetzugang, bzw. Netzwerkzugang. Da die PG504 aber ihr Programm plattformunabhängig anbieten wollte, haben sie zum Ende des 1. Semesters diese Anforderungen gelockert. Es reicht vollkommen aus, wenn der Benutzer folgende Anforderung erfüllt:

- Java JRE 1.5 / 5.0

Überblick

Das Handbuch wurde in 14 Kapitel unterteilt:

1. Einleitung
2. Systemvoraussetzung
3. Erster Programmstart
4. Profil
5. Filterdialoge
6. Knoten einfärben
7. Menüleiste
8. Drucken
9. Export

10. Details
11. Bookmarks
12. Navigation im Graphen
13. Tastenkürzel
14. FAQ

Benutzerfreundlichkeit des Handbuches

Im Handbuch wurden die wichtigsten Punkte zur Bedienung des CBSE aufgenommen. Im 1. Semester gab es erst insgesamt 8 Kapitel, die nach Meinung der Teilnehmer der PG504 ausreichen würden. Aber im 2. Semester sind viele Funktionen hinzugekommen, so dass einige Kapitel erweitert, bzw. neu hinzugefügt wurden. Ausserdem mussten einige kleine Kapitel entstehen, damit diese Kapitel als einzelne html-Dateien erzeugt und aufrufen werden konnten. Dies wurden für die „Hilfe“-Button-Funktion benötigt.

FAQ

Bei den FAQ wurden einige der häufigsten Fehlermeldungen und Probleme durch Benutzer aufgeführt. Dieser FAQ-Katalog wurde im 1.Semester recht klein gehalten, da viele Probleme den Teilnehmern der PG504 nicht aufgefallen sind. Zum Ende des 2. Semesters, wurde dieser Fragenkatalog erweitert. Hierzu wollte die PG504 die Fragebögen auswerten und auf die Fragen und Hinweise der Benutzer eingehen, aber leider ist die Sache mit den Fragebögen nicht zustande gekommen. Am Ende haben die Teilnehmer der PG504 überlegt, welche Fehler auftreten könnten und hofften, dass sie die meisten Fragen beantwortet haben.

Layout

Das Layout des Handbuches wurde übersichtlich und strukturiert gestaltet. Viele Punkte wurden nicht in lange Texte verfasst, sondern wegen der Übersichtlichkeit in eine Aufzählung umgesetzt. Dies erleichtert dem Benutzer die Lesbarkeit und ist benutzerfreundlicher.

3.1.2 Benutzerinterface

Das Hauptfenster des CBSE ist, abgesehen von Menü und Toolbar, in zwei Bereiche aufgeteilt: Den Großteil des Fensters nimmt die Darstellung des Graphen ein, links davon befindet sich eine Seitenleiste (siehe Abb. 1).

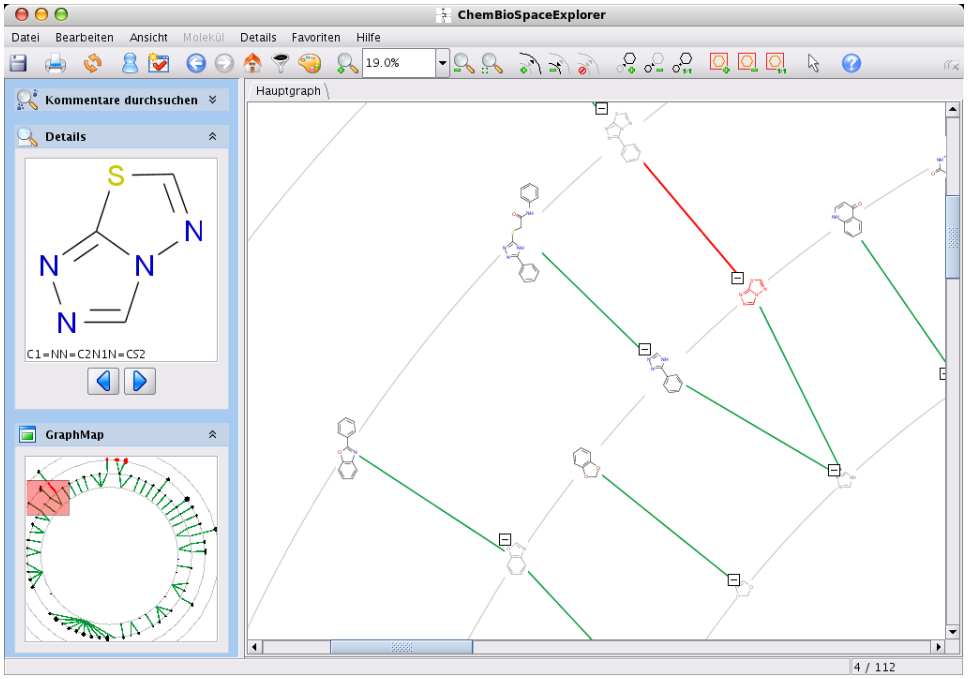

Abbildung 1: Hauptfenster

Die Seitenleiste enthält mehrere Elemente, über die der Benutzer mit dem Graphen interagieren kann. Man kann hier u.a. in den Kommentaren der Scaffolds nach einem bestimmten Text suchen und sich die passenden Scaffolds anzeigen lassen. Ein weiterer Bereich stellt ein Bild des aktuell ausgewählten Scaffolds sowie benutzerdefinierte Detailinformationen dar, ebenso kann man hier durch die ausgewählten Scaffolds blättern und sie im Graphen anzeigen lassen. Desweiteren enthält die Seitenleiste eine Minimap, die eine Übersicht über den Graphen bietet, sowie ein Vergrößerungsglas, um einen Teil des Graphen detaillierter betrachten zu können. Ursprünglich enthielt die Seitenleiste auch ein Feld für die Suche nach Scaffolds anhand ihrer SMILES. Dies wurde aber wieder entfernt, da sich die SMILES je nach Programm, mit dem sie generiert wurden, unterscheiden. Es wäre somit für die Benutzer eher verwirrend gewesen, wenn die SMILES, die sie zur Suche verwenden, nicht mit denen übereinstimmen, die in der Datenbank verwendet werden.

Der Hauptteil des Fensters zeigt normalerweise den Graphen an. Hier werden aber auch Ansichten für Teilbäume des Graphen sowie die Moleküle, die hinter einem Scaffold liegen, angezeigt. Die neuen Ansichten werden in zusätzlichen Tabs angezeigt, so dass der Nutzer eine beliebige Anzahl davon parallel öffnen kann.

Das aktuelle Aussehen des Hauptfensters entspricht größtenteils den ersten Überlegungen, die die GUI-Gruppe sich zur Aufteilung des Fensters gemacht hat (siehe Abb. 2). An einigen Stellen gab es aber auch Änderungen:

- Das „Suchen“-Feld in der Seitenleiste erlaubt jetzt nur noch die Suche in den Scaffold-Kommentaren. Die anderen Suchoptionen werden durch die Möglichkeit abgedeckt, den Graphen nach benutzerdefinierten Kriterien zu filtern.

- Die Favoriten werden nicht mehr in der Seitenleiste dargestellt, sondern im Menü. Zur Verwaltung der Favoriten gibt es nun einen eigenen Dialog.
- Die Minimap wurde in die Seitenleiste integriert.
- Ein Vergrösserungsglas wurde mit in die Seitenleiste aufgenommen.

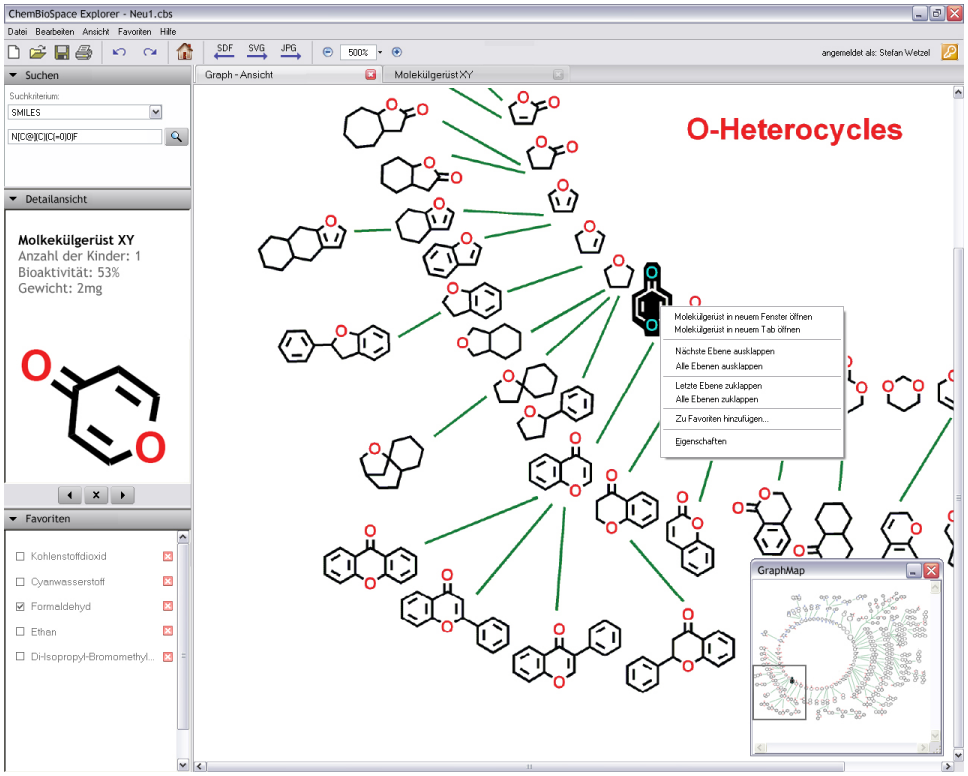

Abbildung 2: Früher Entwurf des Hauptfensters

3.1.3 Profile

Der CBSE soll jedem Benutzer die Möglichkeit bieten, persönliche Einstellungen vorzunehmen und auf diese Weise eine von ihm bevorzugte Sicht auf die dargestellten Daten zu erlangen. Um dies zu gewährleisten wurde eine Profilverwaltung entwickelt, die alle Einstellungen des Benutzers kapselt und auch über das Ende des Programms hinaus speichert.

Zu Beginn der Entwicklungsphase wurden Überlegungen angestellt, welche Benutzerdaten in einem Profil gespeichert werden sollten:

- der Benutzer muss einen eindeutigen Profilnamen besitzen
- der CBSE bietet die Sprachen Deutsch und Englisch an, die Wahl des Benutzers muss gespeichert werden
- die notwendigen Daten für die Verbindung zur Datenbank müssen bekannt sein (URL, Benutzername, Passwort)
- es kann ausgewählt werden, wie viele Ringe beim Start des Programm angezeigt werden sollen
- es kann ein Home-Molekül ausgewählt werden
- es können Scaffold-Eigenschaften gewählt werden, die im Tooltip und im Details-Fenster angezeigt werden
- der Benutzer hat die Möglichkeit, eigene Kommentare zu den Scaffolds zu verfassen
- der Benutzer soll einzelne Scaffolds zu seinen Favoriten hinzufügen können

Daraus ergab sich die in Abbildung 3 dargestellte Klassenstruktur.

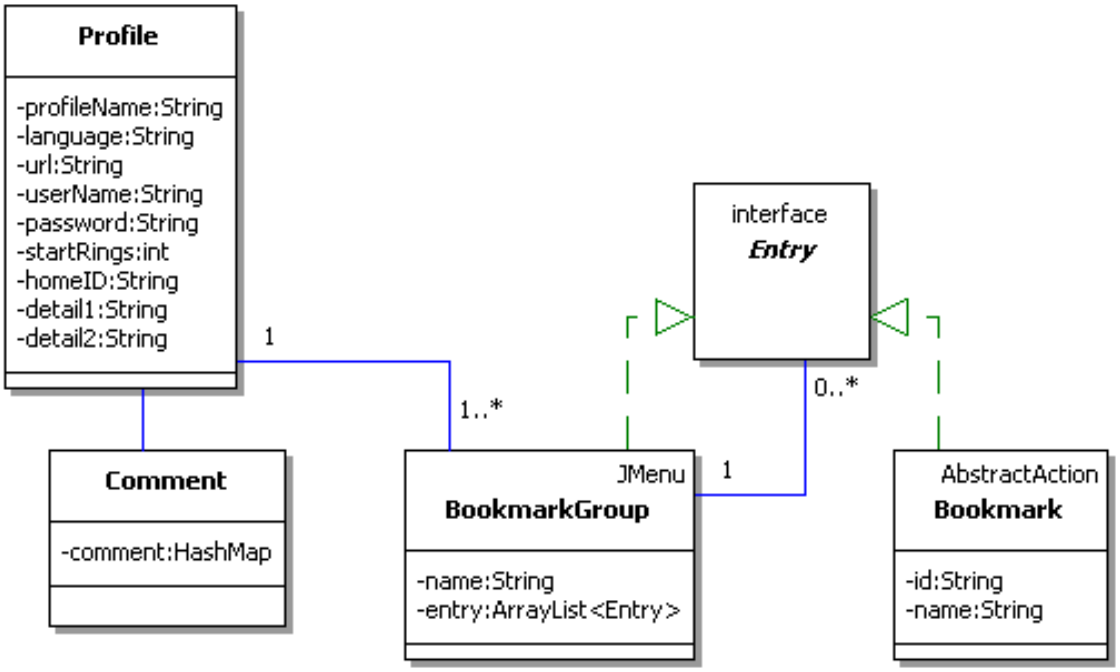

Abbildung 3: Erste Klassenstruktur der Profilverwaltung

Auf diesen Überlegungen aufbauend wurden die ersten Entwürfe für die entsprechenden Eingabemasken entwickelt. Es werden zwei Dialoge benötigt: Einer zur Erstellung eines neuen Profils und ein weiterer zur Änderung und Erweiterung eines bereits existierenden Profils. Damit das Erstellen eines neuen Profils schnell durchgeführt werden kann, sollen in diesem Dialog nur die wichtigsten Daten (der gewünschte Profilname und die Verbindung zur Datenbank) vom Benutzer abgefragt werden. Es soll dem Benutzer jedoch zusätzlich ermöglicht werden bereits an dieser Stelle eine Sprache und die Anzahl der Ringe auszuwählen. Diese beiden Angaben sind jedoch optional, als Standardeinstellungen werden bisher Deutsch als Sprache und die Anzahl der Ringe auf 1 gesetzt.

Sobald das Hauptfenster des CBSE erscheint, ist es möglich, die bisherigen Einstellungen zu verändern - ausgenommen den Profilnamen - und weitere Einstellungen vorzunehmen. So kann nun zusätzlich der SMILES eines Home-Moleküls eingegeben werden, das beim Klick auf den Home-Button in der Toolbar zentriert werden soll. Wird dieses Textfeld frei gelassen, so wird der Mittelpunkt des angezeigten Graphen fokussiert. Ausserdem soll es die Möglichkeit geben, Scaffold-Eigenschaften auszuwählen. Um den Tooltip übersichtlich gestalten zu können, wurde beim ersten Entwurf der Dialoge die Angabe von nur zwei Scaffold-Eigenschaften vorgesehen. Die Kommentare zu einem Scaffold und die Favoriten des Benutzers, die ebenfalls in seinem Profil gespeichert werden sollen, können an anderer Stelle hinzugefügt werden.

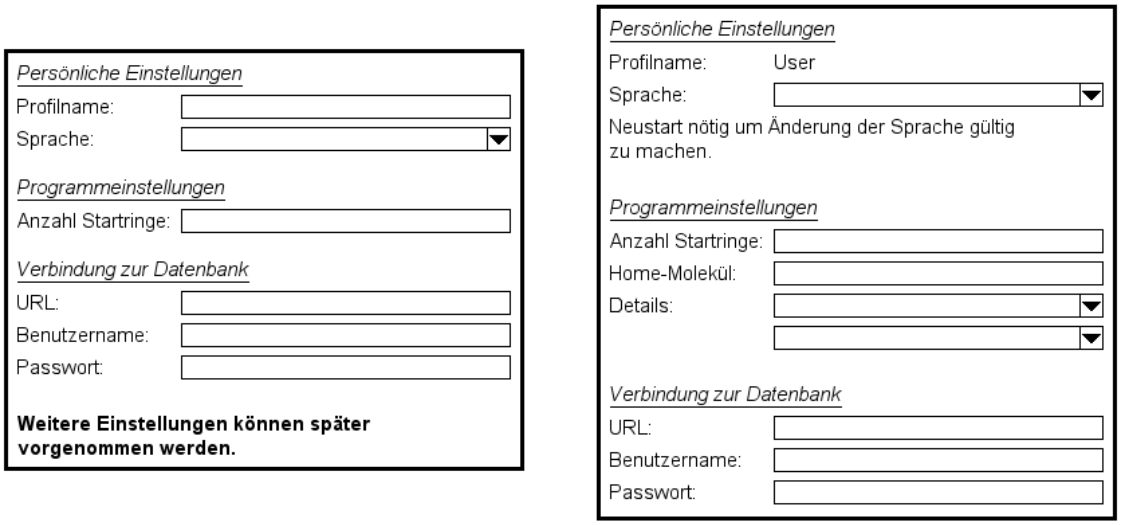

Abbildung 4: Entwurf der Profildialoge

Wie in Abbildung 4 zu erkennen ist, unterscheiden sich die beiden Dialoge nur in einigen wenigen Punkten. Aus diesem Grunde wurde entschieden, beide Dialoge mit einer Klasse `ProfileDialog` zu realisieren und über ein boolesches Attribut `firstLogin` zu entscheiden, ob der Dialog zum Erstellen eines neuen Profils oder der Dialog zum Bearbeiten eines bereits existierenden Profils aufgebaut werden soll.

Während der Entwicklungsphase ergaben sich weitere Einstellungen, die ein Benutzer vornehmen kann und die in seinem Profil gespeichert werden sollen. So wurde das Programm um die Möglichkeit erweitert, Anti Aliasing komplett ein- oder auszustellen oder es nur während Animationen zu deaktivieren. Um diese Einstellung zu speichern, wurde der Klasse `Profile` ein Atribut `renderQuality` vom Typ `int` zugeordnet. Dieses Attribut kann die Werte dreier in der Klasse `VISControl` definierter Konstanten annehmen:

- *`RENDERING_QUALITY_LOW`* falls das Anti Aliasing stets ausgeschaltet sein soll
- *`RENDERING_QUALITY_HIGH`* falls das Anti Aliasing stets eingeschaltet sein soll

- *RENDERING_QUALITY_AUTO* falls das Anti Aliasing während Animationen und Interaktionen deaktiviert werden soll

Desweiteren wurde die Möglichkeit eingebaut das Fokussieren von Scaffolds nach dem Auf- und Zuklappen von Kindern oder Ebenen, die Kamera-Animation und die Layout-Animation ein- bzw. auszustellen. Für diese Einstellungen wurden der Klasse noch drei boolesche Attribute `focusAfterAction`, `cameraAnimation` und `layoutAnimation` hinzugefügt.

Nur das Verhalten des Programms bezüglich Anti Aliasing soll bereits beim Erstellen eines neuen Profils eingestellt werden können. Standardmässig ist das Anti Aliasing immer eingeschaltet. Das Verhalten bezüglich Fokussieren, Kamera-Animation und Layout-Animation kann erst später eingestellt werden. Der am Ende des ersten Semesters im Programm verwendete Dialog zur Erstellung eines neuen Profils ist in Abbildung 5 dargestellt.

Abbildung 5: Dialog zur Erstellung eines neuen Profils am Ende des ersten Semesters

Neben diesen geringen Erweiterungen ergaben sich noch weitere Änderungen, die ein Überarbeiten der Klassenstruktur und der Dialoge notwendig machten. Auf die Änderungen, die sich bezüglich der Bookmarks ergaben, wird in Abschnitt Bookmarks näher eingegangen. An dieser Stelle soll nur auf die Änderungen bezüglich der Auswahl der Scaffold-Eigenschaften für den Tooltip und das Details-Fenster eingegangen werden.

Seitens der Betreuer wurde die Vermutung geäussert, dass eine feste Einschränkung auf die Angabe von nur zwei Scaffold-Eigenschaften vom Benutzer als zu einschränkend empfunden werden könnte. Es wurde eine offenere Handhabung gewünscht, die die Festlegung der Anzahl der Eigenschaften dem Benutzer überlässt. Aus diesem Grunde wurden die beiden Attribute `detail1` und `detail2` vom Typ `String` aus der Klasse `Profile` gestrichen und durch ein Attribut `details` vom Typ `ArrayList<String>` ersetzt. Im entsprechenden Dialog wichen die beiden `JComboBox`en einer aus der Datenbank ausgelesenen Auflistung aller verfügbaren Scaffold-Eigenschaften, von denen jede über eine `JCheckBox` markiert

und somit in den Tooltip und das Details-Fenster aufgenommen werden konnte. Beim Beenden des Dialogs wurden die Namen der ausgewählten Eigenschaften dann in die `ArrayList<String>` geschrieben.

Dies stellte allerdings immer noch keine zufriedenstellende Lösung dar, da zu jeder Scaffold-Eigenschaft unterschiedliche Werte in der Datenbank existieren können. In der uns zur Verfügung stehenden Testdatenbank sind das die folgenden: *mean*, *stddev*, *median*, *min* und *max*. Die Idee, für alle gewählten Scaffold-Eigenschaften nur einen dieser Werte anzuzeigen, wurde schnell als zu einschränkend verworfen. Stattdessen entschieden sich die Entwickler dafür, das Konzept des Filter-Dialogs (siehe Kapitel 3.1.7) wiederzuverwenden. Es ist dem Benutzer nun freigestellt, wie viele Scaffold-Eigenschaften er angezeigt bekommen möchte. Der am Ende des ersten Semesters im Programm verwendete Dialog zur Änderung eines existierenden Profils ist in Abbildung 6 dargestellt.

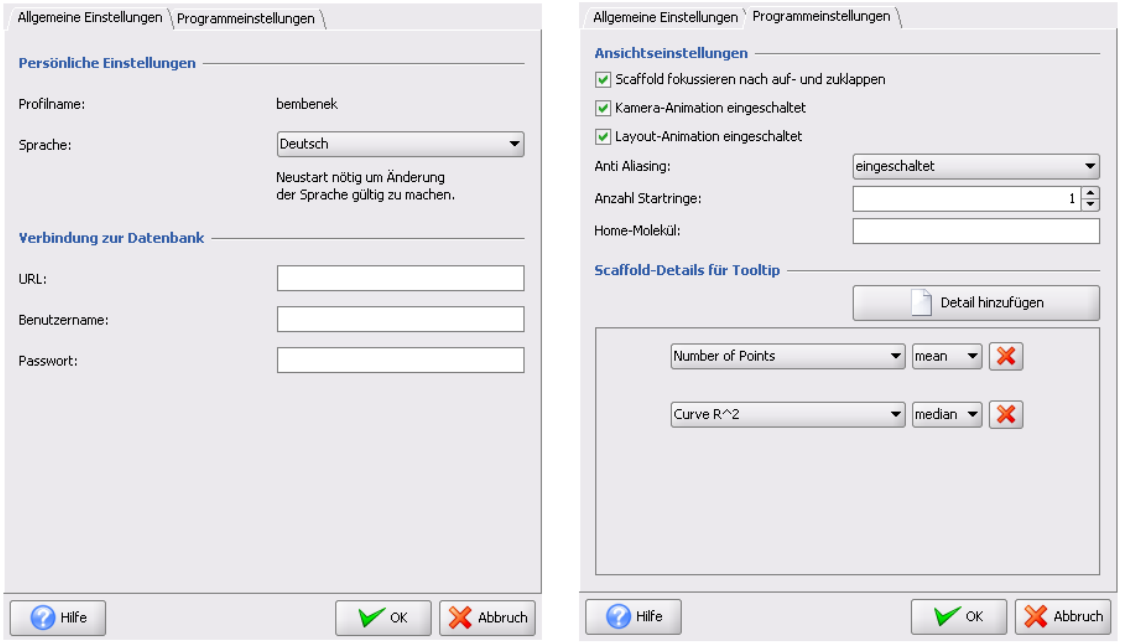

Abbildung 6: Dialog zur Änderung eines existierenden Profils am Ende des ersten Semesters

Um die Wahl der Scaffold-Eigenschaften vernünftig im Profil speichern zu können, wurde eine Datenklasse `Detail` entwickelt, die die Informationen einer Eigenschaft kapselt. Diese Klasse besitzt drei Attribute:

- `propertyID` vom Typ `String` speichert die ID einer Eigenschaft
- `propertyTitle` vom Typ `String` speichert den Titel einer Eigenschaft, der in der `JComboBox` angezeigt wird
- `type` vom Typ `String` speichert den Wertetyp, der für die ausgewählte Eigenschaft angezeigt werden soll

In der Klasse `Profile` wurde der Typ `ArrayList<String>` des Attributs `detail` durch den Typ `ArrayList<Detail>` ersetzt.

Wie der Vergleich der Abbildungen 5 und 6 deutlich macht, sind die Unterschiede beider Dialoge wesentlich grösser geworden. Der Ansatz beide Dialoge durch eine Klasse zu realisieren, wurde daher verworfen. Stattdessen wird in der endgültigen Version des CBSE der Dialog zur Erstellung eines Profils durch die Klasse `NewProfileDialog` und der Dialog zur Änderung eines Profils durch die Klasse

`ProfileDialog` aufgebaut.

Die Kommentare zu einem Scaffold kann der Benutzer über das Eigenschaftsfenster eines Scaffolds eingeben (siehe Abbildung 7 rechts). Gespeichert werden müssen diese jedoch auch über das Profil eines Benutzers. Dafür wurde eine zusätzliche Klasse `Comment` eingeführt, die allerdings nur eine `HashMap` kapselt. In dieser werden die einzelnen Kommentare gespeichert, wobei die ID des Scaffolds als Schlüssel verwendet wird.

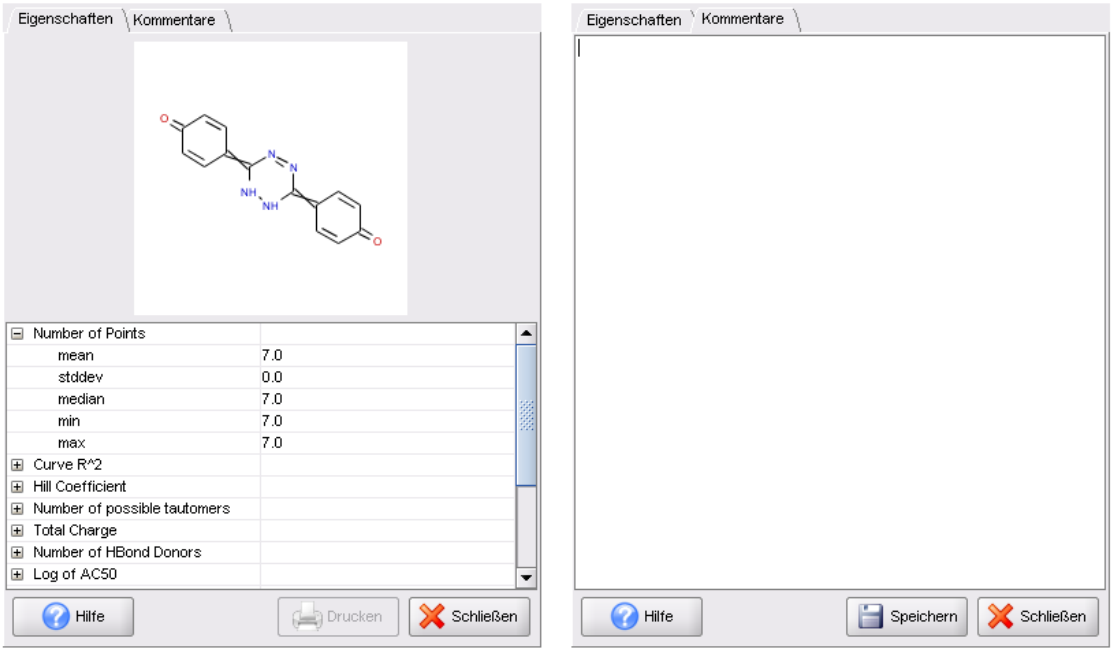

Abbildung 7: Eigenschaften-Dialog eines Scaffolds

Während des zweiten Semesters wurde das Programm noch einmal um weitere Einstellungsmöglichkeiten erweitert, was zu einer erneuten Überarbeitung der Dialoge zur Erstellung und Änderung eines Profils und einer Erweiterung der Klassen führte. Die Anforderung, Profile sowohl auf der Festplatte als auch in der Datenbank speichern zu können, wurde umgesetzt und dem Benutzer musste die Wahl gelassen werden, wo er sein Profil abspeichern möchte. Die Entwickler entschieden sich dafür, diese Auswahl nur beim Erstellen eines neuen Profils zuzulassen, um möglichst zu verhindern, dass Profile mit dem gleichen Namen sowohl auf der Festplatte als auch in der Datenbank gespeichert sind. Daher wurde zwar beiden Dialogen eine `JComboBox` für diese Auswahl hinzugefügt, in dem Dialog zur Änderung eines Profils diese jedoch deaktiviert. Um die Wahl des Benutzers zu speichern, wurde der Klasse `Profile` ein weiteres boolesches Attribut `saveOntoHD` hinzugefügt. Standardmässig wird das Profil auf der Festplatte gespeichert.

Ausserdem wurde es dem Benutzer auf Wunsch der Betreuer ermöglicht, zusätzlich zu den bisher vorgesehenen Verbindungsdaten zur Datenbank den Namen der zu nutzenden Datenbank anzugeben. Dafür wurde den Dialogen ein weiteres `JTextField` hinzugefügt und die Klasse `Profile` um ein Attribut `databaseName` des Typs `String` erweitert. Zur Vereinfachung für den Benutzer wird das Textfeld standardmässig mit einem „\“ gefüllt und es wird nach erfolgter Eingabe zusätzlich überprüft, ob dem Datenbankname ein „\“ vorangestellt ist und, falls nicht, wird dieser ergänzt.

Bis auf das ebenfalls im zweiten Semester hinzugefügte Eingabefeld für die Angabe der maximalen Anzahl sichtbarer Scaffolds, auf dessen Funktionalität im Kapitel 3.1.7 näher eingegangen wird, stellen die eben beschriebenen Erweiterungen die einzigen Neuerungen innerhalb des Dialogs zur Erstellung eines neuen Profils dar. Der in der endgültigen Programmversion genutzte Dialog ist in Abbildung 8 dargestellt.

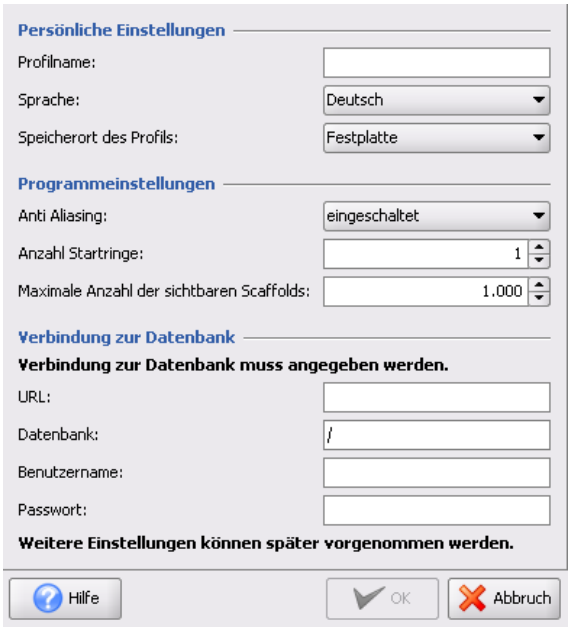

Abbildung 8: Endgültiger Dialog zur Erstellung eines neuen Profils

Für den Dialog zum Ändern eines Profils gab es, ausser den bisher beschriebenen neuen Elementen, zuerst nur wenige Erweiterungen. Es sollte dem Nutzer ermöglicht werden, anzugeben ab welcher Grösse von Unterbäumen das Programm beim Öffnen eines Unterbaums eine Warnung ausgeben soll und wie viele Zeilen und Spalten in der Molekülansicht pro Seite angezeigt werden sollen. Dafür wurden dem Dialog drei `JSpinner` hinzugefügt und der Klasse `Profile` drei Attribute des Typs `int`. Zusätzlich wurden noch zwei `JCheckBox`en hinzugefügt, über die die Animation des Cursors ein- und ausgeschaltet und die Kanten von Unterbäumen ein- und ausgeblendet werden können. Auch für diese Einstellung wurden der Klasse `Profile` zwei neue Attribute hinzugefügt, diesmal vom Typ `boolean`. Standardmässig ist die Animation des Cursors ausgeschaltet und die Kanten der Unterbäume sind ausgeblendet. Der in der endgültigen Programmversion genutzte Dialog ist in Abbildung 8 dargestellt.

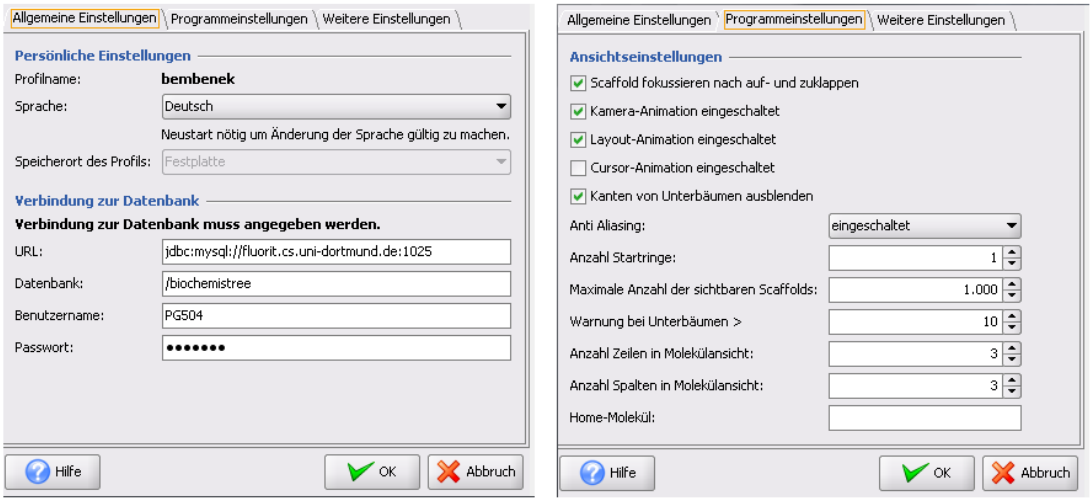

Abbildung 9: Endgültiger Dialog zur Änderung eines existierenden Profils

Wie man in Abbildung 9 sieht, wurde die Auswahl der Scaffold-Eigenschaften für den Tooltip und das Details-Fenster komplett aus dem Dialog zur Änderung eines Profils entfernt. Dies geschah auf Anregung eines Betreuers, der darauf hinwies, dass ein Benutzer es praktikabler finden könnte, diese Einstellung in einem separaten Dialog vorzunehmen, der mit einem Klick über die Toolbar aufgerufen werden kann. So wurde der komplette Punkt in einem eigenen Dialog untergebracht, der durch die Klasse `DetailsDialog` aufgebaut wird (siehe Abbildung 10).

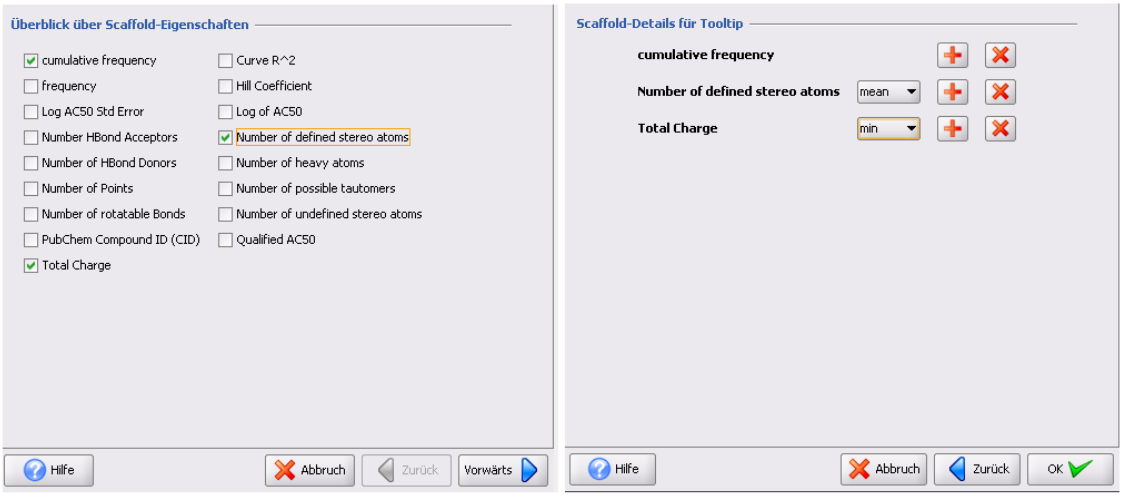

Abbildung 10: Dialog zur Auswahl der Scaffold-Eigenschaften für Tooltip und Details-Fenster

Hierbei wurde erneut das Konzept des inzwischen überarbeiteten Filter-Dialogs wiederverwendet. Durch den zweischrittigen Dialog kann der Benutzer auf schnelle und einfache Weise Eigenschaften und dazugehörige Wertetypen auswählen, deren Werte im Tooltip bzw. Details-Fenster angezeigt werden sollen (siehe Abbildung 11).

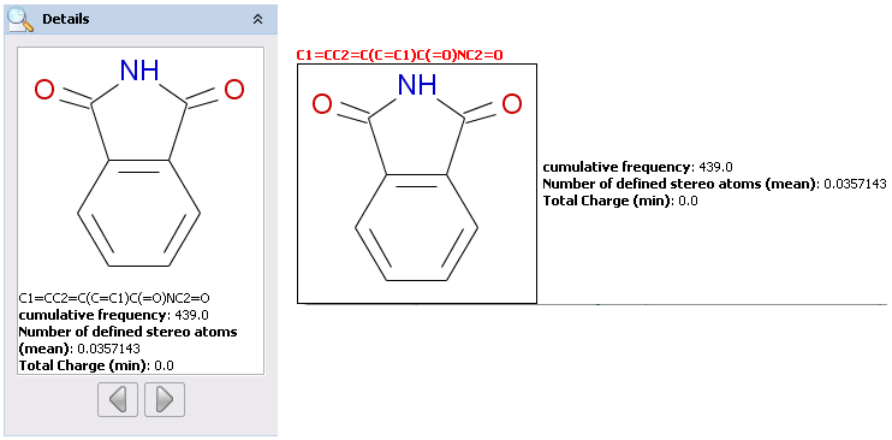

Abbildung 11: Markiertes Scaffold im Details-Fenster und zugehöriger Tooltip

Schließlich wurde der CBSE zum Ende des zweiten Semesters noch um die Möglichkeit erweitert, die Strichstärke, in der die Scaffolds und Kanten des Graphen und der Cursor gezeichnet werden, ebenso wie die Farben, in denen diese Komponenten des Graphen eingefärbt werden, einzustellen. Dafür wurde der Dialog zum Ändern des Profils um das Tab „Weitere Einstellungen“ erweitert, das Eingabemöglichkeiten für die Änderung der Strichstärken und Farben jeder Komponente ermöglicht (siehe Abbildung 12). Um diese Einstellungen speichern zu können, wurden der Klasse `Profile` für jede der Eingabemöglichkeiten ein neues Attribut hinzugefügt.

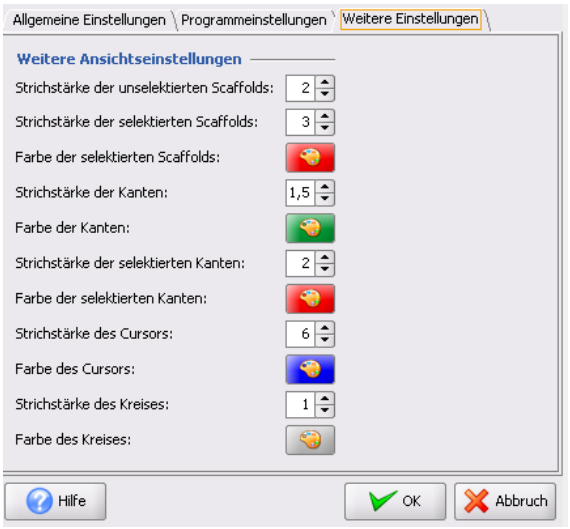

Abbildung 12: Tab „Weitere Einstellungen“ im endgültigen Dialog zur Änderung des Profils

Der Ausschnitt des Klassendiagramms, der die Beziehungen der nötigen Klassen für die Profilverwaltung und der zugehörigen Dialoge veranschaulicht, ist in nachfolgender Abbildung dargestellt. Hier wird noch einmal deutlich, dass die Klasse `Profile` für jede Einstellungsmöglichkeit ein eigenes Attribut besitzt. Die für die Attribute vorhandenen `get`- und `set`-Methoden wurden der Übersicht halber jedoch ausgeblendet.

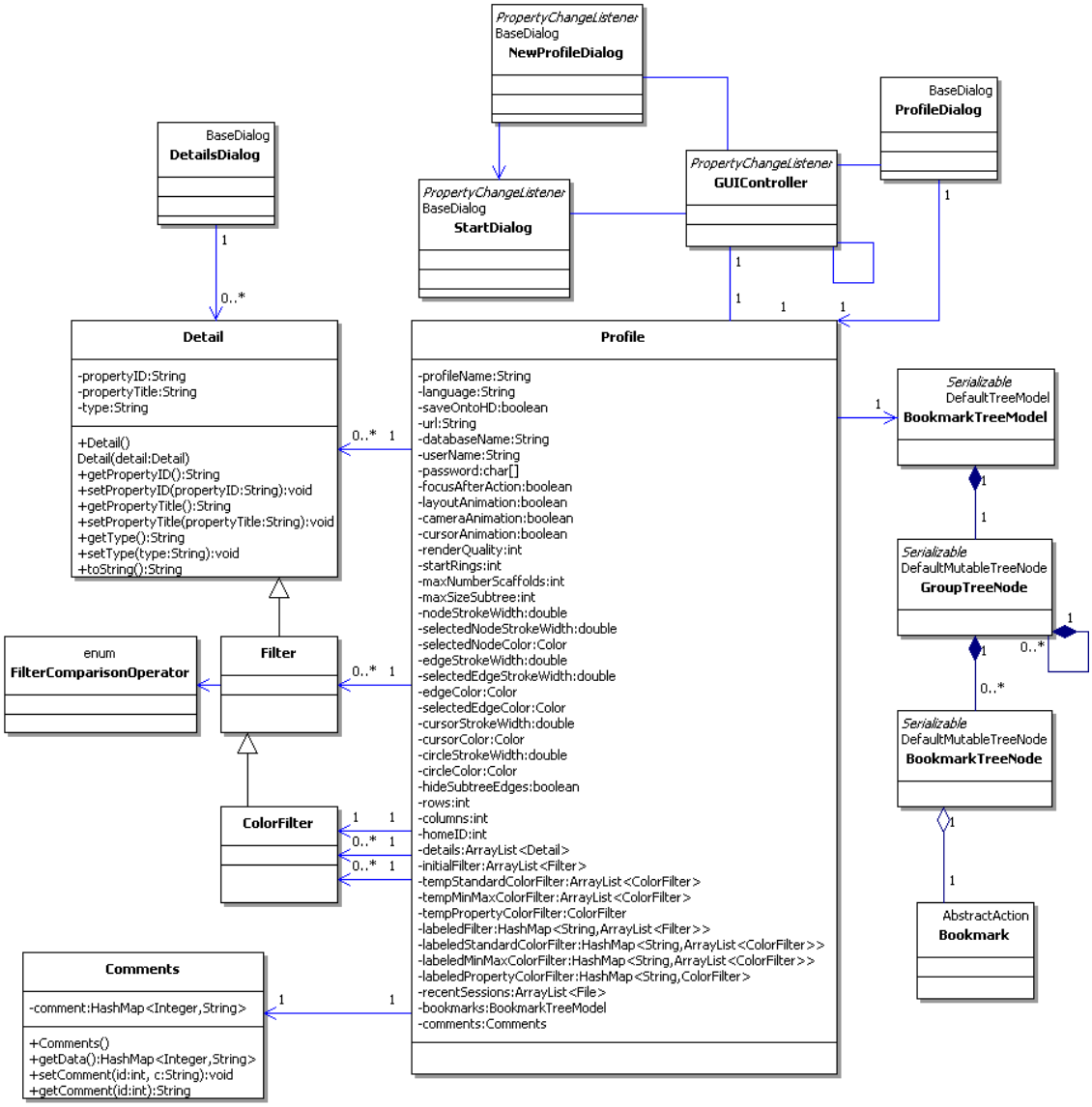

Abbildung 13: Aktuelles Klassendiagramm der Profilverwaltung

3.1.4 Drucken

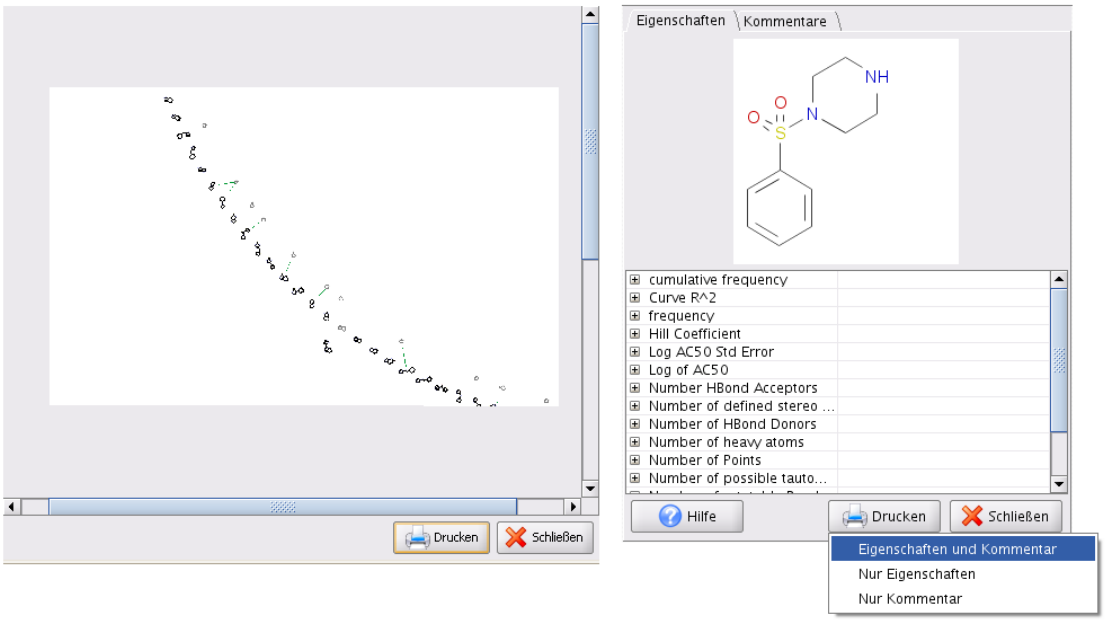

Abbildung 14: Vorschau- und Eigenschaftendialog, über die das Drucken angestossen wird

Eine der Anforderungen an den CBSE war es, dem Benutzer das Drucken sowohl des kompletten Graphen als auch des aktuell sichtbaren Ausschnitts des Graphen zu ermöglichen. Darüber hinaus sollte es ebenfalls möglich sein, die Eigenschaften und Kommentare zu einem Scaffold sowohl einzeln als auch zusammen auszudrucken. In der endgültigen Version des CBSE kann der Benutzer das Drucken des Graphen über das Menü des Hauptfensters auswählen, woraufhin der in Abbildung 14 links dargestellte Vorschaudialog erscheint. Dieser wird über die Klasse `PreviewDialog` implementiert und zeigt dem Benutzer eine Vorschau der gedruckten Version des Graphen. Das Drucken der Eigenschaften und Kommentare eines Scaffolds kann dagegen nur über den in der Klasse `PropertiesDialog` implementierten Eigenschaftendialog angestossen werden (siehe Abbildung 14 rechts).

Um diese unterschiedlichen Druckanfragen möglichst zentral zu verwalten, wurde die Klasse `PrintManager` geschrieben (siehe Abbildung 15), die das Interface `Printable` aus dem Paket `java.awt.print` implementiert. Die Klasse wurde so entwickelt, dass mit ihr alle bisher vorgesehenen Druckanfragen einfach zu realisieren sind und sie in Zukunft für weitere Anfragen problemlos erweitert werden kann.

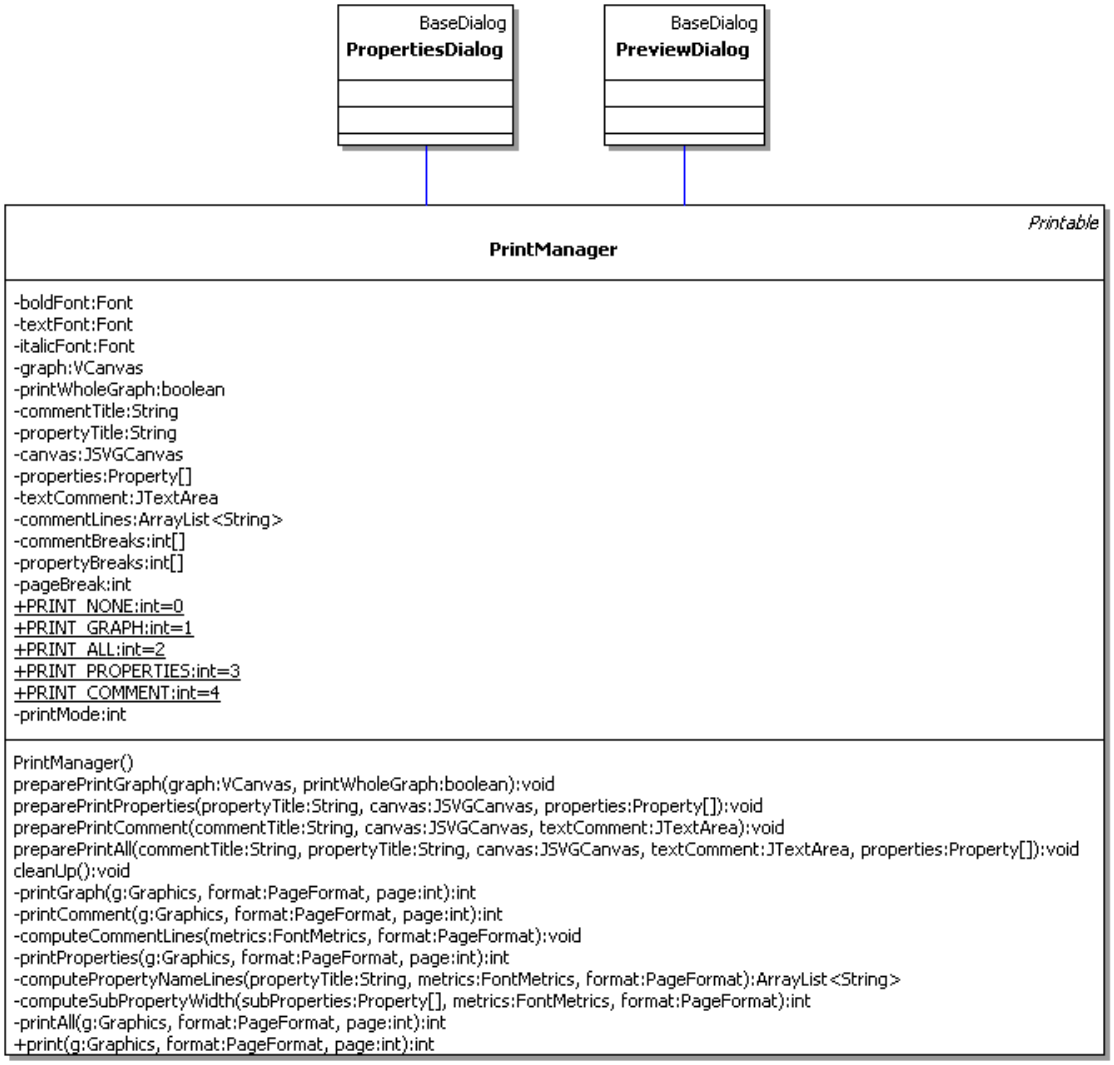

Abbildung 15: Überblick über die Klasse `PrintManager`

Für jede der vier realisierten Druckanfragen (Drucken des Graphen, Drucken der Eigenschaften, Drucken des Kommentars und Drucken der Eigenschaften und des Kommentars) gibt es eine `preparePrint`- und eine `print`-Methode. Über die `preparePrint`-Methoden werden dem `PrintManager` die zu druckenden Elemente übergeben und das `printMode`-Attribut auf den entsprechenden Wert gesetzt. Die jeweilige `preparePrint`-Methode muss aufgerufen werden bevor das Drucken über die Methode `print(Graphics graphics, PageFormat format, int page) : int` des `java.awt.print.PrinterJob`-Objekts, das den gesamten Druckvorgang kontrolliert, angestossen wird. Die den vier möglichen Druckanfragen entsprechenden `print`-Methoden sind private Methoden der Klasse und werden erst aufgerufen, nachdem das Drucken angestossen wurde. In diesen Methoden wird der auszudruckende Inhalt auf das übergebene `Graphics`-Objekt gezeichnet. Die Ausnahme bildet hier das Drucken des Graphen, da in diesem Fall nicht der `PrintManager` das Zeichnen übernimmt, sondern die Zeichenfläche `VCanvas`, auf der der Graph dargestellt wird. Dafür werden in der Klasse `VCanvas` zwei Schnittstellen angeboten (siehe auch 5.1):

- `exportPaintAll(Graphics g, Rectangle2D paintArea) : Rectangle2D`
  Zeichnet den gesamten Graphen in `g`.
- `exportPaintScreen(Graphics g, Rectangle2D paintArea) : Rectangle2D`
  Zeichnet den aktuell sichtbaren Ausschnitt des Graphen in `g`.

In der Methode `printGraph(Graphics g, PageFormat format, int page) : int` wird daher nur die entsprechende Schnittstellenmethode des `VCanvas` aufgerufen. Ob der Benutzer den gesamten Graph oder nur die aktuelle Ansicht des Graphen drucken möchte, wird dabei über den Wert des booleschen Attributs `printWholeGraph` entschieden, welches zuvor beim Aufruf der Methode `preparePrintGraph(VCanvas graph, boolean printWholeGraph) : void` gesetzt wurde.

Nachdem die Druckanfrage bearbeitet und das Drucken angestossen wurde, muss die Methode `cleanUp() : void` des `PrintManager` aufgerufen werden. Innerhalb dieser Methode werden die Referenzen auf die zu druckenden Elemente auf `null` gesetzt und das Attribut `printMode` bekommt den Wert `PrintManager.`*`PRINT_NONE`* zugewiesen.

3.1.5 Bookmarks

Der Nutzer kann für ihn wichtige Scaffolds als Favoriten ablegen und sie damit, ähnlich wie in einem Browser, jederzeit schnell wiederfinden. Die Favoritenverwaltung entspricht dem, was man aus einem Browser kennt: Den Favoriten lassen sich aussagekräftige Namen geben, der Benutzer kann sie mit Hilfe von Ordnern beliebig sortieren, ebenso lassen sie sich natürlich nachträglich umbenennen, verschieben und löschen (siehe Abb. 16). Die Favoriten werden zusammen mit dem Profil gespeichert.

Abbildung 16: Favoritenverwaltung

Die Funktion der Favoriten wird durch vier Klassen implementiert. Dies sind

- `Bookmark`
- `BookmarkTreeModel`
- `BookmarkTreeNode`
- `GroupTreeNode`

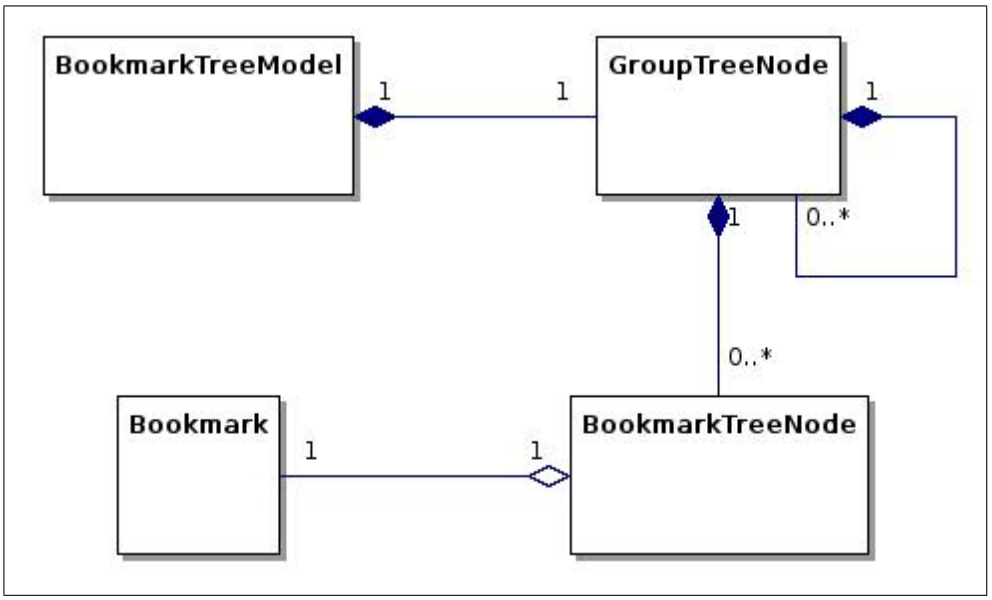

Abbildung 17: Klassendiagramm Bookmarks

Ein `BookmarkTreeModel` enthält als Wurzel einen `GroupTreeNode`. Ein `GroupTreeNode` kann weitere `GroupTreeNodes` oder `BookmarkTreeNodes` enthalten. Einem `BookmarkTreeNode` ist jeweils genau ein `Bookmark` zugeordnet (siehe Abb. 17). Somit lassen sich beliebige Baumstrukturen aufbauen.

Die Klassen bauen auf Javas Standardklassen für Bäume auf. `BookmarkTreeModel` erbt von `DefaultTreeModel`, `BookmarkTreeNode` und `GroupTreeNode` erben von `DefaultMutableTreeNode`. Dadurch hat man ohne Mehraufwand die Möglichkeit, die Baumstruktur zu durchlaufen oder in einem `JTree` anzuzeigen.

Die Klasse `Bookmark` speichert den Namen des Favoriten, der dem Benutzer angezeigt wird, sowie die Datenbank-ID des Scaffolds, auf das der Favorit verweist. Dabei ist der Name änderbar, die ID allerdings nicht. Die `Bookmark`-Objekte werden in den `BookmarkTreeNodes` gespeichert. Die `Bookmark`-Klasse ist so implementiert, dass sie von `AbstractAction` erbt und in ihrer `actionPerformed`-Methode den gerade angezeigten Graphen dazu veranlasst, auf das Scaffold mit der im `Bookmark` gespeicherten ID zu fokussieren. Dadurch lässt sich das Favoritenmenü einfach durch einen Durchlauf durch den Baum und das Übernehmen der gespeicherten `Bookmark`-Objekte in `JMenus` aufbauen. Somit müssen bei Änderungen der Favoriten nur die jeweils aktuellen `Bookmark`-Objekte ausgelesen und daraus das Menü aufgebaut werden, und nicht ständig neue `Action`-Objekte für das Menü erzeugt werden.

Die `Bookmark`-Klasse und ihr Verhalten ist das Einzige, was aus dem ersten Entwurf der Favoriten übernommen wurde (siehe Abb. 18). Der ursprüngliche Entwurf sah zwei Klassen `BookmarkGroup` und `Bookmark` vor. Dabei konnte eine `BookmarkGroup` sowohl weitere `BookmarkGroups` als auch `Bookmarks` enthalten. Dieses Design wurde aber wieder verworfen, da zur Anzeige der Favoriten in einem `JTree`, wie sie für die Favoritenverwaltung benötigt wurde, diese Art der Repräsentation erst in eine `TreeModel`-kompatible Form gebracht werden müsste. Ebenso müssten sämtliche Methoden zum Durchlaufen des Baumes selbst geschrieben werden. Wie schon weiter oben gesagt bekommt man diese Funktionalitäten mit dem neuen Design automatisch.

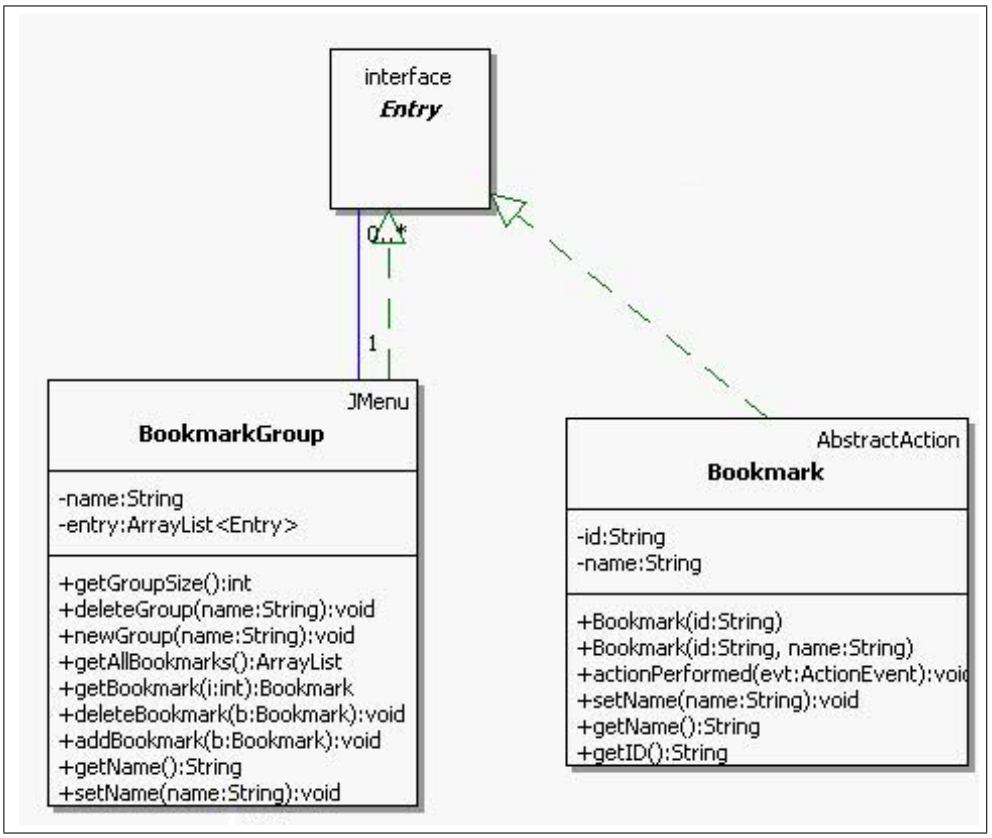

Abbildung 18: Erster Entwurf der Bookmarks

3.1.6 Export

Eine der Anforderungen an den CBSE war es, den Graphen als Bild exportieren zu können. Dabei sollten mehrere Formate angeboten werden. Unter anderem sollte es möglich sein, den Graphen als SVG zu exportieren. SVG ist ein Vektorgrafikformat und ermöglicht es, die exportierten Bilder verlustfrei auf jede gewünschte Größe zu skalieren. Somit ist es ein Leichtes, Ansichten des Graphen zur Illustration in Papern, Präsentationen oder auf Postern darzustellen.

Der CBSE bietet die Möglichkeit, entweder die aktuelle Ansicht des Graphen oder den kompletten Graphen zu exportieren (siehe Abb. 19).

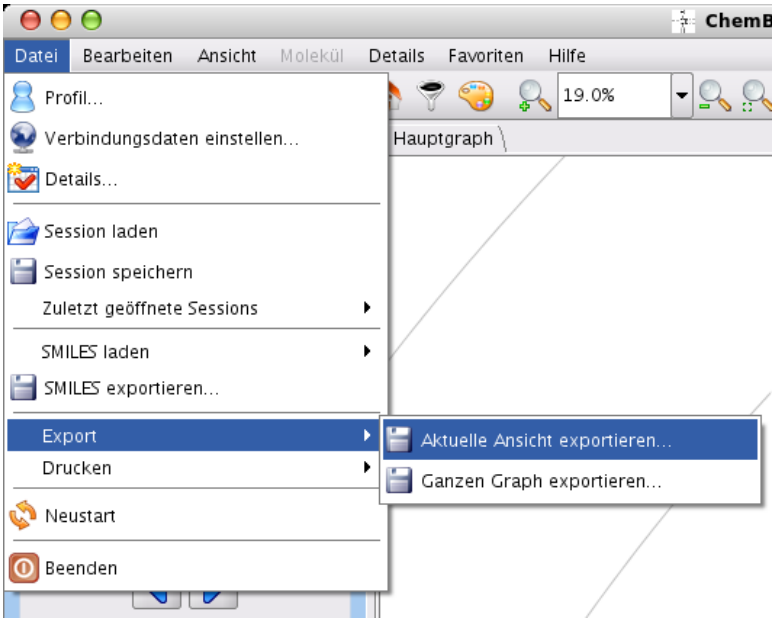

Abbildung 19: Exportmenü

Beim Export lassen sich die Größe des exportierten Bildes sowie der Bildtyp auswählen (siehe Abb. 20). Der CBSE unterstützt den Export als SVG, PNG und TIFF. Da für den Export die Transcoder-API des Batik-Toolkits genutzt wird, kann der CBSE prinzipiell in alle Formate exportieren, die von Batik unterstützt werden. Dazu müssten nur die `ImageType`-Enumeration und die `exportGraph`-Methode der `MainFrame`-Klasse angepasst werden, um dem CBSE weitere Formate bekannt zu machen. Auch ein Export in Formate, die Batik noch nicht unterstützt, wäre möglich, wenn man entsprechende Transcoder-Klassen für Batik implementiert.

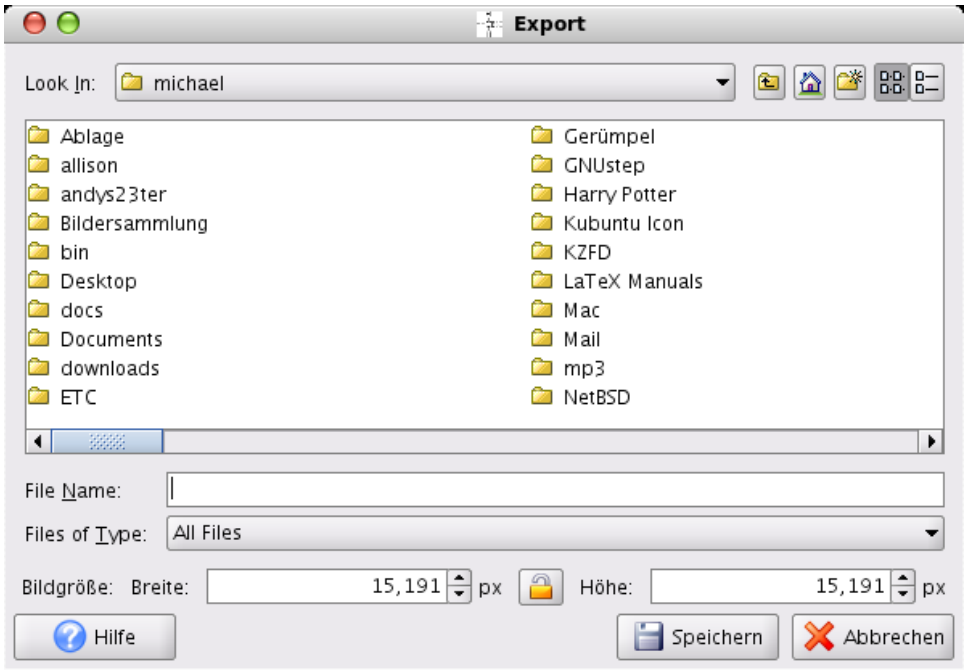

Abbildung 20: Exportdialog

Ein Problem beim Export stellt jedoch der Speicherbedarf dar. Besonders bei großen Graphen mit vielen Scaffolds kann es nötig werden, Java mehr Speicher zur Verfügung zu stellen, damit der Graph erfolgreich als Bild exportiert werden kann.

3.1.7 Filtern

Aufgrund der grossen Anzahl in der Datenbank vorhandener Scaffolds muss der Benutzer zu Beginn des Programms einen Filter einstellen, um die Menge der angezeigten Scaffolds zu verringern. Ein Filter besteht aus mehreren Regeln, die wiederum aus einer Scaffold-Eigenschaft, einem Wertetyp, einem Vergleichsoperator (<, =, >) und einem Wert bestehen. Der dafür vorgesehene Dialog erscheint nachdem der Benutzer sich mit seinem Profilnamen eingeloggt hat.

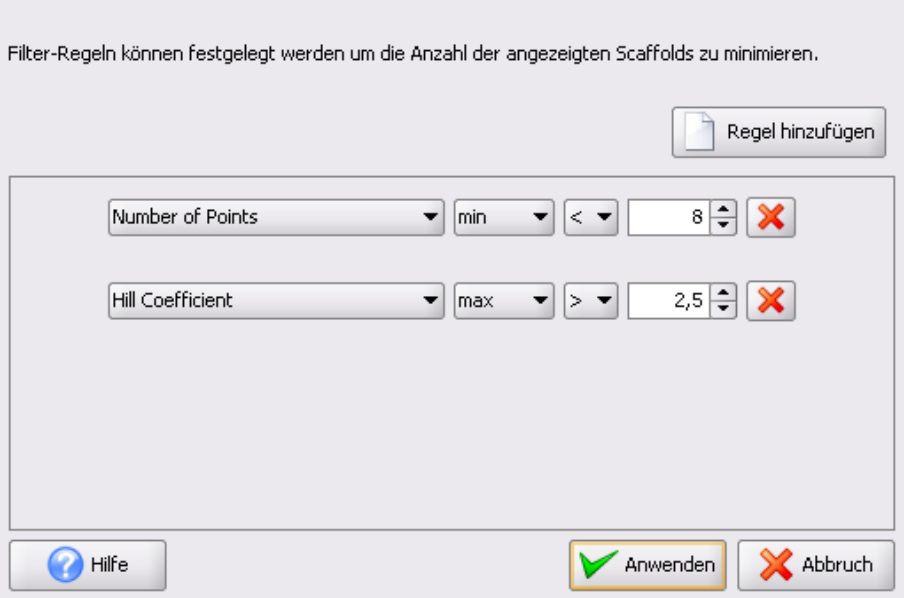

Abbildung 21: Dialog zum Einstellen des Filters am Ende des ersten Semesters

In Abbildung 21 ist der Dialog zum Einstellen eines Filters dargestellt, der zum Ende des ersten Semesters genutzt wurde. Der Benutzer hat hier über den Button „Regel hinzufügen“ die Möglichkeit dem Filter eine neue Regel hinzuzufügen. Ebenso kann er bereits bestehende Regeln editieren oder löschen. Die Anzahl der Regeln, aus denen ein Filter besteht, wird also vom Benutzer festgelegt. Soll eine Regel hinzugefügt werden, so kann in der ersten `JComboBox` die Scaffold-Eigenschaft eingestellt werden, über die gefiltert werden soll. Diese `JComboBox` wird dynamisch mit den Eigenschaften gefüllt, die in der Datenbank vorhanden sind. Es stellt daher kein Problem dar, sollte die Liste an Eigenschaften erweitert werden. In der zweiten `JComboBox` kann nun der Wertetyp und in der dritten der Vergleichsoperator eingestellt werden. Schließlich kann über einen `JSpinner` der Wert eingegeben werden.

Die auf diese Weise eingestellten Filter-Regeln werden an die Klasse `DBController` übergeben. Ausserdem werde sie im Profil gespeichert, damit der Benutzer nicht bei jedem Neustart zu einer erneuten Definition eines Filters gezwungen ist, sondern den Filter auf seinen alten Einstellungen basierend erweitern und verändern kann. Dazu wurde eine Datenklasse entwickelt, die eine Subklasse der bereits in Kapitel 3.1.3 beschriebenen Klasse `Detail` ist (siehe Abbildung 22).

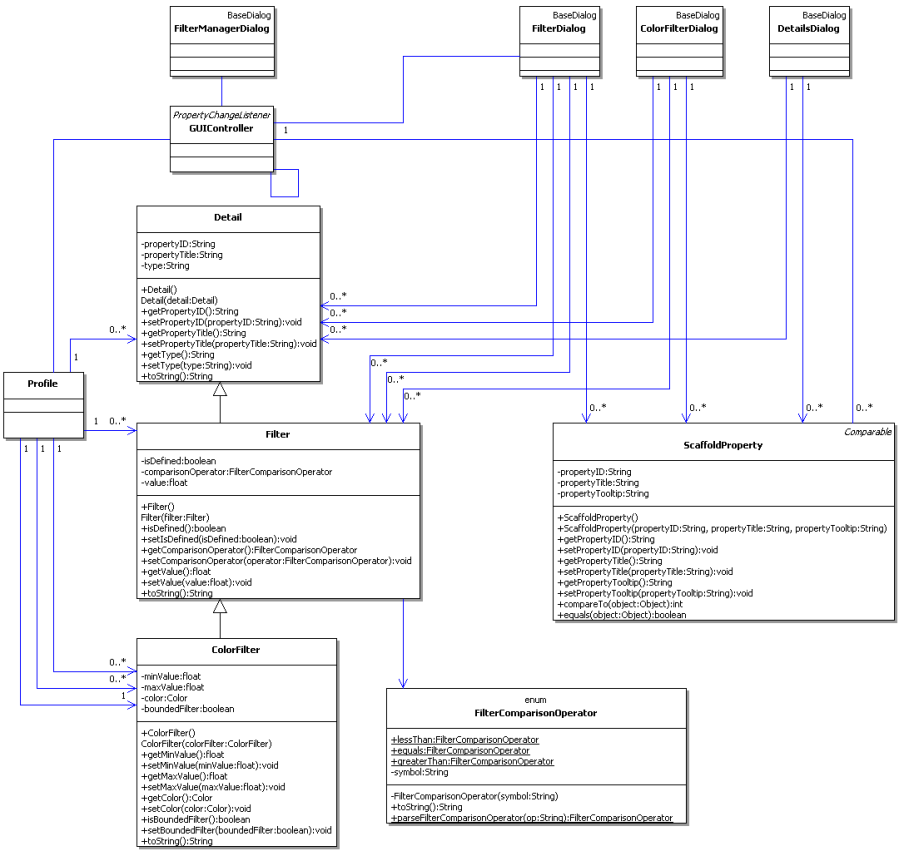

Abbildung 22: Aktuelles Klassendiagramm der Filterverwaltung

Neben den geerbten Attributen `propertyID`, `propertyTitle` und `type` besitzt die Klasse `Filter` die Attribute `operator` und `value`. Da in dem Attribut `operator` ausschließlich die Vergleichsoperatoren <, = und > gespeichert werden dürfen, wurde der `FilterComparisonOperator` entwickelt, der eine Aufzählung der drei Operatoren darstellt. Das Attribut `value` ist vom Typ `float`.

Jede Filter-Regel wird also durch ein Objekt vom Typ `Filter` gekapselt. Die Klasse `Profile` besitzt ein Attribut `initialFilter` vom Typ `ArrayList<Filter>` um den eingestellten Filter zu speichern. Diese `ArrayList<Filter>` wird auch zur Weiterverarbeitung an die Klasse `DBController` übergegeben.

Während des zweiten Semesters wurde der Dialog zum Einstellen eines Filters komplett überarbeitet. Von Betreuerseite wurde der Vorschlag geäussert, dem Benutzer in einem Dialog eine Liste aller in der Datenbank vorhanden Scaffold-Eigenschaften anzuzeigen und ihn diejenigen auswählen zu lassen, für die er eine Regel definieren möchte. Daraufhin wurde ein Dialog entwickelt, der den Benutzer in zwei Schritten durch den Prozess der Definition eines Filters führt. Zuerst bekommt der Benutzer eine Liste aller vorhandenen Scaffold-Eigenschaften angezeigt, die er jeweils über eine `JCheckBox` auswählen kann (siehe Abbildung 23). Nach dem Betätigen des „Vorwärts“-Buttons wird ihm zu jeder gewählten Scaffold-Eigenschaft eine Regel angezeigt, die er nun weiter editieren kann (siehe Abbildung 24). Dafür stehen ihm wie zuvor die `JComboBox` zur Angabe des Wertetyps und die `JComboBox` zur Angabe des Vergleichsoperators sowie der `JSpinner` zur Eingabe des Wertes zur Verfügung. Hinzugekommen ist eine `JCheckBox`, durch deren Auswahl der Benutzer alle Scaffolds in die Filtermenge einschließt, für die die gewählte Eigenschaft definiert ist. Ausserdem wurde jeder Regel ein „Plus“-Button hinzugefügt, mit dem es möglich ist, eine weitere Regel für die jeweilige Scaffold-Eigenschaft hinzuzufügen. Bei jeder Änderung, die an einer Regel vorgenommen wird, wird die Anzeige der Anzahl der sichtbaren Scaffolds aktualisiert. Der Benutzer kann

in seinem Profil eine maximale Anzahl sichtbarer Scaffolds einstellen. Wird diese überschritten, so wird die angezeigte Anzahl sichtbarer Scaffolds rot gefärbt und dem Benutzer wird zusätzlich eine Warnung angezeigt, sollte er das Programm dann starten wollen. Über den „Zurück“-Button kann der Benutzer wieder zu der Auflistung aller Scaffold-Eigenschaften gelangen. Die `JCheckBox`en der Eigenschaften, für die bereits Regeln definiert wurden, sind in dieser Auflistung dann bereits ausgewählt.

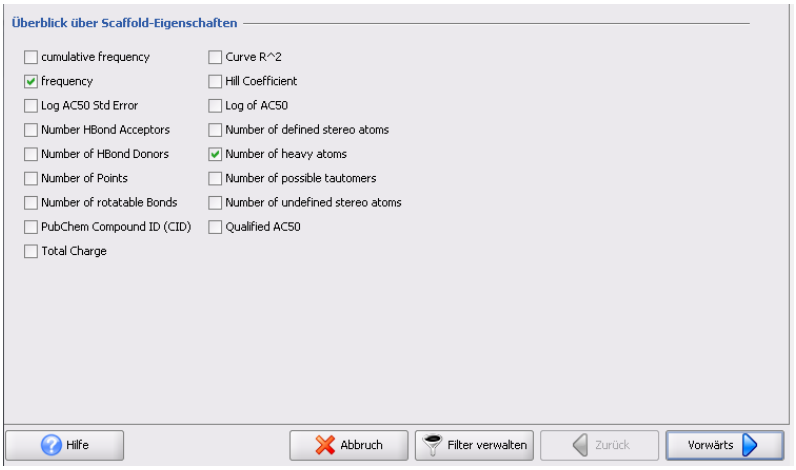

Abbildung 23: Erster Schritt des endgültigen Dialoges zum Einstellen des Filters

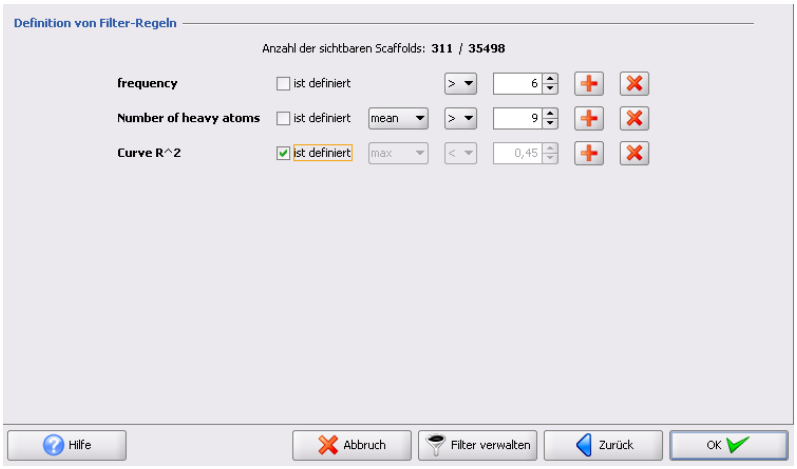

Abbildung 24: Zweiter Schritt des endgültigen Dialoges zum Einstellen des Filters

Im Zuge der Überarbeitung des Dialogs wurde die Datenklasse `ScaffoldProperty` entwickelt, die den Umgang mit den nötigen Daten für eine Scaffold-Eigenschaft vereinfacht. Eine Scaffold-Eigenschaft soll in der Auflistung aller Eigenschaften mit ihrem Titel aufgeführt sein und zusätzlich soll dem Benutzer mit Hilfe eines Tooltips eine kurze Erklärung zu der Eigenschaft angezeigt werden. Die Titel und Erklärungen sind in der Datenbank hinterlegt und können beim Aufbau des Dialogs ausgelesen werden. Ausserdem wird die in der Datenbank gespeicherte ID der Eigenschaft benötigt, da nach dem Auslesen des definierten Filters alle den Filter weiterverarbeitenden Methoden ausschließlich die ID der Eigenschaft nutzen. Die Klasse `ScaffoldProperty` besitzt drei `String`-Attribute, die genau diese Informationen speichern: `propertyID`, `propertyTitle` und `propertyTooltip`. Die Klasse `DBController` bietet die Methode `getScaffoldPropertiesDefinition() : ArrayList<ScaffoldProperty>` an, mit deren Hilfe beim Aufbau des Dialogs alle Scaffold-Eigenschaften mit den zugehörigen Informationen ausgelesen werden können. Damit der Benutzer eine bessere Übersicht über alle vorhandenen Scaffold-

Eigenschaften bekommt, sollte die Auflistung der Eigenschaften im ersten Schritt des Dialoges alphabetisch sortiert sein. Dies konnte einfach dadurch erreicht werden, dass die Klasse `ScaffoldProperty` das Interface `Comparable` aus dem Paket `java.lang` implementiert.

Wie bereits zuvor erklärt, stellt es kein Problem dar, sollte sich die Liste der Scaffold-Eigenschaften ändern, da diese bei jedem Dialogaufbau neu aus der Datenbank ausgelesen wird. Anders stellt sich die Situation bei der Liste der Wertetypen dar, die bis zur Mitte des zweiten Semesters fest in den Code einprogrammiert war und aus den fünf Wertetypen *mean*, *stddev*, *median*, *min* und *max* bestand. Da nicht für jede Scaffold-Eigenschaft für jeden dieser Wertetypen ein Wert hinterlegt sein muss, wurde in der Klasse `DBController` die Schnittstelle `getPropertyType(String numPropertyID) : String[]` angeboten, um die Liste der Wertetypen für eine Eigenschaft auszulesen. So kann beim Aufbau des Dialogs die aus der Datenbank übergebene Liste aller Scaffold-Eigenschaften durchlaufen und für jede Eigenschaft die zugehörige Liste der Wertetypen geholt werden. Jede `JComboBox` zur Einstellung des Wertetypen wird nun dynamisch mit den für eine Eigenschaft vorhandenen Wertetypen gefüllt und dem Benutzer wird nur noch eine sinnvolle Auswahlmöglichkeit geboten.

Mit einer ähnlichen Vorgehensweise werden die Grenzen des `JSpinner` gesetzt, über den der Wert einer Regel eingestellt werden kann. Die untere bzw. obere Grenze des `JSpinner` sollte dem in der Datenbank vorhandenen Minimal- bzw. Maximalwert für den Wertetyp der ausgewählten Eigenschaft entsprechen, damit der Benutzer nur Werte innerhalb des vorhandenen Wertebereichs einstellen kann. Hierfür stellt die Klasse `DBController` die Schnittstelle `getScaffoldMinMax(String numPropertyID, String numPropertyThing) : Float[]` zur Verfügung, mit der es möglich ist, die Grenzen des `JSpinner` beim Übergang in den zweiten Schritt des Dialogs zu initialisieren und bei jeder Änderung des Wertetyps zu aktualisieren.

Im Laufe des zweiten Semesters wurde das Programm um weitere Möglichkeiten der Filterung erweitert. So wurde zusätzlich zum bisher vorgestellten initialen Filtern das iterative Filtern ermöglicht. Dies erlaubt dem Benutzer die im Graphen dargestellte Menge an Scaffolds erneut zu filtern und auf diese Weise zu verkleinern. Das Konzept des initialen Filterns konnte an dieser Stelle fast komplett übernommen werden. Der Dialog für das iterative Filtern wird ebenfalls von der Klasse `FilterDialog` aufgebaut. Der einzige Unterschied besteht darin, dass es nur möglich sein darf, die Menge an Scaffolds zu verkleinern. Daher muss der für den Graph aktuell gültige Filter bekannt sein und berücksichtigt werden. Dies geschieht dadurch, dass für die aktuell gültigen Regeln nur noch der Wert verändert werden darf, d.h. nur der im `JSpinner` eingestellte Wert der Regel ist noch editierbar (siehe Abbildung 25). Auf diese Weise könnte das Vergrössern der Scaffoldmenge jedoch immer noch möglich sein, solange als Grenzen des `JSpinner` die in der Datenbank hinterlegten Minimal- bzw. Maximalwerte genutzt werden. Daher werden in diesem Fall nicht die aus der Datenbank übergebenen Werte als Grenze genutzt, sondern ein auf Grundlage der aktuellen Regel neu berechneter Wertebereich, der sicherstellt, dass die Menge an Scaffolds sich nicht vergrössert.

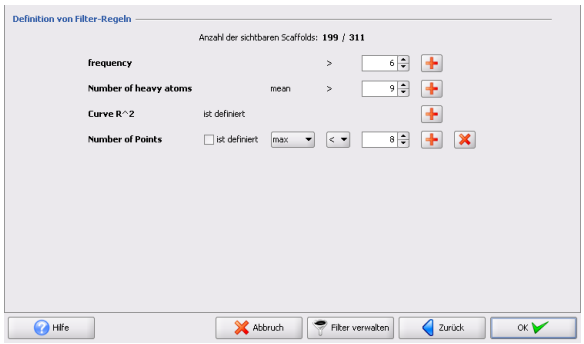

Abbildung 25: Zweiter Schritt des Dialoges zum iterativen Filtern

Im zweiten Semester wurde ebenfalls das Farbfiltern realisiert, wodurch es möglich wurde über die An-

gabe eines Filters die Knoten des Graphen einzufärben. Der Filter kann auf drei Arten eingestellt werden:

- die Angabe eines Standardfilters wie beim initialen Filtern
- die Angabe eines Filters mit definierten Grenzen und ohne Angabe eines Vergleichsoperators
- die Filterung über nur eine Scaffold-Eigenschaft, d.h. die Angabe nur einer Regel mit definierten Grenzen und ohne Angabe eines Vergleichsoperators

Auch hier konnte das Konzept des initialen Filterns fast komplett übernommen werden. Der Übersicht halber wurde eine eigene Klasse `ColorFilterDialog` zur Aufbau des nötigen Dialogs geschrieben. Dieser Dialog kann jedoch für die Angabe aller drei Arten von Filter wiederverwendet werden. Zur Datenspeicherung und Weitergabe des eingestellten Filters wurde die Datenklasse `ColorFilter` entwickelt, die von der Klasse `Filter` erbt (siehe Abbildung 22). Die zusätzlich eingeführten Attribute geben an, ob es sich um einen Filter mit definierten Grenzen handelt (`boundedFilter`), welche Werte für die definierten Grenzen eingestellt wurden (`minValue` und `maxValue`) und in welcher Farbe die Knoten eingefärbt werden sollen (`color`). Der für den Graphen aktuell gültige Filter wird auch hier berücksichtigt. Wählt der Benutzer eine Scaffold-Eigenschaft und einen Wertetyp, für die es in dieser Kombination im aktuell gültigen Filter eine Regel gibt, so werden als Grenzen für den `JSpinner` nicht die aus der Datenbank gelieferten Minimal- und Maximalwerte genutzt, sondern der Wertebereich wird derart neu berechnet, dass der Benutzer keine Werte einstellen kann, die auf Grund des eingestellten Filters auf keinen Scaffold zutreffen können.

Der Filter-Dialog wird in der endgültigen Version des CBSE nun nicht mehr nur zu Beginn des Programms angezeigt, sondern kann in seinen verschiedenen Formen immer wieder vom Benutzer aufgerufen werden. Wie bereits beschrieben wird bei jedem Aufbau eines Filter-Dialogs die Liste aller Scaffold-Eigenschaft samt zugehöriger Liste der Wertetypen aus der Datenbank ausgelesen, was den Aufbau der Dialoge stark verlangsamt. Daher wurde zum Ende der Entwicklungsphase entschieden, diese Informationen einmalig sofort nach dem Aufbau der Verbindung zur Datenbank auszulesen und während der Laufzeit des Programm im `GUIController` zu halten, was den Aufbau der Dialoge deutlich beschleunigte.

Schließlich wurde das Programm im Laufe des zweiten Semesters um eine Filterverwaltung erweitert. Über den „Filter verwalten“-Button können die aktuell im Filterdialog eingestellten Filter unter einem Namen gespeichert werden. Mit Hilfe eines Verwaltungsdialogs (siehe Abbildung 26) können diese gespeicherten Filter dann umbenannt, gelöscht oder in den entsprechenden Filterdialog geladen werden. Dies sollte dem Benutzer eine einfachere Handhabung mit den vom ihm bevorzugten Filtern ermöglichen.

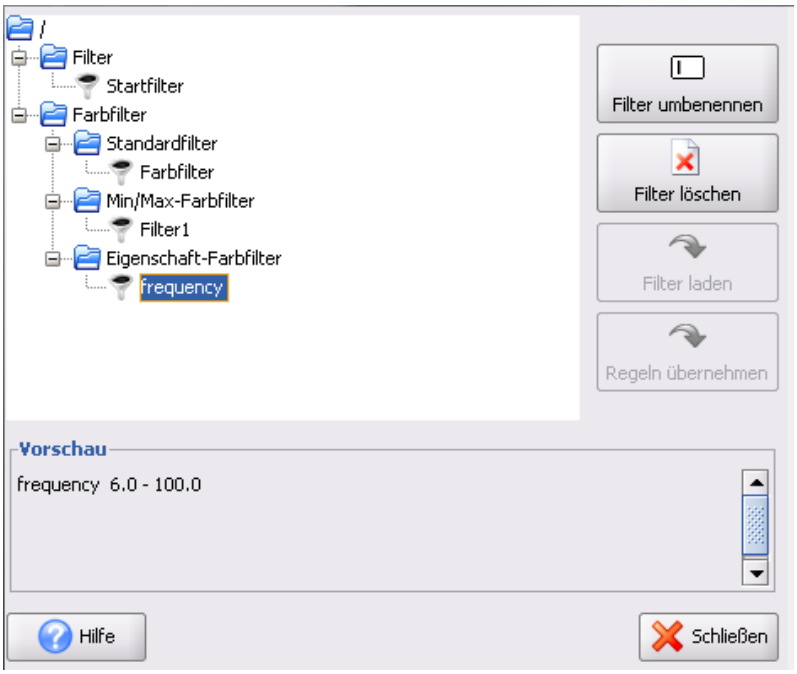

Abbildung 26: Dialog zum Verwalten der Filter

3.1.8 Datenspeicherung

Um sinnvoll mit dem CBSE zu arbeiten, müssen einige Informationen dauerhaft gespeichert werden. Das betrifft hauptsächlich die Daten des Profils. Aber auch Verbindungsdaten können lokal gespeichert werden.

Profil

Im Profil werden die Einstellungen des Benutzers, seine selbst erstellten Filter und seine Bookmarks abgelegt (siehe Abb. 13). Dabei gibt es zwei Möglichkeiten: Der Benutzer kann sein Profil in der Datenbank speichern, so dass es von jedem Rechner aus verfügbar ist. Die entsprechende Funktionalität ist im db-Package implementiert und wird hier nicht weiter behandelt. Alternativ kann der Benutzer sein Profil lokal speichern. Der Vorteil dabei ist, dass in diesem Fall dem Benutzer keine Schreibrechte für die Datenbank gegeben werden müssen.

Ursprünglich wurden die lokalen Profile mit Hilfe der Objektserialisierung von Java gespeichert. Da die alten Profile aber jeweils nach Änderungen an der Profilklasse nicht mehr geladen werden konnten, wurde nach einer besseren Lösung gesucht. In Folge dessen werden die Profildaten nun als XML-Dateien gespeichert. Die Verarbeitung der Profildateien wird von der Klasse `XMLProfile` übernommen. Diese Klasse enthält Methoden, die aus einem Profil-Objekt einen XML-Baum aufbauen und speichern, sowie aus den Daten eines eingelesenen XML-Baums ein Profil-Objekt erzeugen können.

Für die Erstellung und Verarbeitung der XML-Dateien wurde statt der in Java enthaltenen XML-API die einfacher zu verwendende JDOM-API[12] genutzt.

Verbindungsdaten

Damit Nutzer, die ihr Profil in der Datenbank speichern, nicht bei jeder Anmeldung die Verbindungsdaten für die Datenbank eingeben müssen, können Standard-Verbindungsdaten hinterlegt werden (siehe Abb. 27). Die Verbindungsdaten bestehen aus der URL des Datenbankservers, dem Datenbanknamen sowie dem Benutzernamen und Passwort, um auf die Datenbank zuzugreifen.

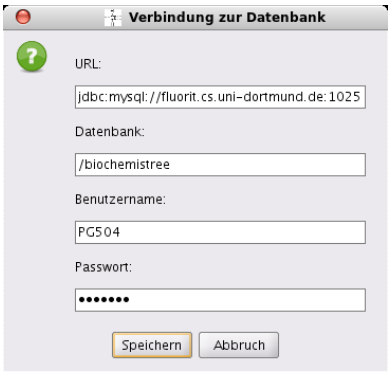

Abbildung 27: Verbindungsdaten

Auch diese Informationen werden lokal in einer XML-Datei gespeichert. Die Verarbeitung dieser Datei wird von der Klasse `XMLDBConnection` übernommen, die analog zu `XMLProfile` aufgebaut ist.

[12] http://www.jdom.org

3.2 Aus Sicht der DB

3.2.1 Aufbau der ursprünglichen Datenbank

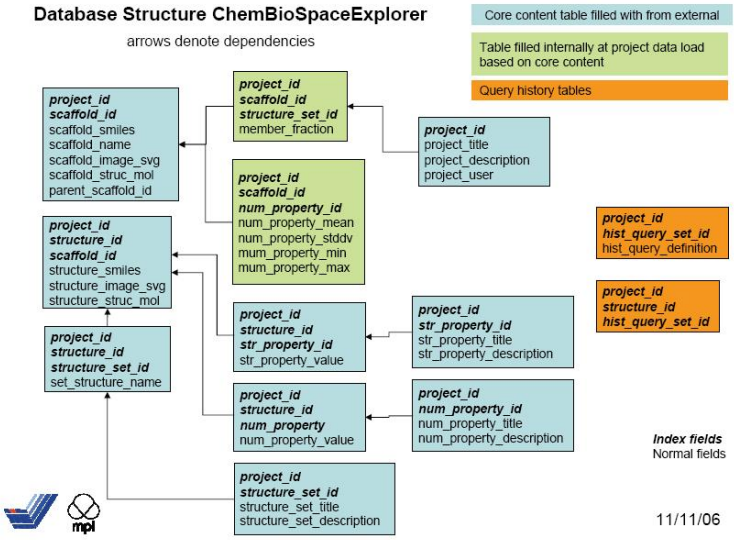

Abbildung 28: Der Aufbau der ursprünglichen Datenbank

Die Datenbank war ursprünglich wie in Abb 28 zu sehen ist aufgebaut. Die Datenbank wurde in MySQL implementiert. Am Anfang waren sehr viele Fehler in der Datenbank. Die Optimierung wird aber Bestandteil des nächsten Kapitels sein, also wird an dieser Stelle nicht weiter darauf eingegangen.

Die Datenbank hat im Wesentlichen zwei Teile. Der eine Teil beschäftigt sich mit den Scaffolds. Hier werden die Scaffolds samt ihren Eigenschaften gespeichert. In der Tabelle Scaffold_data stehen alle Scaffolds die in der Datenbank enthalten sind, die visualisiert werden sollen. Es werden in dieser Tabelle auch ein paar Eigenschaften gespeichert. Dazu zählen die Smile Strings und die SVGs. Die SVGs enthalten 2-dimensionale Darstellungen der Moleküle. Das Format ist hier wie der Name sagt ein SVG. Des Weiteren steht in dieser Tabelle dann noch der Elter dieses Moleküls. Es gibt, wie in der Abbildung zu sehen ist, zwei Tabellen, die sich nun auf dieses Tabelle beziehen. Zu beachten ist hier, dass die Datenbank keine Konsistenzprüfung macht. Der Datenbank Administrator hat hier also für einen sauberen Datenbestand zu sorgen. Die eine Tabelle beinhaltet die Eigenschaften. Zu jedem Molekül kann es also keine, eine oder mehrere Eigenschaften geben. Diese haben dann einen Namen und 5 Eigenschaften (Min, Max,...). Die zweite Tabelle enthält nun die Übersicht der verschiedenen Projekte. Dieses war dazu gedacht, dass die Datenbank nicht nur dem CBSE als Datenquelle dienen sollte. Die Tabelle wird von unserem Programm nicht befragt, also verzichten wir an dieser Stelle auf eine Erklärung.

Der zweite Teil der Datenbank beschäftigt sich nun mit den Strukturen, die jetzt hinter einem Scaffold stehen. Auch hier haben wir nun zwei Eigenschaften für jede Struktur. Die eine beinhaltet die Eigenschaften, die wir als Strings definieren, die andere die numerischen Eigenschaften. Die Details dazu können der Abbildung entnommen werden.

3.2.2 Optimierungen der ursprünglichen Datenbank

Zu Beginn der Entwicklung arbeitete die Gruppe mit einer Test-Datenbank. Sie folgte im strukturellen Aufbau den Vorgaben von Stefan Wetzel, enthielt aber lediglich Einträge, die zu Testzwecken generiert wurden. Diese Datensätze füllten nur einen kleinen Teil der möglichen Attributsmenge. In dieser ersten Phase sollten die bis zu diesem Zeitpunkt gewählten und entwickelten Darstellungsformen des chemischen Strukturraums getestet werden. In der Tabelle *scaffold_data* wurden Einträge eingefügt, die eine einfache Baumstruktur wiederspiegelten. Um die SVG-Implementierungen zu testen, wurden auch in diesem Bereich Einträge vorgenommen. Diese DB (Datenbank) wurde auf dem PG-Server am Lehrstuhl 11 aufgespielt und die Client/Server-Kommunikation in die Evaluierung mit aufgenommen.

In der ersten Januarwoche 2007 erhielt das Team die Datenbank von Stefan Wetzel. Diese enthielt in den Bereichen *scaffold_data* und *structure_data* mehrere tausend Einträge, auch ein Großteil der weiteren Tabellen waren mit Einträgen gefüllt. Sie stellte somit eine geeignete Basis für weiterführende Tests dar. Während die Testdatenbank ohne Komplikationen in Verbindung mit dem CBSE funktionierte, gab es bei der Umstellung auf die erste „richtige“Datenbank diverse Schwierigkeiten. Wurde versucht die Datenbank mit der Testdatenbank zu vertauschen, so benötigte die Ausführung des CBSE sehr lange (25 Minuten) - und führte nach dieser beträchtlichen Zeitspanne ohne erfolgreichen Start zu einem Absturz mit vorerst unbekannter Ursache. Nach längerer Suche fiel folgender inhaltlicher Fehler in der Datenbank auf:

Scaffolds hatten in der Tabelle *parent_child_relationship* IDs als Einträge, die gar nicht in *scaffold_data* existierten. Ergo suchte das Programm nach Daten, die gar nicht in der DB vorhanden waren. Diese Suche nach Geistereinträgen war der Grund, weshalb es nach einer gewissen Zeit zwangsläufig zum Absturz kam. Während das Auslesen der ersten Einträge reibungslos funktionierte, stieß das Programm ab einem gewissen Punkt der Suche auf einen Geisteintrag, der als Kinds-Knoten angegeben war. Die danach ausgeführte Query an die Datenbank lieferte als Ergebnis eine leere Menge zurück, welche bei der weiteren Verarbeitung, wie dem Auslesen des Namens, SVGs usw. zu einem Abbruch und einer Fehlermeldung in Form eines „operation on empty result set“ führte. Dieser Umstand ließ sich durch eine Änderung der Methoden zur Weiterverarbeitung beheben und komplett vermeiden. Eine Durchsicht der Datensätze wurde durch diesen Missstand angeregt.

Aufgrund der langen Startzeit wurde die Realisierung der Datenbank in MySQL genauer untersucht. Hätte es keine Probleme mit o.g. Geistereinträgen gegeben, dann hätte das Programm über 20 Minuten bis zum erfolgreichen Start benötigt - eine Zeitspanne, die inakzeptabel gewesen wäre. Es stellte sich heraus, dass als Verwaltungs-Engine die InnoDB-Instanz verwendet wurde. Der InnoDB-Tabellen-Handler ist eine sehr junge Ergänzung der mySQL-Familie. Es wurde unter dem Gesichtspunkt der Transaktionsverarbeitung entworfen und lehnt sich stark an Oracle an. Da die Datenbank keine Unterstützung zur Eingabe neuer oder Veränderung bestehender Tabellen-Einträge zur Verfügung stellen musste, konnte auf die Vorteile dieser Engine verzichtet werden. Dadurch konnte man sich auch der Nachteile entziehen: Eine geringe Lese-Performanz. Die MyISAM-Engine wurde verwendet, denn als Standard-Storage-Engine von mySQL stellt sie einen guten Kompromiss zwischen Geschwindigkeit und nützlichen Features dar. MyISAM-Tabellen stellen keine Transaktionen und auch kein sehr feines Locking-Modell zur Verfügung, bieten dafür aber eine Volltextindizierung.

Als eine weitere gravierende Performanz-Bremse stellte sich heraus, dass keine der angelegten Tabellen primary keys verwendeten. Es wurden Schlüssel zu den IDs der jeweiligen Tabellen angelegt. In Kombination mit der geänderten Datenbank-Engine, der Verwendung von Schlüsseln, als auch der robusteren Abfragemethoden benötigte ein erstes erfolgreiches Zusammenspiel zwischen dem Programm und der Datenbank nunmehr eine Startzeit von fünf Minuten.

Durch diesen Erfolg angespornt, wurde nach weiteren Optimierungsmöglichkeiten Ausschau gehalten - nun mit Augenmerk auf den Sourcecode erstellter Methoden. Um wiederkehrende Datendankabfragen zu vermeiden, wurden die Scaffolds, sobald sie einmal aus der Datenbank geladen wurden, in einer Hashtabelle abgelegt. Es stellte sich heraus, dass die Visualisierung selbst sich um die Weiterverarbeitung und

Speicherung der Datensätze kümmert - die Hashtabelle verursachte somit einen doppelten Aufwand und konnte demnach entfernt werden. Diese Massnahme verbesserte die Startzeit um dreissig Sekunden. Zum damaligen Zeitpunkt war die Datenstruktur Scaffold derart aufgebaut, dass sie die Kinder eines angeforderten Knotens gleich mit aus der Datenbank lud. Dies erzeugte einen immensen Overhead - Daten wurden „auf gut Glück“ angefordert, ihre tatsächliche Verwendung stand noch in Frage. Liess sich der Anwender beim ersten Start des Programms die erste Hierarchieebene anzeigen und legte sein Augenmerk im weiteren Verlauf auf einen ganz bestimmten Ast innerhalb dieser Menge, so konnte ein bereits geholtes Scaffold zur Anzeige gebracht werden; im Gegensatz dazu blieben die knapp 700 anderen Scaffolds, die schon aus der Datenbank geholt worden waren, auf der ersten Ebene unbenutzt bzw. nicht dargestellt. Die Scaffold-Struktur wurde abgeändert, Kind-Knoten wurden nur durch ihre scaffold-ID und nicht mehr durch eine komplette Scaffold-Instanz repräsentiert. Aus dieser Optmierung konnte ein Performancegewinn von zwei Minuten gewonnen werden, die Startzeit betrug nun etwas mehr als zwei Minuten.

Hinsichtlich der SVG-Abfrage bzw. deren Generierung zur Startzeit erhoffte sich die Gruppe eine weitere Möglichkeit die Startzeit zu verkürzen und auf ein adäquates Zeitfenster in Bezug auf die Usability zu reduzieren. Es sollte ein allzu langes Warten auf die erste Darstellung des chemischen Strukturraums vermieden werden, wobei hier keine explizite Obergrenze genannt wurde. Die SVGs der Scaffolds wurden zuerst als Text aus der Datenbank gelesen und zur Laufzeit generiert. Eine Anzeige aller SVGs auf einem Bildausschnitt ist einerseits durch die hohe Anzahl der Scaffolds auf der ersten Ebene und andererseits durch den Zoomfaktor, bei dem von der SVG zu einer verallgemeinerter Darstellung durch grau eingefärbte Kästen umgeschaltet wird, nicht möglich. Diese Tatsache wurde ausgenutzt, indem das SVG eines Scaffolds nicht bei seiner Konstruktion generiert wurde, sondern erst, wenn das SVG auch tatsächlich auf dem Bildschirm dargestellt wird. Der Ansatz des Auslagerns wurde für eine kurze Zeit noch weitergetrieben: Bei der Erzeugung des Scaffolds wurden keinerlei Daten bzgl. seines SVGs aus der Datenbank geladen. Dies führte jedoch zo einem nicht befriedigenden Ergebnis. Auf der einen Seite wurden 15 Sekunden bei der Abfrage der Scaffold-Einträge vom mySQL-Server eingespart, andererseits gab es bei der Benutzung des CBSE „Denksekunden“. Sobald ein SVG gebraucht wurde, musste eine weitere Datenbankabfrage gestartet werden - durch die daraus entstehende Zeitspanne hinkte die Darstellung der Interaktion des Anwenders hinterher. Die Entwicklung in dieser Richtung wurde somit ein Stück weit zurückgenommen, dank der verbleibenden Änderungen wurde erstmals ein Start unter einer Minute, nämlich mit akzeptablen dreissig Sekunden, möglich.

Es sei zum Schluss darauf hingewiesen, dass die Zeitangaben in diesem Textabschnitt nicht repräsentativ zu verstehen sind. Es wurden Messungen auf einem einzigen Computer vorgenommen, dieser war teils durch eine DSL-Leitung mit dem mySQL-Server verbunden, teils per Wireless LAN. Zudem liefen weitere Prozesse im Hintergrund - es wurde jedoch darauf geachtet, dass die Internetverbindung zum grösstmöglichen Teil der Client/Server-Verbindung zur Verfügung stand. Die Zeiten wurden zudem auf- bzw. abgerundet. Bei der Generierung der SVGs ist die Rechenleistung des Systems die wichtigste Kenngrösse - bei den Messungen stand ein Computer mit einem Centrino M mit 1,5 GHz und 512 MB Arbeitsspeicher zur Verfügung. Ein anders ausgestattetes System würde in diesem Bezug sicherlich zu unterschiedlichen Messergebnissen führen. Nichtsdestotrotz spiegeln die Zeitangaben die verschiedenen Einsparungspotentiale wieder, die durch die Veränderung im Laufe der Entwicklung möglich waren.

3.2.3 Speicherfunktion für die Datenbank

Eine der Anforderungen an den CBSE war, dass Profile in der Datenbank gespeichert werden können. Diese Funktion wurde erst während des 2. Semesters implementiert. Da während der Entwicklung die Profile stetig erweitert wurden, musste die Speicherfunktion immer miterweitert werden und war deswegen erst in einer späten Phase komplett fertig.

3.2.3.1 Realisierung

Zu Beginn der Realisierung entstanden zwei Ideen, wie man die Speicherfunktion in der Datenbank implementieren kann. Eine Idee ähnelte sehr der ersten Speicherfunktion auf der Festplatte. Die zu speichernde Objekte wurden in einen String geschrieben und dann auf der Festplatte abgespeichert. Die Speicherfunktion in der Datenbank sollte demnach genauso funktionieren nur dass dieser String in eine Datenbank geschrieben werden sollte. Im weiteren Verlauf der Entwicklung von CBSE stellte sich diese Speichermöglichkeit als nachteilig heraus. Sobald die Version des Programms geändert wurde, waren die gespeicherten Objekte nutzlos und somit die gespeicherten Profile. Man musste immer wieder alte Profile löschen und neue erstellen.

Die zweite Idee zur Speicherfunktion bestand darin, alle Variablen, die im Profil existieren, einzeln zu speichern und beim Neustart ein leeres Profilobjekt mit diesen Variablen zu füllen. Diese Möglichkeit die Profile zu Speichern ist viel robuster, aber auch viel aufwändiger. Ausserdem musste bei jeder neu hinzugekommene Variable, die Speicherfunktion um diese Variable erweitert werden, aber wenigsten funktionierte es zu jeder Zeit.

Die Implementierung verlief problemlos. Die GUI-Gruppe bot für alle Variablen im Profil get- und set-Methoden an und damit konnten alle Werte in den Variablen gespeichert und geladen werden. Nun mussten nur noch neues Schema und neue Tabellen in der Datenbank erstellt werden. Dazu wurde die Verbindung zur Datenbank angepasst um jedes Schema getrennt nutzen zu können. Es mussten noch neue Routinen zum Speichern, Löschen, Ändern der Profile in der Datenbank geschrieben werden, dann war die Speicherfunktion funktionsfähig. Bis zum Ende des Projekts wurde die Funktion immer wieder um neue Variablen erweitert, und somit auf dem neuesten Stand gehalten. Beim Abschluss des Projekts war es komplett.

3.2.3.2 Aufbau der Datenbank CBSE_Profiles

Diese Datenstruktur ist dafür gemacht alle benötigten Variablen eines Profils zu erfassen. Weiter unten stehende Tabellen sind auf dem letzten Stand seit dem Entwicklungsende von CBSE.

Tabellen:

Tabelle profile_data speichert einfache Daten eines Profils wie numerische oder boolsche Variablen, oder Namen der Datenbankverbindungen und ähnliches.

Name	Variablentyp
profile_name	VARCHAR
language	VARCHAR
url_database	VARCHAR
user_name_database	VARCHAR
password_database	VARCHAR
start_rings	INTEGER
home_id	INTEGER
render_quality	INTEGER
focus_after_action	BOOLEAN
layout_animation	BOOLEAN
camera_animation	BOOLEAN
project_id	VARCHAR
max_number_scaffolds	INTEGER
grid_rows	INTEGER
grid_columns	INTEGER
cursor_animation	BOOLEAN
max_subtree_size	INTEGER
stroke_width	FLOAT
selected_stroke_width	FLOAT
database_scheme	VARCHAR
edge_stroke_width	DOUBLE
selected_edge_stroke_width	DOUBLE
selected_node_color	INTEGER
edge_color	INTEGER
selected_edge_color	INTEGER
cursor_stroke_width	DOUBLE
cursor_color	INTEGER
circle_stroke_width	DOUBLE
circle_color	INTEGER
hide_subtree_edges	BOOLEAN

Tabelle 3: profile_data

Die Tabelle comment_data speichert die Kommentare, die hinter verschiedenen scaffolds hinterlegt sind. Sie enthält entsprechende Spalten um jedes Kommentar einem Bestimmten scaffold unter einem bestimmten Profil zuordnen zu können.

Name	Variablentyp
profile_name	VARCHAR
scaffold_id	INTEGER
comment	TEXT
project_id	VARCHAR

Tabelle 4: comment_data

Die Tabelle filter_data speichert für jeden Profil die Filtereinstellungen, die aktuell eingestellt sind und die, die früher gespeichert worden waren.

Name	Variablentyp
profile_name	VARCHAR
property_id	VARCHAR
property_title	VARCHAR
type	VARCHAR
operator	VARCHAR
value	FLOAT
project_id	VARCHAR
is_defined	BOOLEAN

Tabelle 5: filter_data

Die Tabelle bookmark_data speichert alle Favoriten jedes Profils und die dazugehörige Baumstruktur.

Name	Variablentyp
profile_name	VARCHAR
bookmark_name	VARCHAR
scaffold_id	INTEGER
folder	VARCHAR
project_id	VARCHAR

Tabelle 6: bookmark_data

Die Tabelle detail_data speichert Eigenschaften und Eigenschaftstypen, die im Detailfenster angezeigt werden.

Name	Variablentyp
profile_name	VARCHAR
property_id	VARCHAR
property_title	VARCHAR
type	VARCHAR
project_id	VARCHAR

Tabelle 7: detail_data

Die Tabelle color_filter_data speichert für den jeweiligen Profil die entsprechende Färbefilter. Je nach Filtertyp werden unterschiedliche Spalten Beschrieben.

Name	Variablentyp
profile_name	VARCHAR
filter_name	VARCHAR
property_id	VARCHAR
property_title	VARCHAR
type	VARCHAR
operator	VARCHAR
value	FLOAT
min_value	FLOAT
max_value	FLOAT
color	INTEGER
project_id	VARCHAR
is_defined	BOOLEAN

Tabelle 8: color_filter_data

Die Tabelle session_data speichert alle zuletzt geöffnete Sessions.

Name	Variablentyp
profile_name	VARCHAR
path	VARCHAR

Tabelle 9: session_data

3.3 Aus Sicht der VIS

3.3.1 Layouts

Eine der Hauptaufgaben der Visualisierungsgruppe bestand darin, die ausgewählten Daten in einem geeigneten Layout darzustellen. Da der Benutzer Filterregeln beim Programmstart einstellt, variiert die Anzahl der darzustellenden Knoten von einigen Dutzend bis zu mehr als tausend Knoten. Deshalb musste ein platzsparendes und übersichtliches Layout entwickelt werden, dessen Qualität unabhängig von der Anzahl der Knoten ist. Hauptschwierigkeit dabei war es, ein Layout fÃ¼r sehr viele Knoten zu generieren, vor allem bei vielen Knoten auf der ersten Hierarchieebene. Die darzustellenden Daten sind als Menge von Teilbäumen strukturiert, diese Struktur galt es bei der Wahl des Layout zu beachten und darzustellen.

Der CBSE unterstützt 3 Layouts: Radial Layout, Balloon Layout, Linear Layout.

3.3.1.1 Radial Layout

Die Grundlage für das radiale Layout ist ein Baumdiagramm, das von Stefan Wetzel im Rahmen der Entwicklung des SCONP (Structural Classification Of Natural Products) erstellt worden ist. Dieses Baumdiagramm stellt eines Ausschnitts des Strukturraums dar, bei dem die Grundstrukturen auf einem Kreis angeordnet sind und die Strukturen auf den tieferen Hierachieebenen auf konzentrischen Kreisen um diesen Kreis liegen. Diese statische Grafik musste aufwendig manuell erstellt werden. Im Zusammenhang mit der SCONP ist dieses Baumdiagramm unter den Chemikern sehr bekannt, so dass ein Wiedererkennungswert garantiert ist.

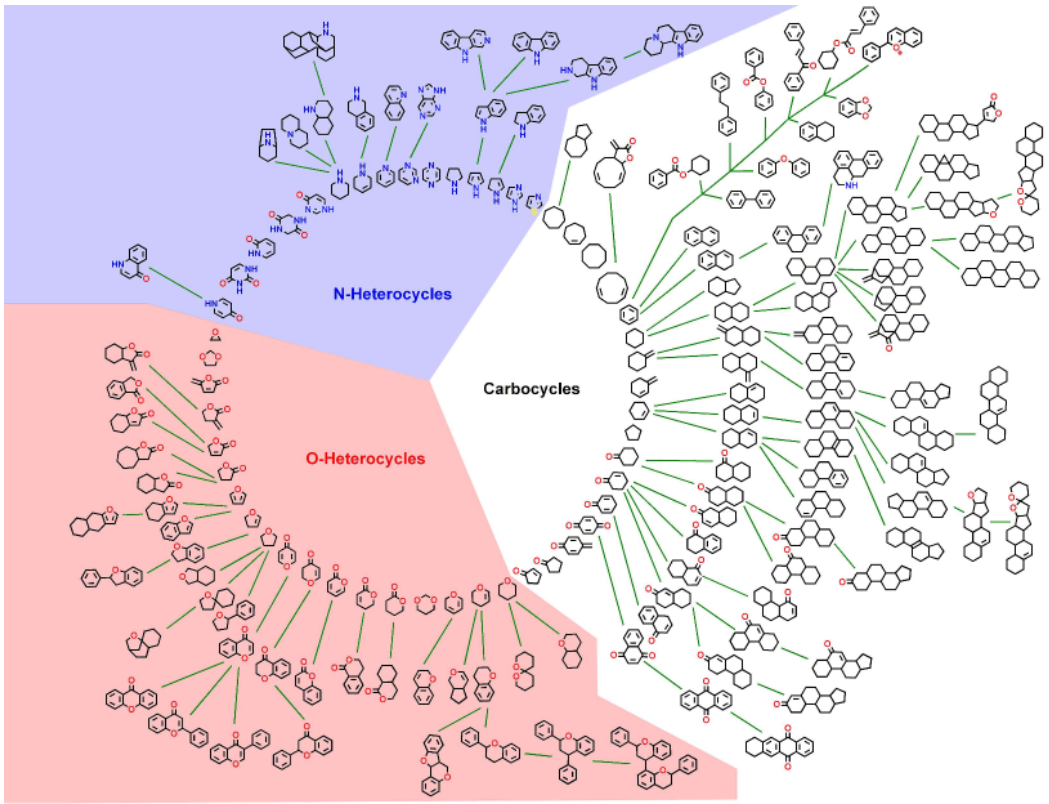

Abbildung 29: Das manuell erstellte Baumdiagramm der strukturellen Klassifizierung von Naturstoffen SCONP

Eine Anforderung an den CBSE und damit auch an das Layout war es, aus beliebigen Daten in der Datenbank eine ähnliche Abbildung automatisch zu erzeugen und in ein Grafikformat zu exportieren, das

für großformatige Ausdrucke geeignet ist.

Wie bereits erwähnt, liegen die Strukturen auf konzentrischen Kreisen, die um den Kreis der Grundstrukturen angeordnet sind. Dies erinnert schon sehr an die in der Graphentheorie weit verbreiteten radialen Layouts. Für diese Layouts gibt es bereits einige sehr effiziente Algorithmen. Radiale Layouts bieten eine gute Übersicht über die Tiefe der Knoten. Es lässt sich also intuitiv erkennen, auf welcher Ebene ein bestimmter Knoten liegt. Zudem lassen sich die Nachbarschaftsbeziehungen zu anderen Knoten sehr gut überblicken. Aus diesem Grund wurde einer dieser Algorithmen als Grundlage für das radiale Layout des CBSE genommen.

Die Daten in der Datenbank entsprechen einer grossen Menge von Bäumen, die nicht zusammenhängend sind. Die Wurzeln der einzelnen Bäume, die Grundstrukturen, sollen auf dem innersten Kreis angeordnet werden. Um diese Darstellung mit einem Algorithmus für ein radiales Layout zu erreichen, wird eine virtuelle Wurzel erzeugt, an die die tatsächlichen Wurzeln angehängt werden können. So erhält man einen zusammenhängenden Baum, dessen Wurzel in der Mitte platziert werden kann. Die Kinder dieser neuen Wurzel (die Grundstrukturen) werden so auf dem Kreis der ersten Hierarchieebene um die Wurzel platziert. Die virtuelle Wurzel und ihre Kanten zu den tatsächlichen Wurzeln werden später ausgeblendet.

Das radiale Layout basiert auf einem Layoutalgorithmus von Peter Eades[13]. Im Laufe der Projektgruppe wurde dieser Algorithmus den speziellen Anforderungen des ChemBioSpace Explorers angepasst.

In der ursprünglichen Version von Peter Eades weist der Algorithmus rekursiv jedem Knoten einen Radius und ein Teilstück des Kreises zu, in das der Knoten seine Kinder zeichnen kann. Dieses Kreisstück verteilt der Knoten wiederum gleichmässig auf seine Kinder. Dadurch erhält man die Polarkoordinaten aller Knoten. Diese werden später in kartesische Koordinaten umgerechnet, um die Knoten auf der Ebene zeichnen. Dieser Algorithmus konnte aber nicht ohne Modifikationen übernommen werden.

Zu Beginn der Entwicklung des Layouts waren alle Knoten SVGs mit einer Größe von 200 x 200px. Durch diese Knotengröße kam es zu Überlappungen zwischen Nachbarknoten innerhalb einer Hierarchieebene. Um dies zu vermeiden, muss der Radius der jeweiligen Hierarchiebene vergrössert werden. Das Problem ist, dass der Abstand zwischen zwei Hierarchiebenen im Algorithmus von Peter Eades konstant ist. Ein zu klein gewählter Radius führt, wie bereits erwähnt, zu Knotenüberlappungen und ein zu gross gewählter Radius führt zu einem Layout, mit einer schlechten Ausnutzung der verfügbaren Layoutfläche. Es werden also dynamische Radien benötigt, die sich an die aktuelle Struktur des Graphen anpassen. Dies wurde erreicht, indem ein Radius berechnet wird, der einen bestimmten Abstand zwischen zwei Knoten garantiert. Natürlich sollte dieser Abstand grösser sein, als die Grösse der SVGs. Ausserdem muss ein minimaler Abstand zwischen den benachbarten Kreisen berücksichtigt werden, so dass es nicht zu Knotenüberlappungen zwischen zwei benachbarten Hierarchieebenen kommt.

Im zweiten Semester der Projektgruppe wurde die Grösse der SVGs umgestellt, so dass es jetzt sowohl grössere als auch kleinere SVGs als 200 x 200 px gab. Dazu wurde der Layoutalgorithmus erneut modifiziert. Zu Beginn des Algorithmus wird die maximale SVG-Grösse auf jeder Ebene berechnet und der Mindestabstand mindestens so gross gewählt, wie dieser Wert.

Ein weiteres Problem bei der Entwicklung des radialen Layouts war die Anzahl der Scaffolds auf dem innersten Ring. Oft müssen mehrere hundert Scaffolds bereits auf dem innersten Kreis platziert werden, so dass der erste Radius sehr gross gewählt werden muss, um Knotenüberlappungen auf der ersten Ebene zu vermeiden. Hierbei ist es aber ein Vorteil, dass die imaginäre Wurzel ausgeblendet wird und für den Benutzer keine Information enthält. Wäre dies nicht so, hätte man Probleme aufgrund des grossen ersten Radius einen Teilbaum von der (zentralen) Wurzel bis zu seinen Kindern gleichzeitig auf dem Bildschirm darzustellen. Aber da die tatsächliche Wurzel eines Teilbaums ja bereits auf der ersten Hierarchieebene liegt, spielt die Grösse des erste Radius keine Rolle. Ausserdem stellt der CBSE viele Filterfunktionen

[13] Peter Eades. Drawing free trees. Bulletin of the Institute of Combinatorics and its Applications, 5:10-36, 1992.

bereit, um die Komplexität des chemischen Strukturraums hinreichend zu verringern und somit auch die Anzahl der Scaffolds auf dem innersten Ring anzupassen.

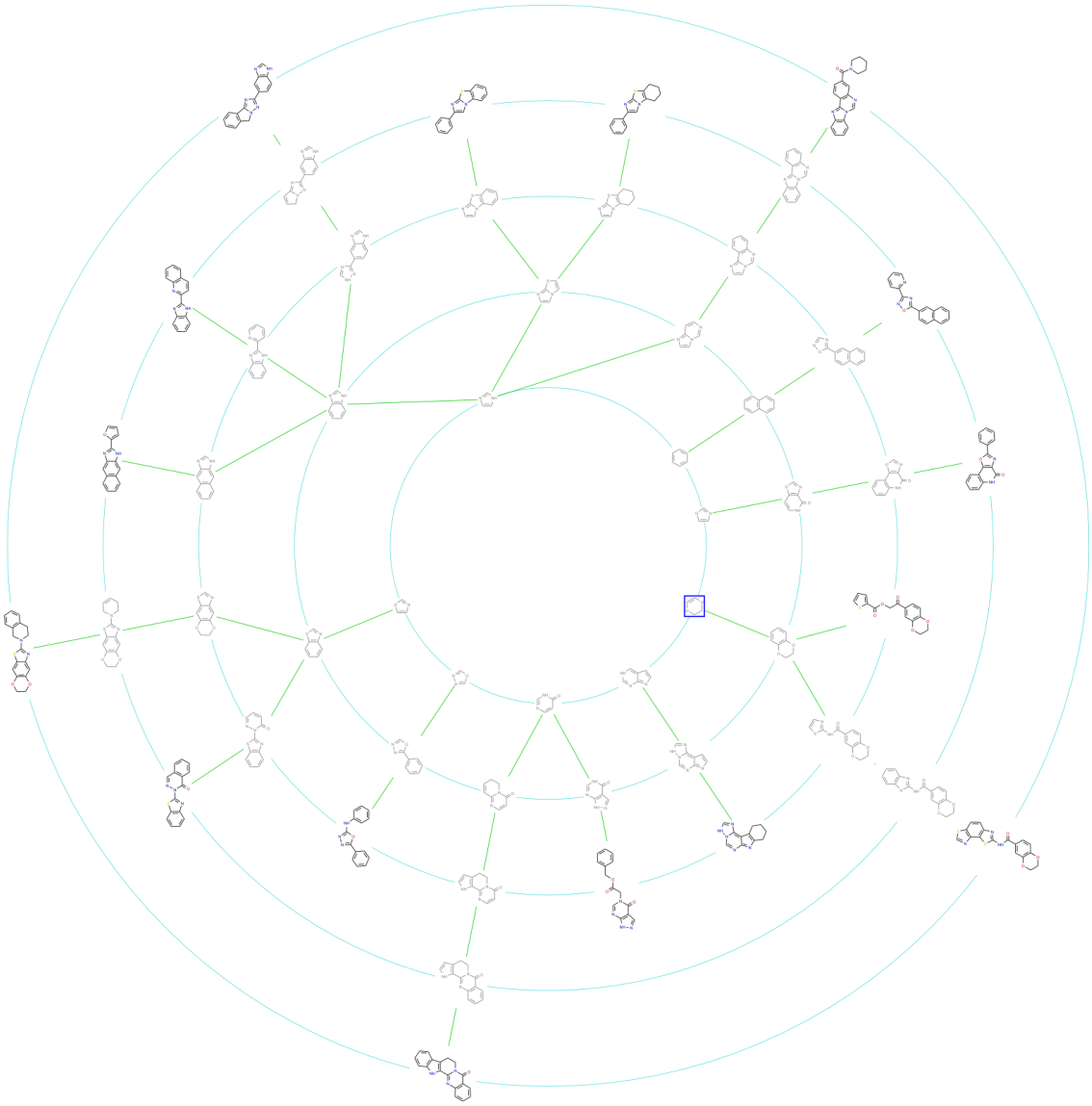

Abbildung 30: Ein automatisch erstelles Baumdiagramm des CBSE mit acht Strukturen auf dem innersten Ring

Hierbei gibt es auch die Funktion sich einen Teilbaum gesondert anzeigen zu lassen. Teilbäume werden dabei komplett angezeigt, mit der Ausnahme, dass die Kanten von der Wurzel zu den Kindern ausgeblendet werden. Bei Teilbäumen mit vielen Kindern auf der ersten Ebene wirkten diese Kanten sehr störend, da sie in einigen Fällen Moire-Muster erzeugten. Die freigewordene Fläche wird ausgenutzt, indem die Wurzel des Teilbaumes vergrössert dargestellt wird. Hierbei macht sich besonders die Skalierbarkeit der SVGs bezahlt.

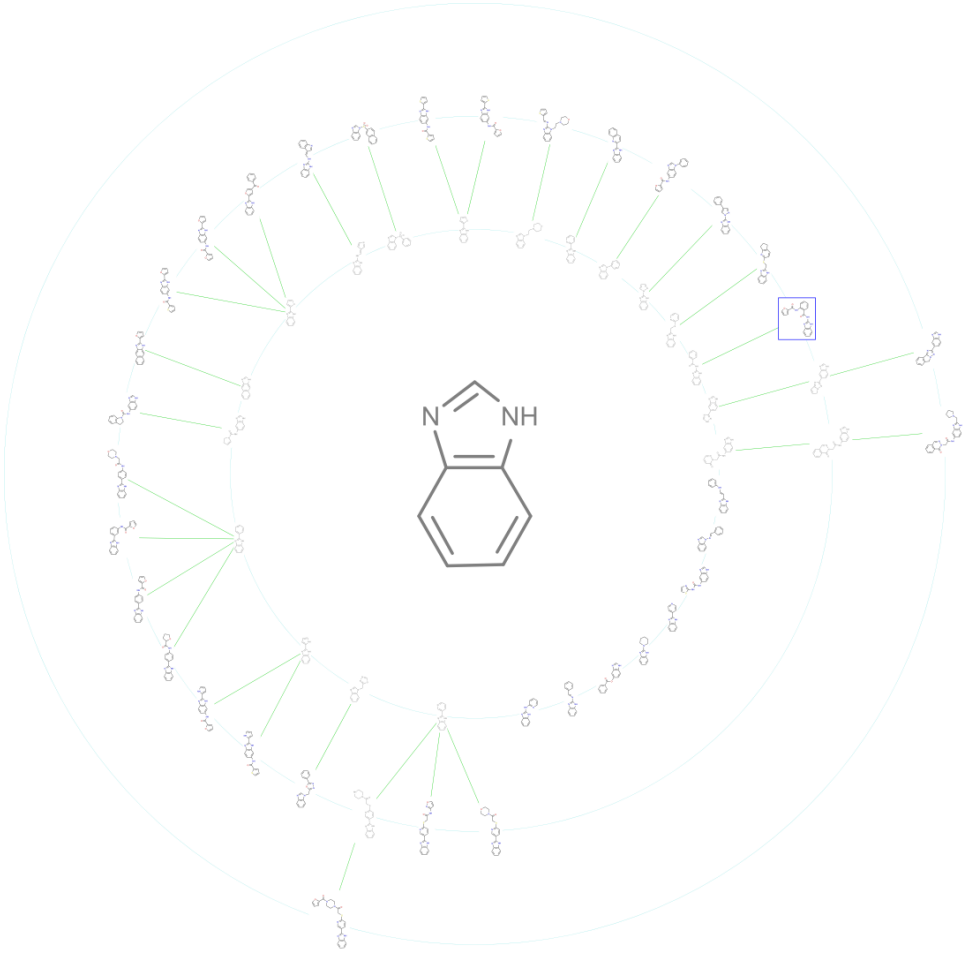

Abbildung 31: Ansicht eines Teilbaums mit ausgeblendeten Kanten auf der ersten Ebene und vergrösserter Wurzel

Aufgrund der oben beschriebenen Eigenschaften der Daten und den dadurch nötigen Modifizierungen am Layout wirkt der Baum bei einer weiten Zoomstufe wie ein Ring. Um die Struktur des Baumes zu verdeutlichen werden die Radien auch abhängig von der Zoomstufe angepasst. Bei einer weiten Zoomstufe werden die Radien vergrössert, um die Kantenverläufe und damit die hierarchische Struktur weiterhin sichtbar darzustellen. Bei einer nahen Zoomstufe werden die Radien verkleinert, um möglichst viele benachbarte Scaffolds auf dem Bildschirm darzustellen.

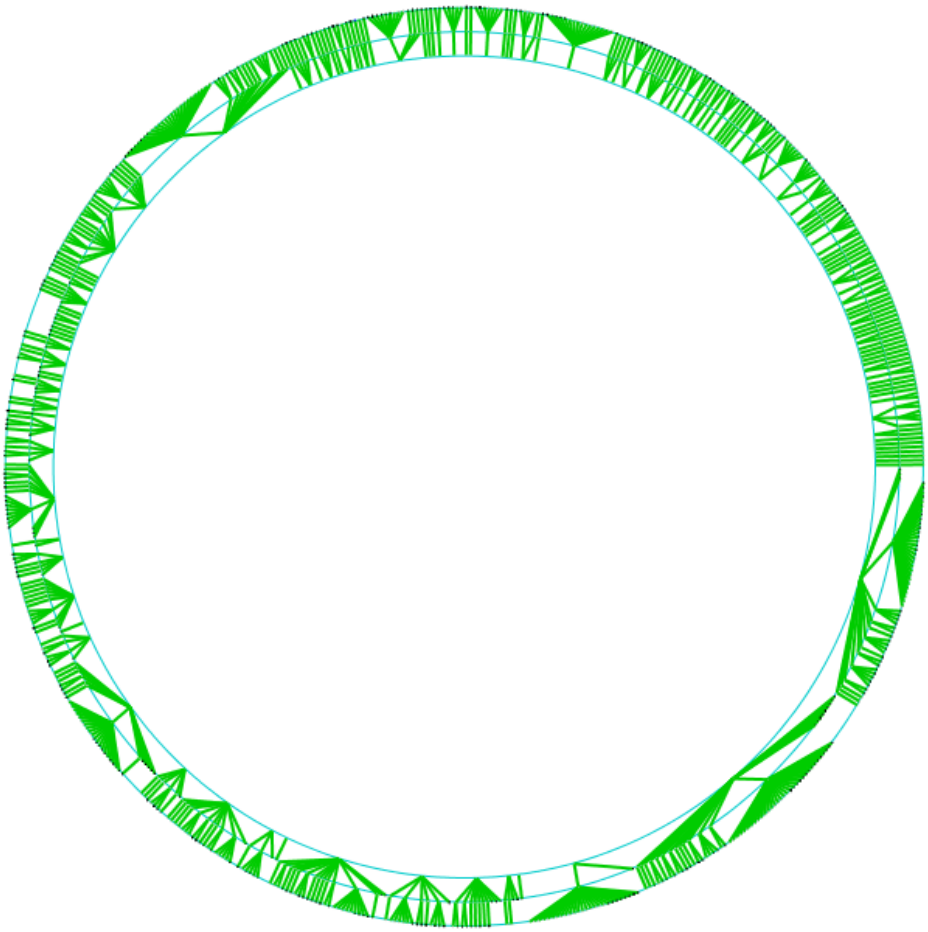

Abbildung 32: Ein automatisch erstelles Baumdiagramm des CBSE mit 135 Strukturen auf dem innersten Ring. Zudem wurden bei dieser Zoomstrufe die Radien vergrössert

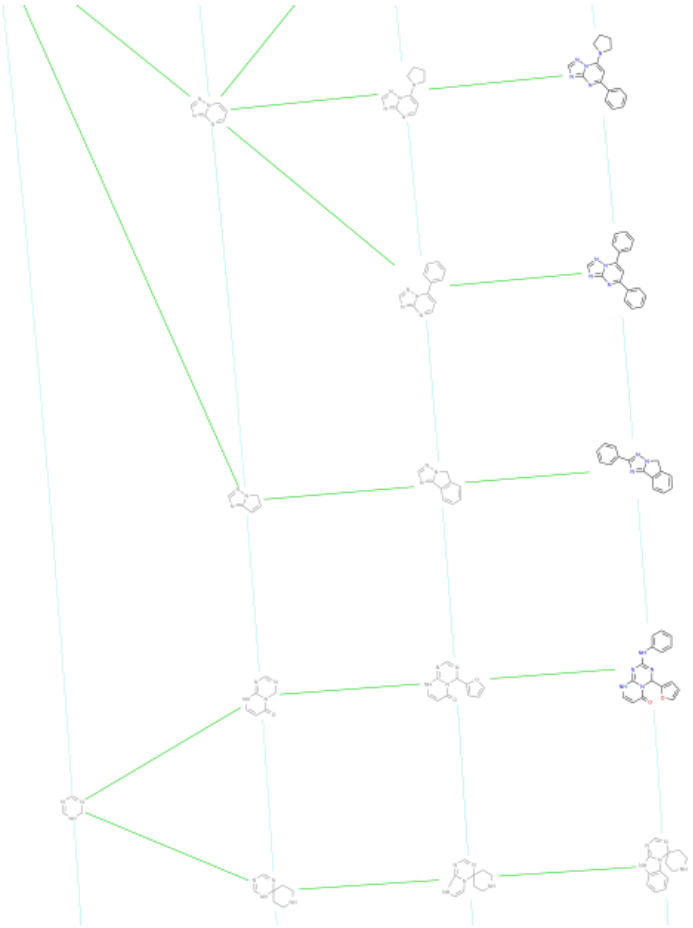

Abbildung 33: Ansicht eines Ausschnittes mit verkleinerten Radien

Eine besondere Herausforderung bei der Programmierung des Layouts war es, das Layout sowohl für eine geringe Anzahl an Strukturen wie auch für eine sehr grosse Menge an Strukturen gleichermassen anzupassen. Genauso wie die Anzahl der Strukturen die Qualität des Layout beeinflusst, spielt die Struktur der vom Benutzer ausgewählten Daten eine Rolle. Es können Bäume entstehen mit vielen Kindern und wenigen Hierachieebenen, aber auch Bäume mit wenig Kindern und vielen Hierarchieebenen.

Um den Benutzer für jeden Datensatz ein geeignetes Layout zu bieten, gibt es die Möglichkeit das Layout manuell anzupassen. Zum einen kann man die Radien zwischen den Hierarchieebenen skalieren, aber auch die Grösse der SVGs ist vom Benutzer modifizierbar. So kann der Benutzer eine optimale Darstellung des Datensatzes erreichen und in Verbindung mit weiteren graphischen Einstellungen, z.B. Strichstärken und Farben, eine druckoptimierte Ansicht einstellen.

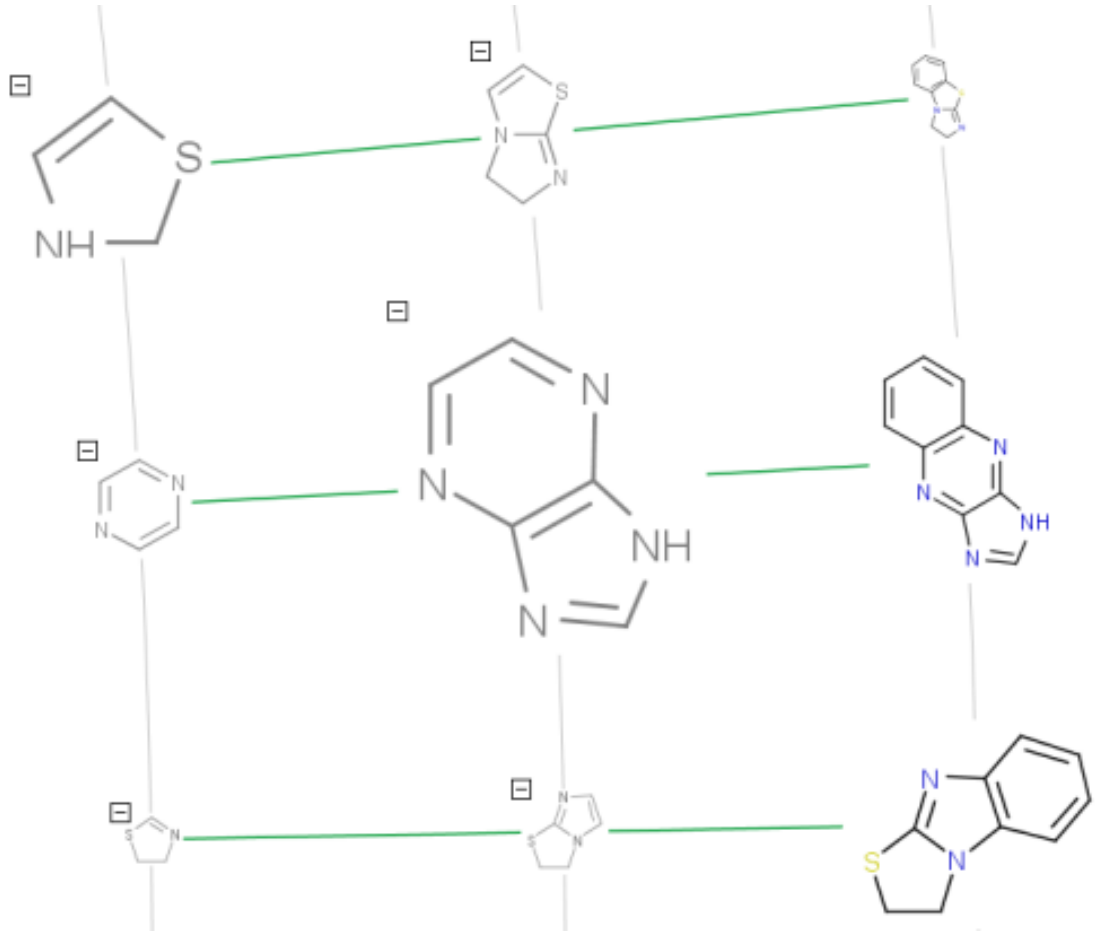

Abbildung 34: Ansicht eines Ausschnittes mit manuell angepassten SVG-Grössen

3.3.1.2 Linear Layout

Das Linear Layout ist im Grunde ein klassisches Baumlayout, bei dem die Wurzel ganz oben steht und die Kinder auf einer horizontalen Ebene gleichmäßig verteilt angeordnet sind. Dieses Layout wurde auf die Seite gedreht, so dass die Wurzeln der Teilbäume alle auf einer vertikalen Ebene ganz links und folgende Ebenen weiter nach rechts gezeichnet werden. (Siehe Abb. 35)

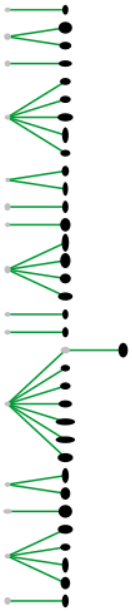

Abbildung 35: Linear Layout

Da bei diesem Layout die Knoten einer Ebene untereinander angeordnet sind, ist es dem Benutzer besonders gut möglich die Sortierung der Knoten zu überblicken. Zur Berechnung des Layouts wird der Datenbaum zwei mal rekursiv durchlaufen. Der erste Durchlauf dient der Berechnung der Anzahl der Blätter für jeden Teilbaum. Diese Information ist später nötig, um zu wissen, wie viel vertikalen Platz jeder Knoten benötigt. Für diese Berechnung wird der Baum Bottom-Up durchlaufen. Jedes Blatt bekommt den Wert 1, der Wert für jeden Knoten setzt sich aus der Summe aller Kinderwerte zusammen. (siehe Abb. 36)

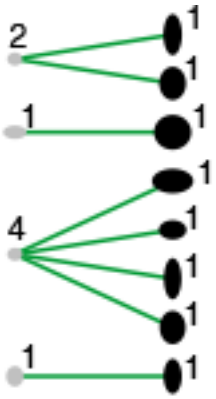

Abbildung 36: Anzahl der Blätter für jeden Teilbaum

Danach wird der Baum rekursiv Top-Down durchlaufen. Dabei wird für jeden Knoten die Position der Kinder anhand der Position des Knotens und der Anzahl der Blätter des Teilbaums berechnet, der unter dem

jeweiligen Kind hängt. Die Position der Wurzel wird mit (0,0) initialisiert. Um Knotenüberlappungen zu vermeiden, werden vor dem Layout alle Knoten im Datensatz durchlaufen, und es wird nach dem größten bzw. höchsten Knoten gesucht. Die Höhe dieses Knotens wird als vertikaler Abstand zwischen den Knoten benutzt. Da es keinen größeren Knoten gibt, ist sichergestellt, dass dieser Abstand ein überlappungsfreies Layout garantiert.

3.3.1.3 Balloon Layout

Beim Balloon Layout werden die Kinder rekursiv auf einem Kreis um ihren Elter angeordnet. So entsteht ein Graph der an Ballons oder an eine Blüte erinnert. (Siehe Abb. 37)

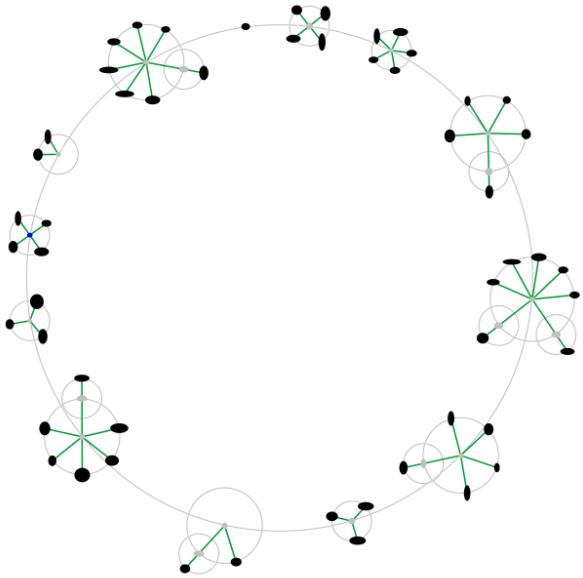

Abbildung 37: Balloon Layout des CBSE

Auch dieser Layoutalgorithmus arbeitet in zwei Schritten. Im ersten Schritt werden alle Knoten Bottom-Up durchlaufen, und es wird für jeden Knoten ein Radius berechnet. Ist der Knoten ein Blatt, also ohne Kinder, so wird sein Radius auf einen Minimumwert gesetzt. Hat der Knoten Kinder, so wird sein Radius wie folgt abgeschätzt: Bildet man eine Summe über alle Radien der Kinder, so bekommt man eine Annäherung an den gewünschten Umfang des Kreises, auf dem die Kinder liegen sollen. Aus diesem Umfang lässt sich der Radius berechnen. Da diese Abschätzung allerdings etwas zu klein ist, wird dieser Radius um den Faktor 1,2 skaliert. Da diese Berechnung bei Knoten mit wenig Kindern immer noch zu kleine Radien erzeugt, legen wir einen Mindestradius für alle inneren Knoten fest.

Nachdem für alle Knoten der Radius berechnet wurde, wird wieder rekursiv für alle Knoten die Position berechnet. Da die Radien der Knoten bereits bekannt sind, fehlt nur noch der Winkel zu den Polarkoordinaten, um die endgültige Position der Knoten zu bestimmen. Dafür wird für jeden Knoten der Winkel berechnet. Wie viel Platz jedes Kind auf dem Kreis bekommt, hängt von der Größe seines Teilbaums, also von seinem Radius ab. Auch beim Balloonlayout wird vor dem eigentlichen Layouten nach dem größten Knoten gesucht. Die Ausmaße dieses Knotens dienen dem Algorithmus als Minima für Radien und Winkel. So wird ein größenbedingtes Überlappen der Knoten vermieden.

3.3.2 Erforschen des Graphen

Der CBSE ist ein Tool mit dem sich große Datenräume erkunden und durchsuchen lassen. Man kann mit den zur Verfügung gestellten Features einen großen Datensatz in kürzester Zeit auf die wesentlichen und vom Benutzer gewünschten Daten reduzieren oder sich einen Überblick über den Datensatz verschaffen. Der Benutzer kann gezielt eine gewünschte Menge von Knoten aus dem Graphen extrahieren oder einfach ausblenden. Dabei helfen ihm die folgenden Funktionen:

Ein- und Ausklappen von Ästen

Nach dem Filtern der Daten bei Programmstart ist die Zahl der darzustellenden Konten immer noch recht hoch und kann vom Benutzer meist nicht auf einen Blick erfasst werden. Der CBSE löst dieses Problem, indem nur die erste Ebene der Daten angezeigt wird. Will der Benutzer die Kinder eines Knoten sehen, so kann er diese ausklappen. Umgekehrt ist es möglich einen ganzen Ast auszublenden. Mit diesem System kann sich der Benutzer sehr intuitiv durch die Daten bewegen und nur die Knoten anzeigen lassen, die ihn gerade interessieren. Folgende Features stehen durch dieses System zur Verfügung:

- **Kinder ein-/ausklappen** Über das Kontextmenü oder das Expandiericon lassen sich die Kinder eines Knotens aus- und einklappen. Bei Knoten, die keine Kinder haben wird das Expandicon nicht angezeigt.
- **Ebene ein-/ausklappen** Über das Kontextmenü kann man die Kinder sämtlicher Knoten auf der Ebene des ausgewählten Knotens aus- und einklappen.
- **gesamten Unterast ein-/ausklappen** Über das Kontextmenü kann man den gesamten Unterbaum ab dem ausgewählten Knoten aufslappen lassen.

Multitabbing

Eine Möglichkeit für den Benutzer die Anzahl der betrachteten Knoten oder den Graphen auf die Knoten zu reduzieren, die den Benutzer interessieren, ist einen neuen Tab zu öffnen. Es gibt dafür zwei Möglichkeiten: Die erste Möglichkeit ist, sich über das Kontextmenü den Unterbaum ab dem ausgewählten Knoten im neuen Tab anzeigen zu lassen. Dabei kann der Benutzer entscheiden, ob der Teilbaum im neuen Tab Knoten enthalten soll, die vom Initialfilter herausgefiltert wurden oder nicht. Nachdem der Benutzer alle gewünschten Knoten markiert hat, kann er sich diese dann über das Kontextmenü im neuen Tab anzeigen lassen.

3.3.3 Interaktion

Innerhalb des CBSE kann mittels Maus und Tastatur interagiert werden. Dabei wurde darauf geachtet, dass von anderen Anwendungen bekannte Benutzungsarten auch im CBSE Verwendung finden. Dadurch sollte ein schnelles Einfinden des Benutzers in den CBSE möglich gemacht werden. In den folgenden Unterkapiteln wird auf einige Interaktionsmöglichkeiten eingegangen:

3.3.3.1 Zoom

Das Hinein- und Hinauszoomen, also der Vergrösserung bzw. Verkleinerung der Arbeitsfläche, ist mit der Tastatur und - falls vorhanden - dem Scrollrad der Maus möglich. Darüber hinaus kann die Grösse der darzustellenden Fläche über die sog. Minimap, einer verkleinerten Darstellung des gesamten dargestellten Strukturraums, verändert werden. Ein rotes Rechteck zeigt innhalb dieser Verkleinerung den aktuell im Hauptfenster dargestellten Bereich an und kann mittels Klicken und Ziehen der Maus neu definiert werden. Geschieht dies ändert sich die Darstellung im Hauptfenster gemäß der neu definierten Begrenzungen. Dadurch kann nicht nur ein ganz anderer Bereich innerhalb der Darstellung gewählt werden, eine Vergrösserung oder Verkleinerung ist hiermit auch möglich.

3.3.3.2 Pan

Die Pan-Funktion erfolgt mit der Maus. Durch drücken der linken Maustaste auf einem freien Bereich im Graphen, kann man mit der Maus ziehen. Die Ansicht bewegt sich entsprechend der Bewegung der Maus mit.

Intern wird dazu ein MouseListener benutzt, welcher bei gedrückter linken Maustaste erkennt ob ein Scaffold, oder eine freie Fläche getroffen wurde und passt bei der Bewegung der Maus die Ansicht laufend an.

3.3.3.3 Cursor

Es ist auch möglich, den CBSE mit der Tastatur zu bedienen und so durch den Graphen zu navigieren. Hilfsmittel ist hierzu der Cursor. Der Cursor wird durch eine standardmässig blaue Umrandung einer Struktur dargestellt. Er dient der Orientierung und der Ausführung der vom Benutzer eingegebenen Funktionen. Die Farbe des Cursors ist vom Benutzer in den Optionen einstellbar.

Durch die Pfeiltasten ist es möglich, den Cursor innerhalb des Graphen zu bewegen. Ausgangspunkt hierbei ist die „12 Uhr“-Position. Durch die rechte Pfeiltaste wird der Cursor im Uhrzeigersinn auf der aktuellen Hierarchieebene bewegt. Zwischen den Hierarchieebenen kann durch die obere bzw. untere Pfeiltaste navigiert werden.

Alle Tastatureingaben gelten für die durch den Cursor markierte Struktur, d.h. wenn der Benutzer durch Drücken der Enter-Taste Kinder aufklappen möchte, werden die Kinder der durch den Cursor ausgewählten Struktur aufgeklappt.

Beim Programmstart ist das ausgewählte Home-Molekül durch den Cursor markiert. Falls vom Benutzer noch kein Home-Molekül ausgewählt worden ist, wird das Molekül mit der internen VNode-ID ausgewählt, das sich immer auf der „3 Uhr“-Position befindet.

Ausserdem kann der Benutzer den Cursor als „Ankerpunkt“ benutzen, da er jederzeit durch drücken der Taste ’C’ den Cursor fokussieren kann.

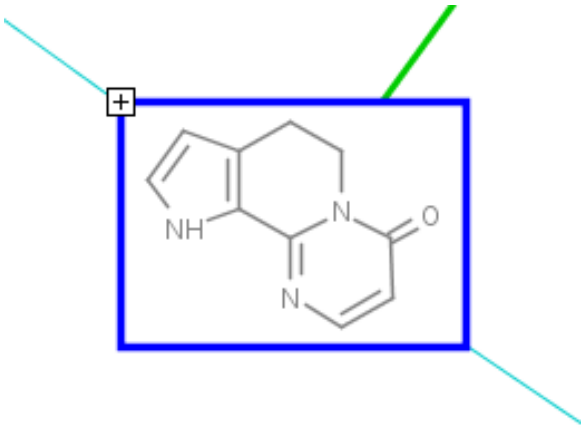

Abbildung 38: Der Cursor wird durch eine Umrandung der Struktur dargestellt

3.3.3.4 Selektion

Im CBSE gibt es eine Möglichkeit Scaffolds zu selektieren. Dabei werden die Knoten entsprechend eingefärbt (Standard rot) und es werden Deteils zu diesen Scaffolds im Detailfenster angezeigt.

Man kann die Scaffolds auf drei verschiedene Weisen selektieren:

- **Selektion durch Mausklick:**
  Hierbei wird jedes nicht selektiertes Scaffold, das angeklickt wird, selektiert und jedes selektiertes Scaffold deselektiert.
- **Selektion durch Mausziehen:**
  Hierbei muss ein Rahmen mit der rechten Maustaste gezogen werden. Alle Scaffolds, die innerhalb des Rahmens liegen werden zu selektierten Scaffolds hinzugenommen. Um mehrere Scaffolds zu deselektiere, muss der Rahmen mit gedrückter Shift-Taste gezogen werden. Alle Scaffolds, die innerhalb des Rahmens liegen werden aus der Selektion herausgenommen.
- **Selektion durch Cursor:**
  In diesem Fall betrifft die Selektion das Scaffold auf dem der Cursor liegt. Durch drücken der Leertaste wird dieses Scaffold zur Selektion hinzugenommen bzw. aus der Selektion entfernt.

Intern wird im Bereich der VIS-Gruppe eine Liste mit den selektierten Scaffolds verwaltet. Zu dieser Liste können Scaffolds hinzugefügt oder entfernt werden. Die Ansicht wird entsprechend angepasst. Zusätzlich dazu wird bei jeder Änderung der Liste die aktuell-selektierte Scaffolds an die GUI-Gruppe mitgeteilt. So wird gleichzeitig das Detailfenster aktualisiert.

3.3.3.5 Einfärbefunktion

Die Einfärbefunktion beinhaltet 3 Teilfunktionen, es sind 3 verschiedene Arten des Färbefilters:

- Knoten entsprechend eines Filters färben
- Knoten innerhalb definierter Grenzen färben
- Knoten für eine Eigenschaft färben

Knoten entsprechend eines Filters färben:
Bei dieser Einfärbefunktion wird ein Standardfärbefilter verwendet. Es können gleiche Einstellungen eingestellt werden wie beim Startdialog. Zusätzlich muss eine Farbe eingegeben werden, mit welcher die zum Filter passenden Knoten eingefärbt werden.

Knoten innerhalb definierter Grenzen färben:
Bei dieser Einfärbefunktion wird ein erweiterter Färbefilter verwendet. Bei diesem Filter können zu jeder Scaffoldeigenschaft zwei Grenzen eingegeben werden. Alle Scaffolds die innerhalb dieser definierter Grenzen liegen werden eingefärbt.

Knoten nach einer Eigenschaft färben:
Bei dieser Einfärbefunktion wird ein erweiterter Färbefilter verwendet. In diesem Fall kann nur eine Eigenschaft gewählt werden, nach welcher man färben will. Es können für diese Eigenschaft, so wie im vorherigen Fall, zwei Grenzen eingegeben werden. Alle Scaffolds die innerhalb dieser definierter Grenzen liegen werden eingefärbt. Die Intensität der Färbung hängt vom Wert der Eigenschaft ab, je näher zu Maximum, desto intensiver die Farbe (bzw. je nach Einstellung umgekehrt).

Um diese Funktion zu Realisieren mussten alle Gruppen mitarbeiten. Die GUI-Gruppe bot die unterschiedliche Färbefilter an und modellierte die Dialoge. Die VIS-Gruppe übernahm die Färbung der Knoten. Und die DB-Gruppe suchte die zum Filter passende Scaffolds in der Datenbank. Der interne Ablauf ist also so, dass zuerst die Filtereinstellungen an die VIS gehen, diese werden an DB weitergegeben. Die gefundene Scaffolds werden zurück an die VIS übergeben und anschliessend werden die entsprechenden Knoten eingefärbt.

3.3.4 Modifikationen der SVGs

Darstellung von SVGs

Was ist SVG?

SVG steht für Scalable Vector Graphics (deutsch: Skalierbare Vektorgrafiken) und ist ein Grafikformat. Im Gegensatz zu Rastergrafik (*.bmp, *.jpg, u.a.) werden aber nicht nur Informationen für jeden Pixel gespeichert, sondern es besteht die Möglichkeit ein Bild aus grafischen Primitiven aufzubauen (Linien, Kreisen, Polygonen,...). Die Beschreibung eines SVGs erfolgt durch XML-Syntax, so dass die Grafik intern wie ein Baum aufgebaut ist, mit Elementen und den Elementen zugewiesenen Attributen (Abb. 39). Dadurch ist es möglich ein SVG mit einem Texteditor zu bearbeiten und sogar speicherplatzeffizient als String abzuspeichern (siehe Datenstruktur).

```
<svg xmlns="http://www.w3.org/2000/svg"
 width="226" height="226">
  <circle cx="110" cy="107" r="80" stroke="black"
     stroke-width="5" fill="red" />
</svg>
```

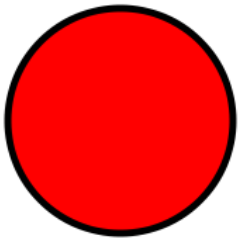

Abbildung 39: SVG Quelltext und grafische Ausgabe

SVGs im CBSE

Der besondere Vorteil von SVGs gegenüber einem Rastergrafikformat ist -wie der Name schon sagt- die Skalierbarkeit, d.h. man kann ein SVG beliebig skalieren ohne erhöhten Speicherbedarf und vorallem ohne Qualitätsverlust.

Die Skalierbarkeit ist der Hauptgrund für die Benutzung des SVG-Formates bei die Darstellung der Scaffolds und Moleküle. Da bei der Navigation im Strukturraum oft zwischen verschiedenen Zoomstufen gewechselt wird, ist es wichtig, dass die Strukturen bei einer hohen Zoomstufe nicht pixelig dargestellt werden und bei einer niedrigen Zoomstufe die Strukturen voneinander unterschieden werden können (Abb. 40).

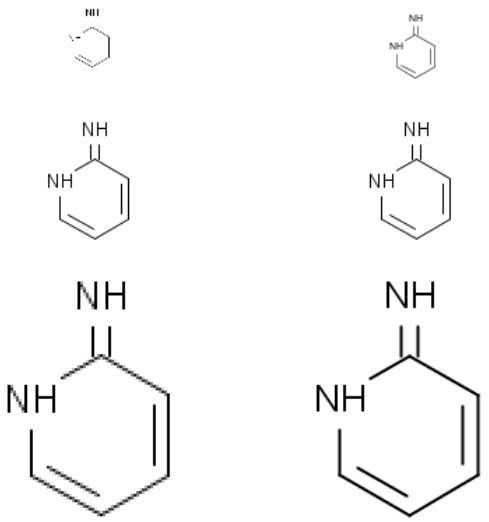

Abbildung 40: Ein Scaffold in den Zoomstufen 50%, 100%, 200%. Links Rastergrafik, Rechts SVG

Manipulation der SVGs

Durch das SVG-Toolkit Batik ist es möglich, die Baumstruktur des SVGs nach der Erzeugung des Objektes der Klasse `SVGDocument` zu durchlaufen und ausgewählte Attribute der Element zu verändern. Die Veränderung erfolgt damit on-the-fly und hat keinen Einfluss auf die in der Datenbank gespeicherten Beschreibung des SVGs. Es werden die folgenden Eigenschaften bei allen SVGs verändert:

- stroke-linecap = „round“ (runder Linienabschluss)
- stroke-width = „2“ (standardmässig doppelte Linienbreite)

Diese Änderungen dienen der Verbesserung der Darstellung der SVGs. Der runde Linienabschluss lässt die Struktur bei einer grossen Zoomstufe „geschlossener“ wirken (Abb. 41).

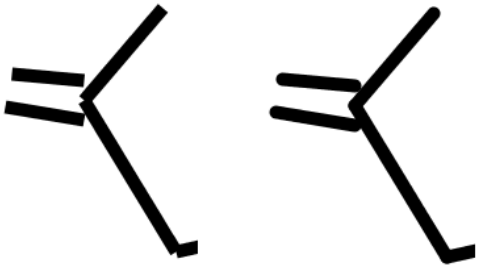

Abbildung 41: Die verbesserte Darstellung durch den runden Linienabschluss

Durch die standardmässig eingestellte doppelte Linienbreite sind die Strukturen bei einer geringen Zoomstufe besser zu erkennen (Abb. 42). Der Benutzer kann die Linienbreite in den Optionen anpassen.

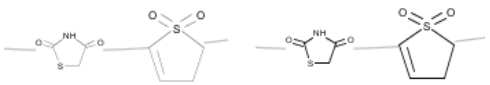

Abbildung 42: Die verbesserte Darstellung durch die doppelte Linienbreite (Zoomstufe: 30%)

Beim Durchlauf der Baumstruktur des SVGs besteht die Möglichkeit die Farbe der Linien und die Farbe der Buchstaben der chemischen Elemente zu verändern. Dies benutzt der CBSE um selektierte und virtuelle Scaffolds hervorzuheben.

Virtuelle Scaffolds werden grau eingefärbt und sind dadurch besser von anderen Scaffolds zu unterscheiden (Abb. 43). Virtuelle Scaffolds sind Scaffolds, bei denen keine Moleküle hinterlegt sind und die nur in die Baumstruktur eingefügt wurden, um die schrittweise Erweiterung der chemischen Struktur zu verdeutlichen.

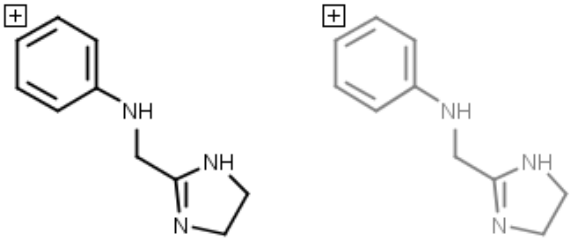

Abbildung 43: Graue Einfärbung von virtuellen Scaffolds

Selektierte Scaffolds werden standardmässig rot eingefärbt, aber der Benutzer kann eine beliebige Farbe in den Optionen auswählen (Abb. 44).

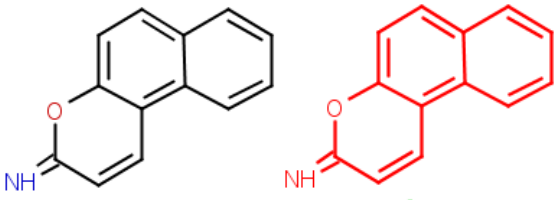

Abbildung 44: Selektierte Scaffolds werden standardmässig rot eingefärbt

Der Weg eines SVGs

Wie bereits erwähnt liegen SVGs in XML-Syntax vor, so dass sie als `String` gespeichert werden können. Mit dem String wird dann durch die Batik-Klasse `SAXSVGDocumentFactory` ein Objekt der Klasse `SVGDocument` erzeugt. Durch das SVGDocument kann auf die Wurzel der XML-Baumstruktur zugegriffen werden und es können bei einem Durchlauf der Baumstruktur Attribute einzelner Elemente des SVGS, z.B. das Attribut Liniendicke aller Linienelementen, modifiziert werden.

Das SVGDocument kann dann duch die Batik-Klasse `GVTBuilder` ein Objekt der Klasse `GraphicsNode` erzeugen. Dieses Objekt beschreibt das SVG in Objekten der AWT-Klasse `Graphics`

(`java.awt.Graphics`). Die `GraphicsNode`-Klasse hat eine Methode
`paint(Graphics2D g2d)` mit der die paint-Methode der VNode-Klasse bei geeigneter Zoomstufe überschrieben werden kann und das SVG als Knoten gezeichnet wird.

Exportfunktion

Neben der Navigation innerhalb des Graphen ist der Export eine wichtige Funktin des CBSE. Bisher mussten diese Graphen sehr aufwendig von Hand gezeichnet werden. Dabei ist für einige hundert Scaffolds schon eine grossformatige Fläche (DIN A3 und grösser) nötig. Der CBSE bietet die Möglichkeit die aktuelle Graphenansicht und den kompletten Graphen als SVG und als PNG-Datei zu exportieren.

Auch hier ist die Skalierbarkeit des SVG-Formates ein grosser Vorteil. Wenn der Graph als SVG exportiert wird, kann er auf ein beliebiges Papierformat ohne Qualitätsverlust skaliert werden. Ausserdem können die exportierten SVGs zum Beispiel mit dem Open-Source Programm Inkscape[14] weiterverarbeitet werden. Eine PNG-Datei kann direkt mithilfe der ImageTranscoder-Funktion von Batik durch den CBSE erzeugt werden.

SVG-Cache

Problematik

Als problematisch bei der Speicherverwaltung erwiesen sich SVGs im Zusammenspiel mit dem Toolkit Batik. Um SVGs in ihrer XML Repräsentation in Java grafisch ausgeben zu können, erstellt Batik zunächst eine Repräsentation des XML Dokuments im Speicher, so dass auf die einzelnen Elementen über die *Document Object Model* (DOM) Schnittstelle zugegriffen werden kann. Hierbei handelt es sich um eine Baumstruktur, die die Elemente des Dokuments hierarchisch organisiert. Diese Struktur kann jederzeit im Speicher über die DOM Schnittstelle verändert werden. Alle Grafiken werden in Java über die Grafikkontext Klasse `Graphics` gezeichnet. Batik enthält das *Graphic Vector Toolkit* (GVT), mit dem es diese Ausgabe erzeugt. Dazu wird ein Baum aus Objekten der Klasse `GraphicsNode` erzeugt, der mit der Struktur des DOM Baums korrespondiert. Zwischen dem GVT Baum und dem DOM Baum erhält ein „Bridging-Modul“ die Konsistenz, so dass sich Änderungen in der DOM Struktur auch auf den GVT Baum auswirken. Durch dieses Konzept entsteht ein erheblicher Overhead und die Datenstruktur nimmt viel Speicher ein. Bei einem durchschnittlich komplexen SVG wie es für den CBSE verwendet wird, benötigt die Datenstruktur von Batik etwas zehn mal so viel Speicherplatz wie die einfache String Repräsentation.

Bei grossen Graphen werden jedoch nie alle Knoten gleichzeitig als SVG dargestellt, da diese entweder ausserhalb des gerade sichtbaren Bereichs liegen oder durch den semantischen Zoom ausgeblendet werden. Angesichts dieser Tatsache ist es unnötig, alle SVGs stets als aufbereitete Struktur im Speicher zu halten. Dieses Problem wurde dadurch gelöst, dass in den `DNode` Objekten nur noch die vergleichsweise wenig speicherintensive Textrepräsentation der SVGs gespeichert wird und nur eine begrenzte Anzahl SVGs von Batik eingelesen und im Speicher gehalten wird. Da das Aufbereiten der SVGs einige Zeit beansprucht, die flüssige Interaktion des Nutzers mit dem Graphen aber nicht beeinträchtigt werden soll, geschieht dies im Hintergrund in einem eigenen Thread. Währenddessen wird ein einfaches graues Rechteck gezeichnet, das ersetzt wird, sobald das richtige SVG geladen wurde.

Technische Umsetzung

Die Umsetzung in einem eigenen Thread, der nur für das Laden eines SVGs zuständig ist, erfordert Kommunikation mit dem Toolkit Piccolo. Piccolo wird - wie Java Swing auch - vollständig im *Event Dispatch Thread* ausgeführt. Die Klasse `EventQueue` speichert Events und arbeitet sie nacheinander ab, indem der Code der zugehörigen `Listener` im *Event Dispatch Thread* ausgeführt wird. Piccolo ist

[14] http://www.inkscape.org/

nicht *thread-sicher*[15] und alle Änderungen am Szenengraph und seinen Elementen sollten ausschließlich aus dem *Event Dispatch Thread* vorgenommen werden. Hierdurch soll die Programmierung, der Quellcode und die Fehlersuche vereinfacht werden. Es bleibt jedoch die Möglichkeit, über die Methode `invokeLater(Runnable r)` der `EventQueue` eine Klasse mit der Methode `run()` zu übergeben. Die `run()`-Methode kann beliebigen Code enthalten, der ausgeführt wird, nachdem alle anstehenden Events vom Dispatch Thread abgearbeitet worden sind. Auf diese Weise kann Piccolo darüber informiert werden, dass ein zuvor angefordertes SVG geladen wurde und der Platzhalter durch das richtige SVG ausgetauscht werden kann.

Zuvor muss jedoch ein SVG angefordert werden. Dies geschieht während des Renderings einer Szene aus der `paint()`-Methode eines `VNode`. Piccolo ruft diese Methode nur für solche Knoten auf, die zur Darstellung des aktuellen Frames benötigt werden. Wird also die `paint()`-Methode für einen Knoten aufgerufen und das zugehörige SVG befindet sich nicht im Cache, wird dieses angefordert und zunächst statt des SVGs ein graues Rechteck gezeichnet. Das Anfordern des SVG aus dem *Event Dispatch Thread* beschränkt sich darauf, eine Referenz auf das `VNode`-Objekt, dessen SVG geladen werden soll, in einer Queue zu speichern. Diese Aktion kann schnell erledigt werden, ohne dass das Rendering lange blockiert wird.

Als Queue wird die Java Datenstruktur `LinkedBlockingQueue` verwendet. Die Datenstruktur arbeitet nach dem FIFO (first-in-first-out) Prinzip und ist in ihrer Grösse nicht beschränkt. Eine Besonderheit dieses Datentyps ist es, dass über die Methode `take()` stets auf das vorderste Element der Queue zugegriffen werden kann. Ist die Queue leer, wird nicht `null` zurückgegeben wie es bei einer gewöhnlichen Queue der Fall wäre, sondern der Ablauf des Threads wird blockiert bis ein Element eingefügt wurde, das zurückgegeben werden kann. Es wird also gewartet bis weitere SVGs angefordert werden. Nach der Java API eigenen sich die Datenstruktur besonders für Erzeuger-Verbraucher-Modelle. In diesem Sinne stellt Piccolo den Erzeuger dar: Navigiert der Benutzer im Scaffoldbaum zu einem Scaffold, dessen SVG noch nicht geladen worden ist, wird beim Zeichnen der neuen Ansicht festgestellt, dass das SVG geladen werden muss und als „Auftrag“ in der `BlockingQueue` abgelegt. Ein neuer Auftrag wird also im *Event Dispatch Thread* erzeugt. Der „SVG Loader Thread“ hingegen nimmt die Rolle des Verbrauchers ein, indem er den Auftrag aus der Queue entfernt und bearbeitet. Da also zwei unterschiedliche Threads auf der gleichen Datenstruktur arbeiten, muss ausgeschlossen werden, dass inkonsistente Zustände enstehen können. Die Implementierung der Datenstruktur ist bereits *thread-sicher*, so dass dadurch keine Probleme verursacht werden.

[15] Piccolo Developer's FAQ, `http://www.cs.umd.edu/hcil/jazz/learn/dev-faq.shtml`

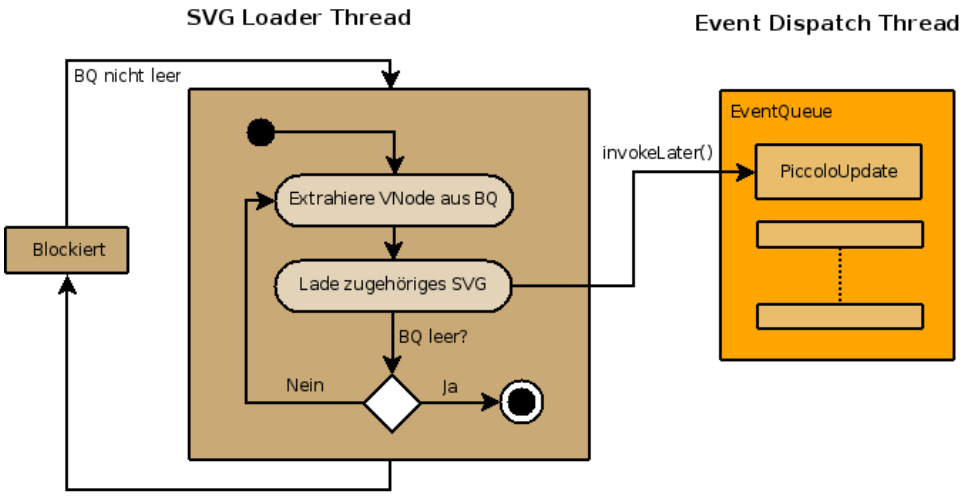

Abbildung 45: Zustände des SVG Loader Threads und Zusammenspiel mit Piccolo (BQ steht für die Datenstruktur `LinkedBlockinQueue`)

Der Thread arbeitet also nur, wenn wirklich SVGs geladen werden müssen und ist ansonsten inaktiv. Abb. 45 verdeutlich die Arbeitsweise des Systems. Nachdem ein SVG geladen wurde wird die Grafik wie beschrieben nicht unmittelbar aktualisiert, sondern über die Klasse `PiccoloUpdate` aus dem *Event Dispatch Thread*.

Die maximale Anzahl von SVGs, die im Cache gespeichert werden sollen, kann über einen Parameter festgelegt werden. Wird dieser Wert überschritten, werden alle SVGs gelöscht, die in der aktuellen Ansicht nicht dargestellt werden.

Unabhängig von der Anzahl der Tabs gibt es nur einen Cache, in dem alle SVGs verwaltet werden. Batik bietet über die DOM Schnittstelle komfortabele Möglichkeiten zum Anpassen eines SVGs. Diese Funktion wird unter anderem verwendet, um SVGs entsprechend der Eigenschaften des `VNode`-Objekts zu verändert. So kann beispielsweise ein SVG in einem Tab markiert sein (geänderte Farbe der Linien) und im anderen nicht. Deshalb ist es notwendig, wenn in zwei Tabs die gleichen Scaffolds dargestellt werden, zwei evtl. unterschiedlich angepasste SVGs zu verwenden. Es kann also für jeden `VNode` ein eigenes angepasstes SVG geladen werden. Diese SVGs werden beide aus dem gleichen XML String generiert, die daraus resultierende DOM Struktur aber evtl. unterschiedlich geändert. Da die maximale Anzahl der SVGs für alle Tabs gilt, entsteht dadurch kein zusätzlicher Speicherverbrauch.

3.3.5 Funktionsweise von Piccolo

Da der Aufbau des Pakets `vis` eng an die Struktur und Funktionsweise von Piccolo angelehnt ist, soll in diesem Abschnitt die Arbeitsweise des Toolkits genauer beschrieben werden. Piccolo verwendet ein flexibles Szenengraph-Modell, wie es häufig von 3D-Systemen benutzt wird und für 2D-Oberflächen sonst untypisch ist. Piccolo greift auf Grafikoperationen von Java 2D zurück. So kann Piccolo als Zwischenschicht betrachtet werden, die auf der Java API für primitive Grafikoperationen aufsetzt und selbst komplexere Operationen zur Verfügung stellt.

Java `Graphics2D` API

Java stellt gewisse primitive Grafikoperationen über die Klasse `Graphics2D` zur Verfügung. Objekte dieser Klassen können unter anderem zur Ausgabe auf dem Bildschirm, auf einem Drucker oder in ein Bild genutzt werden. Somit lässt sich auch die von Piccolo erzeugte Grafik auf diesen Medien ausgeben.

Über die Klasse `Graphics2D` können beispielsweise geometrische Formen (Klasse `Shape`), Texte und Bilder direkt gezeichnet werden. Dabei wird stets ein interner Zustand verwaltet, der sich auf jede solche Zeichenoperationen auswirkt. Dieser umfasst unter anderem die Farbe der Komponente, Schriftart, *Clipping* und eine *affine Transformation*.

Eine besondere Rolle für den Szenengraph von Piccolo spielen *affinen Transformationen* (repräsentiert durch die Klasse `AffineTransform`). Diese ermöglichen es, Koordinaten von einem Koordinatensystem in ein anderes zu übertragen. Geometrisch entsprechen die möglichen Abbildungen der Verschiebung (Translation), Drehung (Rotation), Skalierung und Scherung des Koordinatensystems bzw. des dargestellten Objekts. Eine *affine Transformation* im 2-dimensionalen Raum lässt sich durch eine 3×3 Matrix A darstellen. Die Bestimmung der neuen Koordinaten eines Punktes $p = (x, y)$ kann durch Multiplikation von A mit den *homogenen Koordinaten* $p_h = (x, y, 1)^T$ von p gewonnen werden:

$$A \cdot p_h = \begin{pmatrix} x' \\ y' \\ h \end{pmatrix}$$

Das Ergebnis entspricht dem Punkt $p' = (x'/h, y'/h)$. Mehrere Transformationen lassen sich durch Matrixmultiplikation zu einer zusammenfassen.

Szenengraph Modell

Piccolo verwendet einen hierarchischen Szenengraph, dessen Elemente alle von der Klasse `PNode` erben, die wesentliche Funktionalität für die grafische Darstellung, die Event-Verarbeitung und die Eltern-Kind-Verknüpfungen im Graph bereitstellt. Jedes Objekt dieser Klasse verfügt über eine eigene *affine Transformation* und definiert somit ein eigenes Koordinatensystem.

`PNode` ist die Basisklasse für alle grafischen Objekte und ihre Unterklassen können anwendungsspezifisch angepasst werden. Einige häufig benötigte Unterklassen sind in Piccolo enthalten wie `PImage`, `PText` und `PPath` für Bilder, Text und Pfade aus Linien und Kurven. Ein Objekt der Klasse `PNode` kann Kinder vom Typ `PNode` haben. Dadurch lassen sich komplexere Strukturen hierarchisch durch viele einfache zusammensetzen.

Weiterhin gibt es einige spezielle Unterklassen von `PNode` mit eigenen Aufgaben: `PRoot` repräsentiert die Wurzel des Szenengraphen und verarbeite in einer Schleife eingehende Ereignisse. In jedem Schleifendurchlauf werden vier Aktionen durchgeführt[16]:

[16] Piccolo Patterns, `http://www.cs.umd.edu/hcil/piccolo/learn/patterns.shtml`

1. Bearbeiten von Eingaben: Neu eingegangene Events von Swing werden ausgewertet und an den zugehörigen `EventListener` delegiert
2. Ausführen von Aktivitätsschritten
3. Validieren von Bounding-Boxen, falls sie als ungültig markiert worden sind
4. Zeichenfläche aktualisieren: Regionen, die zuvor als ungültig markiert worden sind, werden neu gezeichnet

`PLayer` sind Elemente im Szenengraph, die von einer Kamera (Klasse `PCamera`) betrachtet werden können. Ihre Kinder sind die eigentlichen grafischen Objekte. Die Ordnung der Layer bestimmt die Zeichenreihenfolge, so dass ein Layer durch einen anderen teilweise „überdeckt“ werden kann. Ein `PLayer` kann von ein oder mehreren `PCamera` Objekten betrachtet werden.

`PCamera` repräsentiert eine Ansicht auf ein oder mehrere Layer und ist Kind der Wurzel des Szenengraphen. `PCamera` erweitert die Klasse `PNode` um eine weitere *Bounding-Box* und eine *affine Transformotaion*, die vor dem Zeichen aller weiteren Objekte auf den assoziierten Layern angewendet wird. Ihre Skalierungseigenschaft bezieht sich auf alle Objekte, die später gezeichnet werden. Auf diese Weise ist die Zoomfunktion implementiert. *Panning* lässt sich durch die Translationseigenschaft der Transformation realisieren. Die *Bounding-Box* wird dabei stets entsprechend der Transformation angepasst. Eine Kamera ist normalerweise mit einem `PCanvas` verbunden, wobei es sich um einen Swing Komponente handelt, die die Ansicht der Kamera darstellt. Kameras können jedoch ausserdem auch als „interne Kameras“ im Szenengraph unterhalb von anderen Kameras auftauchen.

Die Ansicht einer Kamera wird gezeichnet, indem nacheinander alle Layer zusammen mit ihren Objekten gezeichnet werden. Dabei wird der Szenengraph ähnlich wie bei einer Tiefensuche durchlaufen und an jedem Knoten wird zuerst die zugehörige *affine Transformation* angewendet, anschliessend der Knoten selbst und danach seine Kinder gezeichnet. Dann erst wird die zum Knoten gehörige Transformation rückgängig gemacht. Dadurch beeinflusst die Transformation eines Knotens auch die Transformation all seiner Kinder, deren Transformation wiederum mit der gerade aktiven verknüpft wird. Piccolo überprüft dabei stets, ob sich ein Knoten (bzw. einer seiner Nachfolger) im momentan sichtbaren Ausschnitt der Kamera befindet. Ist dies nicht der Fall, wird der an diesem Knoten beginnende Unterbaum nicht verfolgt. Somit werden für jeden Frame nur die benötigten Objekte gezeichnet.

`PNode` bietet ausserdem die Möglichkeit, Event-Listener zu registrieren, um Benutzerinteraktion mit den Objekten zu ermöglichen. Piccolo erkennt dabei beispielsweise, über welchem Knoten sich gerade die Maus befindet oder auf welches Objekt geklickt wurde.

Bei Piccolo handelt es sich um ein *monolithisches* Toolkit. Damit ist gemeint, dass es eine Klasse gibt, die die wesentliche Funktionalität für alle grafischen Objekte zur Verfügung stellt, und diese an andere Klassen vererbt. Bei Piccolo heißt diese Klasse `PNode`. Dies unterscheidet den Szenengraph von Piccolo von vielen Modellen, die von 3D-Systemen bekannt sind, bei denen Funktionalität (wie Transformationen oder Transparenz) durch entsprechende eigentständige Elemente im Szenengraph realisiert wird. Solche Ansätze können als *polylithisch* bezeichnet werden. *Monolithische* Ansätze haben den Vorteil, dass sie meist einfacher zu erlernen sind, die Struktur des Szenengraph leichter zu verstehen ist und der Quellcode übersichtlicher ist[17].

[17] Toolkit Design for Interactive Structured Graphics, Bederson, B. B., Grosjean, J., & Meyer, J. (2004). IEEE Transactions on Software Engineering, 30 (8), pp. 535-546.

3.3.6 Datenstruktur und Speicherverwaltung

Eine geeignete Datenstruktur muss einerseits den Vorgaben aus den Cheminformatik gerecht werden und andererseits sollten sich bekannte Layout Algorithmen aus dem Gebiet des Graphenzeichnens einfach implementieren lassen. Hinzukommt, dass der CBSE Tabs unterstützt und somit mehrere Ansichten auf einen Teilgraphen parallel existieren können, was mit einem möglichst geringen zusätzlichen Speicherverbrauch realisiert werden soll. Die Datenstruktur soll ausserdem ermöglichen, dass Daten dynamisch aus der Datenbank nachgeladen werden können, wenn sie benötigt werden, wobei eine klar definierte Schnittstelle zum Paket `db` erhalten bleiben soll.

Die Elemente des chemischen Strukturraums lassen sich hierarchisch ordnen, sodass Bäume entstehen, deren Elter-Kind-Beziehungen auf der Struktur der chemischen Substanzen beruhen[18]. Dadurch ergibt sich eine Vielzahl von Bäumen, allerdings kein zusammenhängender Graph wie er für viele Algorithmen wünschenswert ist. Ein zusammenhängender Baum lässt sich aber leicht durch Einfügen einer neuen Wurzel erzeugen, die als Kinder die einfachsten Strukturen hat, die ihrerseits Wurzeln kompletter Bäume sind. Diese zusätzliche Wurzel soll nicht dargestellt werden, sondern nur die Arbeit mit der Datenstruktur vereinfachen.

Um der Tatsache gerecht zu werden, dass in mehreren Tabs die gleichen Knoten angezeigt werden können, werden nicht alle Daten eines Knotens in einem einzigen Objekt gehalten, sondern es wurde eine Aufteilung in die zwei Klassen `DNode` und `VNode` vorgenommen wie Abb. 46 zeigt. Ein Objekt vom Typ `DNode` enthält die Daten aus der Datenbank, die für die Darstellung des Knotens benötigt werden. Dazu gehören u.a. das SVG in seiner String Repräsentation (siehe unter 3.3.4, S. 81) und die Information, ob der Knoten Kinder hat. Diese Objekte beinhalten allerdings keine Informationen zu einer konkreten Darstellung wie Koordinaten für das Layout. Da `DNode` und `DTree` solche Informationen beinhalten, die sich direkt aus der Datenbank gewinnen lassen, fällt die Implementierung in den Arbeitsbereich der Datenbank-Gruppe. Beide Klassen wurden deshalb als Interface ausgelegt und stellen somit die Schnittstelle des Paketes `vis` zu den zu visualisierenden Daten dar. Diese müssen nicht zwingend aus einer Datenbank gelesen werden und es besteht die Möglichkeit, dass Daten dynamisch in den Speicher nachgeladen werden.

[18] Charting biologically relevant chemical space: A structural classification of natural products (SCONP); Koch, M. A., Schuffenhauer, A., Scheck, M., Wetzel, S., Casaulta, M., Odermatt, A., Ertl, P., Waldmann, H.; Proc. of the National Academy of Sciences of the United States of America; 2005

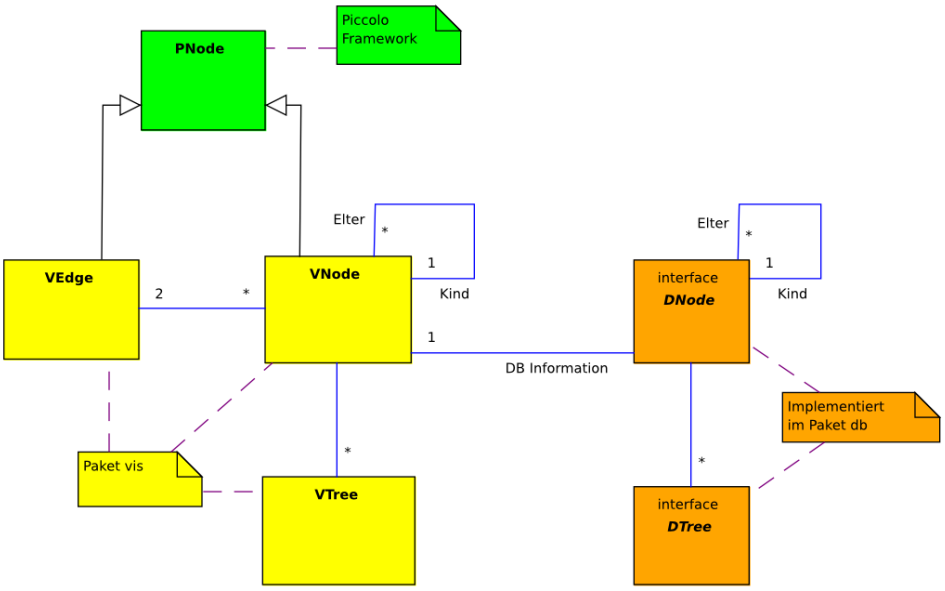

Abbildung 46: Vereinfachtes Klassendiagramm des Pakets `vis`

Die Darstellung selbst übernehmen Objekte der Klasse `VNode`, die als eine grafische Repräsentation von `DNode` Objekten gesehen werden können und in den Szenengraph des Toolkits Piccolo integriert werden können. Dies wird dadurch ermöglicht, dass von `PNode` (Piccolo Szenengraph Element) geerbt wird wie Abb. 46 verdeutlicht. Eine kurze Erläuterung zum Modell des Szenengraphen wie er von Piccolo verwendet wird ist im Abschnitt 3.3.5 auf S. 85 zu finden.

Da jede Instanz von `VNode` eine Referenz auf sein zugehöriges `DNode` Objekt hält, ist es problemlos möglich, dass mehrere `VNode` Objekt in unterschiedlichen Tabs auf ein und demselben Datenobjekt beruhen, aber z.B. unabhängig voneinander selektiert/deselektiert oder angeordnet werden können. Letzteres geschieht durch die *affine Transformation* eines jeden Knotens, die u.a. das Verschieben des Objekts auf eine beliebige Position ermöglicht. Auf diese Weise werden die Koordinaten der einzelnen Knoten in jedem `VNode` Objekt verwaltet, so dass Layout Algorithmen auf diesen ausgeführt werden können.

Um größtmögliche Flexibilität und effizientes Traversieren des Baums zu gewährleisten, hält jeder Knoten Verweise auf alle seine Kinder in einer `ArrayList` und gleichzeitig einen Verweis auf seinen Elter. Gegenüber einer Datenstruktur, die in jedem Knoten Elter, linkes Kind und den nächsten Geschwister Knoten speichert, bietet die gewählte Datenstruktur den Vorteil, dass das i-te Kind eines Knotens in konstanter Zeit ermittelt werden kann. Dieser Vorteil wird evtl. durch einen höheren Speicherverbrauch erkauft, der allerdings von der konkreten Baumstruktur abhängt. Ein weiterer Vorteil ist, dass sich viele vorgefertigte Algorithmen für `Collections` auf `ArrayList`en anwenden lassen. So muss zur Sortierung von Kindern eines Scaffolds nur noch ein Vergleichskriterium (`Comparator`) angegeben werden, das vom vorgefertigte Algorithmus verwendet werden kann, um die Ordnung der Elemente zu ermitteln. Da der Speicherverbrauch eines jeden Knotens von seiner grafischen Darstellung dominiert wird, stand dieses Argument bei der Entscheidung für die Datenstruktur weniger im Vordergrund.

Ein Baum (`VTree`) bestehend aus `VNodes` wird aus dem `DTree` aufgebaut, indem dieser wie vom Benutzer angefordert durchlaufen wird, wobei für jedes besuchte `DNode` Objekt eine Instanz der Klasse `VNode` erzeugt wird. Ausserdem werden die in der Datenstruktur implizit gegebene Kanten in grafische Objekte (`VEdge`) umgesetzt. Die Klasse `VTree` bietet unter anderem Methoden an, um alle Kinder oder alle Nachfolger eines Scaffolds auszuklappen. Dazu kann der Baum einfach ausgehend von dem Scaffold durchlaufen werden. Es ist aber auch möglich, den Baum um einen beliebigen, nicht-ausgeklappten Kno-

ten zu erweitern. Hierzu wird ausgehend von diesem Knoten über die Elter-Verknüpfungen der Pfad in Richtung der Wurzel so lange verfolgt, bis ein „Anschlusspunkt“ im `VTree` gefunden wurde. Es ist die Aufgabe der Klasse, den Baum um Scaffolds zu erweitern oder diese zu entfernen, wobei der Zusammenhang des Baums bewahrt bleiben muss. Gleichzeitig müssen die Scaffolds auch in den Szenengraph von Piccolo eingehängt bzw. aus diesem entfernt werden. Nach allen Veränderungen im Scaffoldbaum wird das Layout neu berechnet.

3.4 Aus Sicht des Anwenders

3.4.1 Anwendungsszenarien

Um den Umgang eines Chemikers mit dem CBSE zu veranschaulichen, sollen hier drei denkbare Anwendungsszenarien beschrieben werden.

3.4.1.1 Szenario 1

In diesem Szenario verschafft sich der Nutzer einen Überblick über den Datensatz bezüglich einer gewissen Eigenschaft und grenzt schnell die für ihn interessanten Scaffolds ein. Der Benutzer wählt bei Programmstart geeignete Filtereigenschaften für den Filter aus und lässt sich einen ersten großen Startgraphen anzeigen. Er möchte die große Scaffold-Startmenge auf eine überschaubare Menge reduzieren, dazu färbt er die Scaffolds nach gewünschten Eigenschaften ein. Die eingefärbten Scaffolds werden jetzt alle ausgewählt und in einem neuen Tab geöffnet. So erhält der Benutzer einen kleineren Graphen, der nur noch die für ihn relevanten Knoten enthält. Um die Menge der „interessanten“ Scaffolds noch weiter einzugrenzen, wählt der Nutzer diejenigen Scaffold per Hand aus, die Hotspots darstellen, also Knoten mit einem hohen Färbungsgrad. Diese Scaffolds lassen sich wiederum in einem neuen Tab anzeigen und er erhält so alle Scaffolds, die für ihn am wichtigsten sind. Diese „handliche“ Menge kann der Nutzer jetzt exportieren.

3.4.1.2 Szenario 2

Der Nutzer möchte einen schnellen Überblick über einen Datensatz gewinnen. Er möchte einen importierten Datensatz auf seine Struktur hin untersuchen und ein Gefühl dafür bekommen, welche Scaffolds mit welchen Eigenschaften enthalten sind. Der genaue Vorgang ist im Grunde der gleiche wie in Szenario 1. Die Filtereigenschaften werden gesetzt, die Knoten des Startgraphen werden nach ausgewählten Eigenschaften eingefärbt und in einem neuen Tab angezeigt. Dann werden alle Scaffolds per Hand markiert, die einen besonders hohen Färbungsgrad haben, d.h. besonders viele Moleküle haben die gewünschte Eigenschaft. Diese werden dann wieder in einem neuen Tab geöffnet. So kann der Nutzer die Eigenschaftsverteilung auf einem Datensatz erkennen.

3.4.1.3 Szenario 3

Der Nutzer möchte zwei Datensatze auf ihre Schnittmenge hin untersuchen, zum Beispiel als Unterstützung für eine Kaufentscheidung. Der Nutzer möchte den angebotenen Datensatz mit seinem eigenen vergleichen und so erkennen, welche Strukturen sein Datensatz bereits enthält und welche er vom Anbieter noch kaufen könnte. So ließen sich auch mehrere Anbieter miteinander vergleichen.

Insgesamt ist der große Vorteil des CBSE, dass man einen sehr großen Datensatz in kurzer Zeit auf eine überschaubare Menge von interessanten Elementen eingrenzen und reduzieren kann.

3.5 Ausblick

Am Anfang der Projektgruppe wurde ein Pflichtenheft erstellt, in dem Anforderungen an das Programm definiert wurden. Während der wöchentlichen Sitzungen fanden sich immer wieder neue Möglichkeiten, die in das Programm integriert werden konnten, um dem Benutzer mehr Anwendungsmöglichkeiten zu bieten. Auch wurde den Teilnehmern der PG504 zwischendurch weitere Wünsche von den Chemikern nahegelegt. Diese wurden besprochen und einige davon abgewiesen, da sie nach deren Meinung nicht der Produktivität des Programmes entsprachen oder nicht umsetzbar waren. Die meisten Punkte, die noch zum Anfang des 2. Semester offen waren, wurden von den Teilnehmern der PG504 so gut wie alle zum Semesterende hin abgearbeitet und integriert. Der eigentliche Punkt, der noch aus den Anforderungen, bzw. Pflichtenheft übrig geblieben ist, war die Smiles- und Smart-Suche. Da das Programm CBSE aber der Öffentlichkeit zur Verfügung gestellt werden soll, könnten diese Funktionen später noch integriert werden.

3.6 Lizenz

Nach einer längeren Diskussion über die verschiedenen Lizenzen haben sich die Teilnehmer in einer Abstimmung (über das Forum) dazu entschieden, ihr Programm mit der GPL v3 Lizenz zu versehen. CBSE hat also eine Lizenzierung als freie Software mit Copyleft. Die Details sollte man der Lizenz entnehmen und darauf wird hier nicht weiter eingegangen. Die Einverständniserklärung der Studenten liegt vor.

4 Fazit

4.1 Das eigentliche Fazit

Innerhalb eines Jahres entstand aus der Idee zum CBSE eine einsetzbare Software. In dieser Zeit konnte die PG504 den Großteil der zu Beginn festgelegten Funktionen umsetzen.

Diese Entwicklung geschah nicht immer gradlinig; den Teilnehmern der PG504 bot sich die Möglichkeit, verschiedene Umsetzungsoptionen auszuprobieren und sich im weiteren Verlauf der Projektentwicklung für eine Lösung zu entscheiden. Nicht nur die unterschiedlichen Ansätze, sondern auch Fehleinschätzungen und Missverständnisse unter den Projektteilnehmern trugen dazu bei. Durch diesen Verlauf lernte die PG504, an welchen wichtigen Punkten eines Projekts Schwierigkeiten zu erwarten sind und warum eine gründliche Planung und Kommunikation bestimmter Aspekte wichtig für dessen Entwicklung sind.

Der CBSE wurde während seiner Entwicklung vielen Interessenten vorgestellt. Obwohl zu den verschiedenen Zeitpunkten nur Teile der Funktionen, die der CBSE bei seiner Fertigstellung zur Verfügung stellen sollte, implementiert waren, erfuhr die PG504 stets eine positive Rückmeldung seitens der Anwender und Zuschauer. Diese Resonanz spornte die Umsetzung bis zum Ende der PG504 an.

Die Teilnehmer der Projektgruppe freuen sich auf die weitere Entwicklung des von ihnen erarbeiteten CBSE.

4.2 Resümee der Teilnehmer

Werden die einzelnen Mitglieder der Projektgruppe gefragt, warum sie sich für dieses Projekt im Besonderen entschieden haben, stellen sich viele Gemeinsamkeiten heraus: Erstgenannt sind sowohl das Interesse an Themen zur Visualisierung im Allgemeinen und der Wunsch, an einem Projekt mitarbeiten zu können, welches das Potential hat, ein in der Praxis einsetzbares Werkzeug zu werden. *„Von vielen abgeschlossenen PGs hört man, dass diese in irgendeiner Schublade verschwinden; darauf hatte ich keine Lust und sah im CBSE die Möglichkeit, etwas Praktikables zu entwickeln, was auch später - ob in der Forschung oder Wirtschaft - Verwendung finden kann“* (Phillipp)

Das Projekt bot die Möglichkeit nicht nur eine konkrete Problemstellung zu bearbeiten und nach ihrer Lösung ein abgeschlossenes Konstrukt abzuliefern, sondern eine Software zu entwickeln, die über die Zeit der PG hinaus Bestand hat und weiterentwickelt werden kann. *„Das Konzept des CBSE bot von Anfang an eine Menge Features, die man implementieren konnte; mehr, als unsere Projektgruppe in einem Jahr schaffen konnte. Das wurde uns sehr schnell klar, und durch den strukturierten Aufbau der einzelnen Softwarebereiche haben wir es geschafft, eine solide Basis zu entwickeln, die ohne Umstände weiterentwickelt werden kann“* (Henning)

Nach den bekannten Zielen und Anforderungen des Projekts ging es primär um eine adäquate Anwendung vorhandener Technologien und Ressourcen. Verschiedene Toolkits, Visualisierungsmöglichkeiten und unterschiedliche Optionen der Implementierung mussten recherchiert und ihre Verwendungsmöglichkeit innerhalb des Projekts evaluiert werden. *„Gleich bei der Projektvorstellung wurde betont, dass es um die Visualisierung unheimlich grosser Datenmengen ging. Mein Interesse wurde mitunter dadurch geweckt. ästhetische bzw. übersichtliche Bilder zu zeichnen, das war schon Herausforderung genug; dazu noch eine performante Lösung zu finden, spornte mich zusätzlich an.“* (Adalbert)

Des Weiteren hat für viele Teilnehmer die Möglichkeit zum „Blick über den Tellerrand“ in Bezug auf die Spezialisierung im biochemischen Bereich zu der Entscheidung für die PG beigetragen. Die Einarbeitung in diese Thematik bot einen interessanten Einblick in die Anforderungen von Biochemikern an Software und Technologien.
Das Thema des Projektes entstand aus einer Zusammenarbeit des Max-Planck-Institutes mit der Firma Novartis | durch sie wurde die Entwicklung zum CBSE angestossen. Damit wurde ein Auftraggeber mit in die Planung einbezogen, der wenig Wissen über die technischen Möglichkeiten und Grenzen zur Visualisierung des biochemischen Strukturraumes hatte, jedoch eine genaue Vorstellung von den im CBSE zu enthaltenden Funktionen mit einbringen konnte. Dadurch entstand eine Dynamik in der Entwicklung, bei der verschiedene Konzepte sowohl in der Theorie als auch in der Praxis erarbeitet und durch den regen Austausch der beiden Parteien ausgearbeitet sowie optimiert wurde. Durch diesen Entwicklungsprozess konnten die Projektmitglieder einen Einblick in die Vorgehensweise bei der Entstehung eines Softwareprojektes mit einem Projektpartner erhalten.

5 Anhang

5.1 Schnittstellen

Während der Planungsphase des Projektes hatten die Teilgruppen die Aufgabe, Schnittstellen für jeden Progammteil zu definieren. Im Laufe der Realisierung des Projektes wurden diese überarbeitet und ergänzt. Die folgende Auflistung enthält alle Schnittstellen der aktuellen Programmversion.

5.1.1 Visualisierung

Innerhalb des Paketes `vis` wird die Zeichenfläche eines Graphen implementiert und verwaltet. Es müssen Schnittstellen angeboten werden, um eine solche Zeichenfläche zu erzeugen, um die Einstellungen des Nutzers bezüglich der Art der Darstellung des Graphen zu übergeben und um die Navigation innerhalb des Graphen zu steuern, die der Nutzer nicht direkt über die Zeichenfläche, sondern über andere Elemente der GUI anstösst.

Schnittstellen der Klasse `VISControl`

Methoden für den Mouseover

- `setMouseoverDelay(int millisec) : void`
  Legt die Wartezeit fest, bis das Mouseoverfenster erscheint. Defaultwert: 1000 Millisekunden.
- `getMouseoverDelay () : int`
  Gibt die Zeit aus, die es dauert, bis das Mouseoverfenster erscheint.

Methoden für die Darstellung

- `setHideSubtreeEdges(boolean enable) : void`
  Aktiviert/Deaktiviert die Anzeige der Kanten auf der ersten Ebene bei der Anzeige eines Unterbaumes.
- `setNodeStrokeWidth(double stroke_width) : void`
  Legt die Strichstärke der Knoten fest. Defaultwert: 2 Pixel.
- `setSelectedNodeStrokeWidth(double selected_stroke_width) : void`
  Legt die Strichstärke der selektierten Knoten fest. Defaultwert: 3 Pixel.
- `setEdgeStrokeWidth(double stroke_width) : void`
  Legt die Strichstärke der Kanten fest. Defaultwert: 1,5 Pixel.
- `setSelectedEdgeStrokeWidth(double selected_stroke_width) : void`
  Legt die Strichstärke der selektierten Kanten fest. Defaultwert: 2 Pixel.
- `setCircleStrokeWidth(double stroke_width) : void`
  Legt die Strichstärke der Kreise des radialen Layouts fest. Defaultwert: 1 Pixel.
- `setCursorStrokeWidth(double stroke_width) : void`
  Legt die Strichstärke des Cursors fest. Defaultwert: 6 Pixel.
- `setSelectedNodeColor(Color color) : void`
  Legt die Farbe der selektierten Knoten fest. Defaultfarbe: rot.
- `setCursorColor(Color color) : void`
  Legt die Farbe des Cursors fest. Defaultfarbe: blau.
- `setEdgeColor(Color color) : void`
  Legt die Farbe der Kanten fest. Defaultfarbe: dunkelgrün.

- `setSelectedEdgeColor(Color color) : void`
  Legt die Farbe der selektierten Kanten fest. Defaultfarbe: rot.
- `setCircleColor(Color color) : void`
  Legt die Farbe der Kreise des radialen Layouts fest. Defaultfarbe: hellgrau.

Methoden für die Animation

- `setCameraAnimation(boolean enabled) : void`
  Aktiviert/Deaktiviert die Animation von Kamerabewegungen.
- `setLayoutAnimation(boolean enabled) : void`
  Aktiviert/Deaktiviert die Animation von Layoutveränderungen.
- `setCursorAnimation(boolean enabled) : void`
  Aktiviert/Deaktiviert die Animation von Cursorbewegungen.

Methoden für die Fokussierung

- `setFocusAfterAction(boolean enabled) : void`
  Falls aktiviert, wird nach dem Ausklappen eines Knotens die Kamera wieder auf diesen zentriert.

Methoden für die History

- `setMaxHistoryLength(int newlength) : void`
  Setzt die maximale Anzahl von Wiederherstellungspunkten der History.

Schnittstellen der Klasse `VCanvas`

Methoden für die Initialisierung

- `createFullSubtree(int scaffoldID) : VCanvas`
  Gibt einen neuen `VCanvas` zurück, der den kompletten Unterbaum des Scaffolds mit der übergebenen `scaffoldID` enthält. Dabei werden die notwendigen Daten in einem Worker Thread geladen, so dass sofort ein leerer `VCanvas` zurückgegeben werden kann.
- `createTree(int depth, ArrayList<Filter> filter) : VCanvas`
  Erzeugt einen neuen `VCanvas` und einen neuen Datensatz nach dem übergegebenen Filter. Nach der Initialisierung werden die ersten `depth` Ringe ausgeklappt.
  (Worker Thread wie bei `createFullSubtree`)
- `createSubtree(int scaffoldID, int depth, VCanvas canvas) : VCanvas`
  Erzeugt einen neuen `VCanvas` mit einem Unterbaum, der bei dem Scaffold mit `scaffoldID` beginnt. Datensatz und Filtereinstellungen werden von `canvas` übernommen. Es werden die ersten `depth` Ringe ausgeklappt.
- `createCopy(VCanvas canvas) : VCanvas`
  Erzeugt eine Kopie des übergebenen `VCanvas`, wobei Filtereinstellungen und Datensatz direkt übernommen werden und nicht kopiert.
- `createSelectionTree(Collection<Integer> scaffoldIDs,VCanvas canvas) : VCanvas`
  Der erzeugte `VCanvas` stellt einen Baum mit allen Scaffolds aus der `Collection` dar. Damit der Baum zusammenhängend ist, werden evtl. zusätzliche Knoten eingefügt. Der Datensatz und die Filtereinstellungen werden direkt von `canvas` übernommen. Sollte der Datensatz einen Knoten in der `Collection` nicht enthalten, wird dieser ignoriert.

Methoden für das Rendering

- `setRenderingQuality(int newquality) : void`
  Methode um die Renderqualität zu beeinflussen. Als Werte für `newquality` gibt es in der `VISControl` drei Konstanten, zu erreichen folgendermassen:
  - `VISControl.RENDERING_QUALITY_LOW` (= Anti-Aliasing immer aus)
  - `VISControl.RENDERING_QUALITY_HIGH` (= Anti-Aliasing immer an)
  - `VISControl.RENDERING_QUALITY_AUTO` (= Anti-Aliasing aus bei Animation und Interaktion, ansonsten an)

Methoden für Zoom und Fokus

- `focusOn(String id) : void`
  Fokussiert Knoten mit `id`, klappt benötigte Kinder gegebenenfalls aus.

- `zoomIn(Point2D viewzoompoint) : void`
  Zoomt einen Standardwert in den Punkt `viewzoompoint` hinein.

- `zoomOut(Point2D viewzoompoint) : void`
  Zoomt einen Standardwert aus dem Punkt viewzoompoint heraus.

- `zoomToOverview() : void`
  Zoomt zur Übersicht des gesamten Graphen im `VCanvas`.

- `setZoomFactor(Point2D viewzoompoint, int zoomfactor) : void`
  Legt den Zoomfaktor direkt fest. Dieser kann Werte zwischen 0 und 500% annehmen.

- `getZoomFactor (): int`
  Liefert den Zoomfaktor in Prozent zurück.

Methoden für die Darstellung

- `scaleNode(VNode v, double factor) : void`
  Skaliert den Knoten `v` um den Faktor `factor` und fokussiert den Knoten ohne Animation. Ein Faktor von 1.0 entspricht 100%, also der aktuellen Knotengrösse.

- `scaleAllNodes(double factor) : void`
  Skaliert alle Knoten um den Faktor `factor`.

- `scaleSelectedNodes(double factor) : void`
  Skaliert die selektierten Knoten um den Faktor `factor`.

- `normalizeNode(VNode v) : void`
  Normalisiert die Skalierung des Knoten `v`, d.h. setzt den Knoten auf die ursprüngliche Grösse zurück.

- `normalizeAllNodes () : void`
  Normalisiert die Skalierung aller Knoten.

- `normalizeSelectedNodes () : void`
  Normalisiert die Skalierung der selektierten Knoten.

Methoden für das Layout

- `expand(String scaffoldID) : boolean`
  Klappt alle Kinder des Knotens mit `scaffoldID` aus. Gibt `true` zurück, falls erfolgreich.
- `isExpandable(String scaffoldID) : boolean`
  Gibt `true` zurück, falls Kinder des Knotens mit `scaffoldID` ausgeklappt werden können.
- `expandNextLevel(String scaffoldID) : boolean`
  Klappt die nächsten Ebene des Knotens mit `scaffoldID` komplett aus. Gibt `true` zurück, falls erfolgreich.
- `expandSubtree(String scaffoldID) : boolean`
  Klappt den kompletten Unterbaum auf, der an dem Knoten mit `scaffoldID` hängt. Gibt `true` zurück, falls erfolgreich.
- `reduce(String scaffoldID) : boolean`
  Klappt alle Kinder (und deren Nachfolger) des Knotens mit `scaffoldID` ein. Gibt `true` zurück, falls erfolgreich.
- `isReducible(String scaffoldID) : boolean`
  Gibt `true` zurück, falls Kinder des Knotens mit `scaffoldID` eingeklappt werden können.
- `reduceToThisLevel(String id) : void`
  Klappt alle Knoten ein, die Nachfolger von Knoten des Ringes sind, auf dem sich der Knoten mit `id` befindet.
- `sort(int scaffoldID) : void`
  Sortiert die Scaffolds nach Ähnlickeit zu dem Scaffold mit `scaffoldID`.

Methoden für die Auswahl

- `deselectNodes() : void`
  Wählt alle Knoten im `VCanvas` ab.
- `getSelectedNodes() : LinkedList<VNode>`
  Gibt eine `LinkedList` mit allen ausgewählten `VNodes` des `VCanvas` zurück.
- `getSelectedNodesAsArrayList() : ArrayList<String>`
  Gibt eine `ArrayList` mit Scaffold-IDs der ausgewählten `VNodes` des `VCanvas` zurück.
- `selectColoredNodes() : void`
  Selektiert alle durch einen Filter gefürbten Knoten.

Methoden für das Filtern

- `colorNodes(String[] nodeIDs, int[] graduations, Color color) : void`
  Färbt eine Liste von Knoten ein: Erster Parameter ist ein Array der Scaffold-IDs der Knoten, die eingefürbt werden sollen. Zweiter Parameter ist ein Array der Färbungsgrade der einzelnen Knoten in Prozent. `color` gibt die Grundfarbe an.
- `uncolorNodes() : void`
  Entfürbt alle Knoten des `VCanvas`.
- `getColoredNodes() : LinkedList<ColoredNode>`
  Gibt eine `LinkedList` mit allen gefürbten `VNodes` des `VCanvas` zurück.
- `setFilter(ArrayList<Filter> filter) : void`
  Wendet den `filter` auf den aktuellen Datensatz an. Dieser darf nur weniger Scaffolds erlauben als der aktuelle Filter. Herausfallende sichtbare Scaffolds werden entfernt.

Methoden für die History

- `historyBackward() : boolean`
  Undo: Setzt auf den letzten (automatisch gesetzten) Checkpoint zurück (falls möglich).
- `historyForward() : boolean`
  Redo: Setzt auf den nächsten (automatisch gesetzten) Checkpoint vor (falls vorhanden).
- `getBackwardPossible() : boolean`
  Hiermit kann man erfahren ob es möglich ist einen *Undo* durchzuführen.
- `getForwardPossible (): boolean`
  Hiermit kann man erfahren ob es möglich ist einen *Redo* durchzuführen.
- `getBackwardActionCode() : int`
  Gibt einen `int`-Wert zurück, der die *Undo*-Aktion beschreibt, die als nächstes rückgängig gemacht werden kann. Die Codes finden sich in der Klasse `VHistory`.
- `getForwardActionCode() : int`
  Gibt einen `int`-Wert zurück, der die *Redo*-Aktion beschreibt, die als nächstes wiederhergestellt werden kann. Die Codes finden sich in der Klasse `VHistory`.

Methoden für den Export und das Drucken

- `exportPaintScreen(Graphics g, Rectangle2D paintArea) : Rectangle2D`
  Zeichnet den aktuellen Ausschnitt in `g`, ohne Semantic Zoom und ohne +/- Symbole. Mit dem Parameter `paintArea` kann ein Rechteck angegeben werden, in das gezeichnet wird. Die Ausgabe wird also so verschoben und skaliert, dass sie komplett innerhalb des Rechtecks liegt. Zurückgegeben wird ein Rechteck, das den tatsächlich verwendeten Zeichenbereich angibt (dieser unterscheidet sich entweder in der Höhe oder der Breite von `paintArea`, da das Seitenverhältnis nicht geändert wird).
- `exportPaintAll(Graphics g, Rectangle2D paintArea) : Rectangle2D`
  Zeichnet den gesamten Graphen (soweit ausgeklappt) in `g`, ohne Semantic Zoom und ohne +/- Symbole.
- `exportSVGScreen() : SVGDocument`
  Gibt aktuelle Ansicht im SVG Format zurück (ohne Semantic Zoom, +/- Symbole).
- `exportSVGAll() : SVGDocument`
  Gibt den gesamten Graphen im SVG Format zurück (ohne Semantic Zoom, +/- Symbole).
- `Dimension getExportAllDimension() : Dimension`
  Gibt die Höhe und Breite des gesamten Graphen als `Dimension` zurück.
- `Dimension getExportScreenDimension() : Dimension`
  Gibt die Höhe und Breite des aktuellen Ausschnitts als `Dimension` zurück.

Schnittstellen der Klasse `VMiniMap`

Ein Objekt von `VMiniMap` muss von der GUI erstellt werden.

Methoden für den Connect/Disconnect

- `connect(VCanvas canvas) : void`
  Die `VMiniMap` stellt `canvas` dar. Die Methode kann bei jedem Tabwechsel und nach dem Erstellen eines neuen Tabs benutzt werden.

- `disconnect() : void`
  Die `VMiniMap` wird nicht länger aktualisiert. Diese Methode sollte aufgerufen werden, wenn die `VMiniMap` eingeklappt wird. Vor dem Ausklappen muss natürlich wieder `connect` aufgerufen werden.

Schnittstellen der Klasse `VMagnifyingMap`

Ein Objekt von `VMagnifyingMap` muss von der GUI erstellt werden.

Methoden für den Connect/Disconnect

- `connect(VCanvas canvas) : void`
  Die `VMagnifyingMap` stellt `canvas` dar. Die Methode kann bei jedem Tabwechsel und nach dem Erstellen eines neuen Tabs benutzt werden.
- `disconnect() : void`
  Die `VMagnifyingMap` wird nicht länger aktualisiert. Diese Methode sollte aufgerufen werden, wenn die `VMagnifyingMap` eingeklappt wird. Vor dem Ausklappen muss natürlich wieder `connect` aufgerufen werden.

Methoden für das Zentrieren und den Zoom

- `centerOnMouse() : void`
  Schaut nach, wo sich der Mauszeiger befindet; ist dies auf dem `VCanvas`, dann zentriert er den PoV der `VMagnifyingMap` darauf.
- `dispose() : void`
  Löscht die `VMagnifyingMap`.
- `getZoomFactor() : double`
  Gibt den Vergrösserungsfaktor zurück.
- `setZoomFaktor(double) : void`
  Setzt den Zoomfaktor.
- `setBound(int x,int y,int h,int w) : void`
  Über die vier Ecken wird der Bildausschnitt, den die `VMagnifyingMap` vergrössert, definiert.
- `setRenderingQuality(int) : void`
  Setzt die Darstellungsqualität.

Schnittstellen der Klasse `VLayout`

Methoden für die Darstellung

- `updateRadii(double delta) : void`
  Beim Radial und Linear Layout wird der Abstand zwischen den Hierarchieebenen um den Wert `delta` verändert.
  BeimBallon Layout hat diese Methode keine Funktion.
- `setFixedLayout(boolean enable) : void`
  Aktiviert/Deaktiviert die Fixierung der Radiien. Wenn die Radien fixiert sind, werden sie beim zoomen nicht dynamisch mit dem Zoomfaktor skaliert.

5.1.2 GUI

Innerhalb des Paketes `gui` werden alle Elemente der graphischen Benutzeroberfläche verwaltet, ausgenommen die Zeichenfläche des Graphen. Es müssen Schnittstellen angeboten werden, um die Benutzeraktionen innerhalb der Zeichenfläche weiterverarbeiten zu können.

Schnittstellen der Klasse `GUIController`

- `showContextMenu(Component invoker, Point coordinates, String id) : void`
  Zeigt ein Kontextmenü auf der angegebenen Komponente an den übergebenen Koordinaten an. Die Aktionen des Kontextmenüs beziehen sich auf das Scaffold mit der angegebenen ID.

- `showMouseover(Component invoker, Point coordinates, String id) : void`
  Zeigt ein Mouseover auf der angegebenen Komponente an den übergebenen Koordinaten an.

- `hideMouseover() : void`
  Blendet den vorher mit `showMouseover` eingeblendeten Mouseover wieder aus.

- `scaffoldClicked(String id) : void`
  Benachrichtigt die GUI, wenn Scaffolds angeklickt (selektiert/deselektiert) werden.

- `showMolecules(String id) : void`
  Zeigt die zu einem Scaffold gehörenden Moleküle in einem neuen Tab an.

- `setStatusMessage(String msg) : void`
  Zeigt die übergebene Nachricht in der Statuszeile an.

- `clearStatusMessage() : void`
  Löscht die angezeigte Nachricht.

- `initStatusProgress(int min, int max) : void`
  Setzt die Minimal- und Maximalwerte der Progressbar.

- `setStatusProgress(int value) : void`
  Setzt den aktuell anzuzeigenden Wert in der Progressbar.

- `setStatusProgressText(String msgKey) : void`
  Setzt den Text, der im ProgressDialog angezeigt werden soll.

- `clearStatusProgress() : void`
  Löscht die Anzeige der Progressbar.

- `startStatusProgressIndeterminate() : void`
  Aktiviert den Indeterminate-Modus der Progressbar, d.h. die Progressbar zeigt an, dass Aktionen stattfinden, aber nicht, wie weit sie fortgeschritten sind.

- `stopStatusProgressIndeterminate() : void`
  Beendet den Indeterminate-Modus.

- `getCurrentVCanvas() : VCanvas`
  Gibt den im aktuellen Tab angezeigten `VCanvas` zurück.

- `enableHistoryForward(boolean enable) : void`
  Aktiviert den *Vorwärts*-Button bei der Übergabe von `true`, deaktiviert ihn bei der Übergabe von `false`.

- `enableHistoryBackward(boolean enable) : void`
  Aktiviert den *Zurück*-Button bei der Übergabe von `true`, deaktiviert ihn bei der Übergabe von `false`.
- `updateZoomFactor(double zoomFactor) : void`
  Setzt den in der Zoom-Combobox angezeigten Zoomfaktor auf den übergebenen Wert.
- `getHomeID() : int`
  Liefert die ID des Home-Moleküls zurück.
- `setSelectedScaffoldsList(ArrayList<Integer> scaffolds) : void`
  Benachrichtigt die GUI, dass mehrere Scaffolds markiert wurden.
- `toggleMouseover() : void`
  Wechselt den *Ausgewählt*-Status des *Mouseover*-Buttons.
- `toggleFixRadius() : void`
  Wechselt den *Ausgewählt*-Status des *Radius fixieren*-Buttons.
- `switchLayoutTo(int layout) : void`
  Ändert das Layout des Graphen in das übergebene Layout:
  - 0: Radial-Layout
  - 1: Balloon-Layout
  - 2: Linear-Layout
- `showConnectionErrorMessage() : boolean`
  Informiert den Benutzer über einen Verbindungsfehler und fragt ihn, ob die Operation wiederholt oder das Programm beendet werden soll. Liefert `true` zurück, falls die Operation wiederholt werden soll, `false`, falls das Programm beendet werden soll.

5.1.3 Datenbank

Innerhalb des Paketes `db` werden alle Anfragen an die Datenbank durchgeführt. Dafür müssen entsprechende Schnittstellen zur Verfügung gestellt werden. Ausserdem wird innerhalb dieses Paketes die Datenstruktur für den darzustellenden Graph aufgebaut. Es müssen Schnittstellen angeboten werden, um für die Zeichnung relevante Informationen über die einzelnen Knoten abzufragen.

Schnittstellen der Klasse `DBController`

- `setWhiteList(LinkedList<String> whiteList) : void`
  Erstellt eine Liste von Knoten, die immer angezeigt werden, egal wie der Filter ist. Die `Strings` müssen *canonical smile strings* sein.
- `setWhiteListActive(boolean activate) : boolean`
  Aktiviert/Deaktiviert die `WhiteList`.
- `getInstance() : DBController` Gibt eine Referenz des instanziierten `DBControllers` zurück. Sollte keine Instanz vorhanden sein, wird eine angelegt und zurückgegeben.
- `setConnection(String dbUrl, String scafSchema, String user, String pass) : void`
  Versucht unter der angegebenen Url mit dem entsprechenden Nutzernamen und Passwort eine Verbindung zu einer Datenbank aufzubauen.
- `getFullSubtree(int scaffoldId) : DBTree`
  Diese Methode liefert einen kompletten Baum aus Scaffolds, ohne Filter.
- `getMaintree():DBTree`
  Diese Methode liefert den Baum aus Scaffolds im ersten Tab.
- `getSVGDocument(int scaffoldId) : SVGDocument`
  Liefert das SVG des Scaffolds im `SVGDocument`-Format.
- `getStructureSVGDocument(int structureId) : SVGDocument`
  Liefert das SVG der Struktur als `SVGDocument`.
- `getScaffoldProperties(int scaffoldId) : HashMap<String, ArrayList>`
  Gibt alle Eigenschaften des Scaffolds - wie auch immer diese aussehen werden - zurück.
- `getScaffoldPropertiesDefinition() : ArrayList<ScaffoldProperty>`
  Gibt in einer `ArrayList` die Eigenschaften von Scaffolds samt Beschreibung zurück.
- `getScaffoldMinMax(String numPropertyID, String numScaffoldThing) : Float[]`
  Die Parameter sind:
  - `num_property_id` : Hier kommt die ID rein, wie z.B. *Number of Points*
  - `num_scaffold_thing` : Hier kommt rein, was man haben möchte, z.B. *num_scaffold_molecules*

  Liefert an Stelle [0] den MIN-Werte zurück und an Stelle [1] den MAX-Wert zurück.
- `getStructureProperties(String structureId) : HashMap<String, String>`
  Gibt in einer `HashMap` die Eigenschaften von Molekülen samt Beschreibung zurück. Das Format ist vom Typ: `<String, String>`.

- `getStructurePropertyDefinition(String propertyId): ArrayList <String>`
  Liefert eine Beschreibung über die gegebene Eigenschaft.
- `getScaffoldStructures(int scaffoldId): ArrayList<String>`
  Liefert die Strukturen zum angegebenen Scaffold als Liste der Structure-IDs.
- `countScaffoldsWithoutProperties() : int`
  Liefert die Zahl der Scaffolds, die über keine Properties verfügen und deshalb nicht in gefilterten Graphen vorkommen.
- `countScaffoldsWithProperties(ArrayList<String> propertyIds) : int`
  Liefert die Zahl der Scaffolds, für die Werte in ALLEN übergebenen Eigenschaften (`propertyIds`) verfügbar sind.
- `checkScaffoldFilter(ArrayList<Filter> filterList) : int`
  Gibt die Zahl der Scaffolds zurück, die den Filtereinstellungen entsprechen.
- `setScaffoldFilter(ArrayList<Filter> filterSettings) : void`
  Aktiviert den Filter und stellt diesen entsprechend der Filter-Objekte ein.
- `setScaffoldFilter(boolean active) : boolean`
  Je nach übergebenem Wahrheitswert wird der Filter (de)aktiviert.
- `getFilteredGraph() : void`
  Liest die Scaffolds aus, die die Filter-Anforderungen erfüllen. Baut dann aus diesen bottom-up einen neuen Graphen auf.
- `getScaffoldID(String smiles) : int`
  Liefert die zum SMILES-`String` passende ID des Scaffolds.
- `getScaffoldSMILES(int scaffoldId) : String`
  Liefert den zur ID passenden SMILES-`String`.
- `loadProfile(Profile profile) : void`
  Lädt ein Profil aus der Datenbank.
- `saveProfile(Profile profile) : void`
  Speichert das Profil in der Datenbank.
- `deleteProfile(String profileName) : void`
  Löscht ein Profil aus der Datenbank.
- `getScaffoldsToColor(ColorFilter filter, int graduation) : ArrayList<Integer>`
  Gibt die Scaffolds zurück, die gemäß der übergebenen Kriterien eingefürbt werden sollen.

5.2 Pflichtenheft

Zielbestimmung

Der Chem-Bio-Space-Explorer (CBSE) ist ein Visualisierungstool zum Darstellen und Durchsuchen des chemischen Strukturraums, der als Graph visualisiert wird. Dabei bedient sich der CBSE einer SQL-Datenbank, die alle Scaffolds und Moleküle beinhaltet.

Musskriterien

Die Funktionen des CBSE können im wesentlichen in 5 Bereiche unterteilt werden:

Allgemeine Funktionen

- Favoritenfunktionen
  - Hinzufügen von Favoriten(Scaffolds) in eine Favoritenliste
  - Löschen von Favoriten(Scaffolds) aus der Favoritenliste
  - Verwalten von Favoriten in der Favoritenliste

Bewegen im Graphen

- Zoom und Pan Funktionen
- Home-Button um den Graphen auf einen vom Benutzer eingestellten Knoten zu zentrieren
- Auf einen Favoriten fokussieren

Interaktion mit dem Graphen

- Scaffolds auswählen/abwählen
- Ast ab bestimmtem Knoten auf/zuklappen
- Molekülgruppe eines Scaffolds anzeigen
- Teilgraphen ab gewählten Knoten in neuem Fenster zeigen
- Eigenschaften anzeigen
- Mouseover(zusätzliche Informationen zu Scaffolds)
- Suchen nach Molekülen (SMILES-Suche)
- Highlighting der Scaffolds nach vom Benutzer ausgewählten Kriterien (Filter-Funktion)

Datenbank

Darüber hinaus soll eine Anbindung an eine MySQL-Datenbank erstellt werden. Diese Datenbank enthält alle Daten zum chemischen Strukturraum, der mit dem CBSE untersucht werden soll.

Layout

Der Graph soll trotz hoher Anzahl an Knoten übersichtlich bleiben. Es soll dem Benutzer ein Höchstmass an Informationsdichte angeboten werden, dies darf aber nicht auf Kosten der ̈bersichtlichkeit und Orientierung im Graphen geschehen.

Wunschkriterien

- Verschiedene Layouts auswählbar

Abgrenzungskriterien

- Anzeigen des gesamten Strukturraumes

Produkteinsatz

Anwendungsbereiche

Die Applikation dient der Untersuchung des chemischen Strukturraumes.

Zielgruppen

Chemieunternehmen, die sich mit Molekularbiologie beschäftigen Personen, die über eine SQL-Datenbank verfügen und im Bereich Molekularbiologie forschen.

Betriebsbedingungen

- Tägliche Betriebszeit: 30 Min - 10 Std
- Häuslicher Computerarbeitsplatz

Produktübersicht

Zur Übersicht soll eine Liste der Anwendungsfälle dienen:

- Favoriten hinzufügen
- Favoriten löschen
- Favoriten verwalten
- Programm beenden
- im Graphen navigieren
- im Graphen zoomen
- auf das Home-Molekül zoomen
- Scaffold im Graphen auswählen/abwählen
- Ebene/Ast im Graphen aufklappen/zuklappen
- Molekülgruppe eines Scaffolds anzeigen
- Teilbaum ab ausgewähltem Scaffold anzeigen
- Eigenschaften eines Scaffolds/Moleküls anzeigen
- Mouseover

Produktfunktionen

Favoriten hinzufügen

Der Benutzer fügt die ausgewählten Moleküle zu seiner Favoritenliste hinzu.

Favoriten löschen

Der Benutzer löscht die ausgewählten Moleküle aus der Favoritenliste.

Favoriten verwalten

Der Benuter kann die Liste der Favoriten sortieren, beliebige Favoriten in Ordnern zusammenfassen, Ordner erstellen und löschen.

Programm beenden

Die Einstellungen des Benutzers weden gespeichert und das Programm wird geschlossen.

Im Graphen navigieren

Der Benutzer kann mittels linker Maustaste (Drag) den Graphen „verschieben“, und sich so im Graphen bewegen.

Im Graphen zoomen

Der Benutzer kann mittels Mausrad den Graphen näher heranzoomen oder weiter wegzoomen.

Das Home-Molekül zentrieren

Der Benutzer kann den Graphen auf ein von ihm/ihr eingestelltes Home-Molekül zentrieren.

Scaffold auswählen/abwählen

Der Benutzer kann mittles Mausklick (links) ein Scaffold zur Menge der selektierten Scaffolds hinzufügen oder aus der Menge der selektierten Scaffolds entfernen.

Ebene/Ast im Graphen auf-/zuklappen

Der Benutzer kann mittels Mausklick(links) auf das (-)-Symbol neben einem Scaffold den Teilgraphen, der von diesem Scaffold ausgeht zuklappen, oder falls das Scaffold keine Nachfolger hat, eine Hierarchieebene aufklappen.

Molekülgruppe eines Scaffolds anzeigen

Der Benutzer kann mittels Mausdoppelklick (links) auf ein Scaffold eine Liste mit den Molekülen aufrufen, die zu diesem Scaffold zugeordnet sind.

Teilbaum ab ausgewähltem Scaffold anzeigen

Der Benutzer kann sich den Teilgraphen ab einem von ihm ausgewählten Knoten in einem neuen Fenster/Tab anzeigen lassen.

Eigenschaften eines Scaffolds/Moleküls anzeigen

Der Benutzer kann sich die Eigenschaften und verschiedene Daten zu einem gewählten Scaffold/Molekül detailiert anzeigen lassen.

Mouseover

Verharrt der Benutzer eine gewisse Zeit (z.B. 500ms) mit dem Mauszeiger über einem Scaffold, wird ein Kontextfenster neben dem Mauszeiger eingeblendet, das gemittelte Werte der Eigenschaften der darunterliegenden Moleküle enthält. Dieses Fenster schliesst automatisch, sobald der Benutzer die Maus wieder bewegt.

Produktleistungen

Performance

Die Applikation soll trotz hoher Datenmengen und groÃ¿er Anzahl an Knoten bedienbar schnell laufen.

Systemvoraussetzungen

Hardware

Mindestanforderungen

- Pentium 400-MHZ-Prozessor
- Mindestens 128 MB RAM
- Mindestens 20 MB ferfügbarer Speicherplatz auf der Festplatte
- Tastatur und Microsoft Mouse oder ein kompatibles Zeigegerät
- Videoadapter und Monitor mit Super VGA (800 x 600) oder höherer Auflösung

Empfohlen

- Pentium 1-GHz-Prozessor
- 512 MB RAM
- Grafikkarte und Monitor mit einer Auflösung von 1024 x 768

Software

- Windows 98/2000/XP
- Java JRE 1.5 / 5.0
- Internetbrowser

Sonstiges

- MySql-Datenbank
- Internetzugang/Netzwerk

Zusätzlich rein:

- Verschiedene Sprachen
- deutsches Handbuch
- Radial Layout