Automatisches Zeichnen von UML-Klassendiagrammen durch den Sugiyama-Algorithmus

Holger Eichelberger

Lehrstuhl für Informatik II, Am Hubland, 97074 Würzburg
eichelberger@informatik.uni-wuerzburg.de

Zusammenfassung SugiBib ist ein Framework, das eine allgemeine Implementierung des Sugiyama-Algorithmus bereitstellt. Es wurde ausgeprägt, um das Layout von UML-Klassendiagrammen zu berechnen. In diesem Beitrag wird das Framework und der praktische Einsatz im Software-Entwurf und in der Lehre beschrieben.

Schlüsselwörter: Sugiyama, Klassendiagramme, UML, st-Nummerierung, Manhattan-Layout, gridline-Layout, Framework, spring embedder, graph drawing, ELP, graph folding

1 Einleitung

Im Rahmen des Software-Engineering ergibt sich im praktischen Einsatz häufig die Notwendigkeit, das Layout von Software-Entwurfsdiagrammen anhand einer textuellen Beschreibung automatisch zu berechnen. Einsatz für diese Technik findet sich u.a. im inkrementellen Round-Trip-Engineering, wie auch im Re-Engineering bestehender Software.
Grundziel der Entwicklung des in JAVA geschriebenen Frameworks SugiBib war eine möglichst allgemeine Implementierung des Sugiyama-Algorithmus [10]. Besondere Aufmerksamkeit sollte auf allgemeine Entwurfsziele, wie Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit gerichtet werden. Das Framework wurde zur Berechnung des Layouts von UML-Klassendiagrammen ausgeprägt und in Form einer einfachen Applikation, die zunächst nur Eingaben in Form von UMLscript [7] bearbeitet, verfügbar gemacht. Mit der Veröffentlichung der UML Version 1.3 im Jahr 1999 wurden die Fähigkeiten der UML-Ausprägung von SugiBib modifiziert und erweitert. Seit Januar 2000 steht eine Server-Variante auf dem WWW-Server unseres Lehrstuhls für Interessierte zur Verfügung. Zudem sind diverse Erweiterungen, die einen größeren Anwendungsbereich erschließen, geplant.
Im folgenden Abschnitt werden zunächst kurz die verwendeten Algorithmen beschrieben. Danach gibt der Abschnitt „Das Framework SugiBib" einen Einblick in die Struktur unseres Frameworks. Abschließend folgt eine Übersicht über Anwendungen und zukünftige Entwicklungen.

2 Sugiyama- und Seemann-Algorithmus

In der Grundform besteht der Vorschlag von K. Sugiyama zur Visualisierung hierarchischer Datenstrukturen [10] aus einem mehrstufigen Algorithmus:

1. Die Eingabe, ein gerichteter azyklischer Graph, wird in der ersten Stufe auf einzelne Ränge verteilt. Lösungsansätze findet man in den Methoden der linearen Programmierung oder dem Netzwerk-Simplex-Algorithmus.
2. Die dabei entstandene Hierarchie wird durch eine zweite Stufe, die die Kantenkreuzungen zwischen zwei benachbarten Rängen minimiert, nachbearbeitet. „Heuristics are appropriate since minimizing edge crossings in layouts of ranked graphs is NP-complete, even for only two ranks“[3].
3. Die dritte Stufe berechnet die Koordinaten der Knoten und Kanten. Die allgemeine Lösung betrachtet ein lineares Programm, dessen Komplexität mit der Größe der zu betrachtenden Matrix ($VE+E^2$ Einträge mit V Knoten und E Kanten) wächst.

Grundlegend für eine Implementierung ist die Arbeit von Gansner et. al. [3], in der diverse Ansätze für Approximationen und Kompromisse in Form von Heuristiken vorgestellt werden. Wir verwenden den Netwerk-Simplex-Algorithmus zur Rangberechnung eines azyklischen Graphen, Median- oder Barycenter-Gewichte in einer Sortierheuristik zur Kantenkreuzungsminimierung und mehrere, teilweise interferierende Einzelheuristiken für die Koordinatenberechnung.
Ein UML-Klassendiagramm besteht aus einer Menge rechteckiger Knoten, die i.d.R. unterschiedliche Größe besitzen, Vererbungs- und Assoziationskanten (siehe Abbildung 1). Kanten und Knoten besitzen jeweils eine vertikale und eine horizontale Ausdehnung. Knoten werden in zudem in einen Außenbereich und einen Innenbereich unterteilt (siehe unten). Eine Anwendung des oben kurz skizzierten Algorithmus auf UML-Klassendiagramme ist ohne weitere Anpassungen nicht möglich, da ein Klassendiagramm i.allg. einen zyklischen Graphen darstellt.

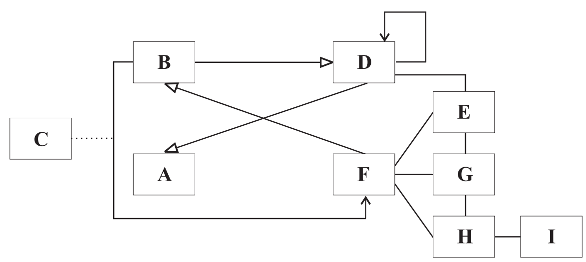

Abbildung 1. Ein beliebiges Klassendiagramm vor Anwendung des Seemann-Algorithmus. Kanten mit Pfeilen wie zwischen den Knoten D und A sind Vererbungskanten, die anderen Kanten stellen Assoziationen dar. Knoten C ist eine Assoziationsklasse dar, Knoten D besitzt eine reflexive Assoziation.

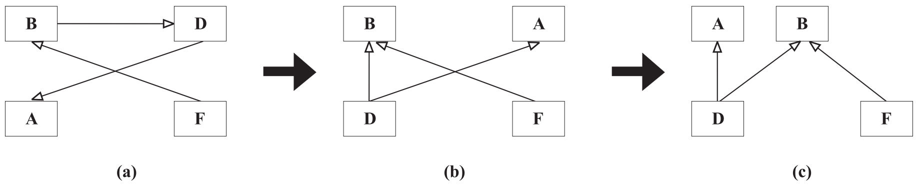

Abbildung 2. Entfernung der Assoziationskanten und zugehörigen Knoten (a), Rangzuweisung (b) und Minimierung der Kantenkreuzungen (c) am Beispielgraphen.

J. Seemann beschreibt in [9] ein geeignetes Konzept, im Folgenden kurz Seemann-Algorithmus genannt, das die Anwendung des Sugiyama-Algorithmus auf Klassendiagramme ermöglicht. Grundlegend für den Seemann-Algorithmus ist die Beobachtung, dass ein Klassendiagramm einen azyklischen Graph in Form des Vererbungsgraphen enthält. Entfernt man also temporär die Assoziationskanten aus dem Eingabegraphen, kann man die ersten zwei Schritte des Sugiyama-Algorithmus ohne Modifikationen anwenden.
Problematisch ist die Existenz reflexiver Assoziationen, also Assoziationskanten, die gleichen Start- und Endknoten besitzen. Diese stellen Zykel im Graph dar und erschweren die Beschreibung der Algorithmen. Seemann folgend, werden diese Kanten im Vorbereitungsschritt entfernt und als Information den betreffenden Knoten hinzugefügt. Die Kanteninformation kann dann z.B. in Attribute umgewandelt werden. Sollte der Graph einen Wald darstellen und keine eindeutige Wurzel besitzen, wird ein virtueller Knoten als Wurzel in den Graph eingefügt. Virtuelle Knoten sind in der Regel nicht sichtbar und dienen z.B. zur Formatierung langer Kantenketten, die nicht durch eine einzige Linie dargestellt werden können.

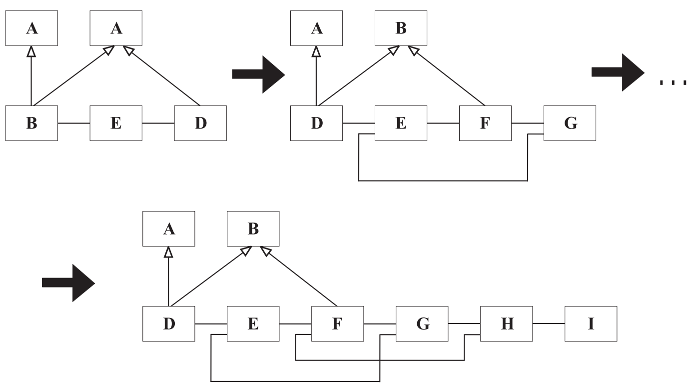

Abbildung 3. Inkrementelle Layout-Erweiterung (incremental extension) nach Seemann.

In einem zusätzlichen Schritt, der inkrementellen Erweiterung, wird das Grundlayout des Vererbungsgraphen durch die entfernten Assoziationkanten angereichert und eine Approximation des Eingabegraphen aufgebaut.
In Abhängigkeit von der Anzahl der Assoziationskanten werden ggf. Ränge in weitere Unterränge aufgeteilt und zwischen den existierenden Rängen eingefügt (siehe Abbildung 4). Eine Optimierung eines Unterrangs bezüglich der Kantenkreuzungen kann z.B. durch einen matrixbasierten Algorithmus, der durch st-Nummerierung und Strafgewichte gesteuert wird, erreicht werden.

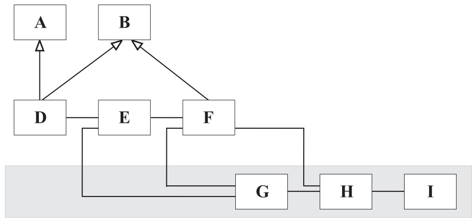

Abbildung 4. Aufspalten eines Rangs in einen Unterrang (grau hinterlegt) und Optimierung der Anzahl der Kantenkreuzungen.

Für die Koordinatenberechnung schlägt Seemann ein geteiltes Layout vor: Die Assoziationskanten werden orthogonal dargestellt, die Vererbungskanten direkt gezeichnet. Assoziationskanten verlassen Knoten nur an den vertikalen Knotenseiten, Vererbungskanten nur an den horizontalen Seiten. Das Lesen der Diagramme wird durch die genannten Darstellungstechniken für Kanten erheblich vereinfacht. Die Abbildungen 2 mit 5 entsprechen diesem Vorschlag.
Die Koordinatenberechnung garantiert zunächst, dass Kanten, die über mehrere Ränge geführt werden müssen, in Kantenketten aufgebrochen werden. Anschließend werden die Platzbedürfnisse der Knoten und Kanten berechnet. Unter Verwendung der Knotensortierung, die sich aus der Rangzuweisung und der Kantenkreuzungsminimierung ergibt, lassen sich initiale Koordinaten für Knoten und Kanten festlegen. Durch iterative Anwendung mehrerer Einzelheuristiken (siehe auch [3]) ergibt sich eine geeignete Koordinatenzuweisung: Ausrichten der Knoten unter Verwendung einer prioritätsgesteuerten Median-Heuristik, alternativ durch einen angepassten spring-embedder Algorithmus, Positionieren der virtuellen Knoten (in Bezug auf die benachbarten Knoten und Kantenketten), Platzieren von Knoten unter Berücksichtigung von Sohn- und Elternknoten, usw. Diese Heuristiken werden angewendet bis das Qualitätsmerkmal, eine gewichtete Summe der Kantenlängen, nicht mehr verbessert werden kann. Dabei werden Verschlechterungen in gewissen Grenzen toleriert. Abschließend werden besondere Korrekturen am Resultat vorgenommen, z.B. Abstände benachbarter Knoten bzw. Ränge minimiert und besondere Kantenkonstellationen bereinigt.
Durch die Verschiebung mehrerer Knoten in Form von Knotenbereichen unter Beachtung des Layouts der Vererbungskanten, die i.d.R. mit höherer Priorität

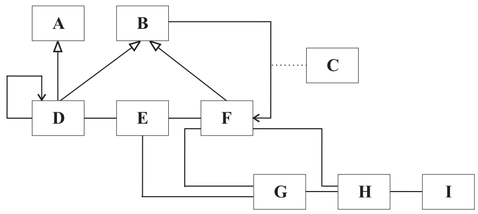

Abbildung 5. Endergebnis für das Beispieldiagramm.

bearbeitet werden, lässt sich für die meisten Assoziationskanten ein orthogonales Layout erreichen. Innerhalb eines Rangs wird ein modifiziertes Manhattan-Layout (einfacher gridline-Algorithmus) eingesetzt. Kanten(ketten), die zu diesem Zeitpunkt nicht orthogonal dargestellt werden können, werden erneut aufgebrochen, und zwischen den Rängen geführt. Die Kreuzungsminimierung dieser Kanten wird bisher heuristisch durch Sortieralgorithmen angenähert, ist aber noch nicht optimal.

3 Das Framework SugiBib

SugiBib wurde in reinem JAVA (JDK 1.2.2) implementiert und garantiert somit Plattformunabhängigkeit. Um eine möglichst frei einsetzbare Grundlage für weitere, zukünftige Anwendungen legen zu können, orientiert sich der Entwurf am Baukastenprinzip. Eine Stufe sei begrifflich zunächst an eine Stufe des Sugiyama-Algorithus angelehnt, kann aber prinzipiell die Implementierung eines Teils eines beliebigen anderen Graph-Algorithmus sein. Daraus ergeben sich fünf Grundprinzipien, die die Struktur des Frameworks im Wesentlichen festlegen:

1. Das Framework implementiert die einzelnen Stufen des Sugiyama-Algorithmus in getrennten Paketen. Zusätzliche Stufen, die sich durch eine Modifikation des Sugiyama-Algorithmus oder Erweiterung durch andere Algorithmen ergeben, werden in geeigneten Paketen realisiert.
2. Das Paket `sugi.admin` ist Grundlage aller Implementierungen einer Stufe. Es enthält z.B. die Basisimplementierung eines Graphen, eines Knotens und einer Kante, die jeweils auf geeigneten Interfaces beruhen. Weitere Klassen, wie Matrizen und Sortieralgorithmen erleichtern und optimieren die Implementierung. Die Graph-Klasse(n) einer Stufe besitzt eine implizite Transformationssemantik: Durch Aufruf des Konstruktors einer Graph-Klasse mit dem Eingabegraphen der jeweiligen Stufe, wird implizit das Resultat der Stufe berechnet.
3. Die Stufen sollen prinzipiell beliebig kombinierbar sein, wobei sich eine sinnvolle Reihenfolge hier durch den Seemann-Algorithmus ergibt. Neue Stufen sollen nahtlos zwischen bereits bestehenden Stufen eingefügt werden können. Die temporären Informationen, die zur Berechnung des Ergebnisses einer

Stufe notwendig sind, sollen in die folgende Stufe nur in dem Maße übernommen werden, in dem die Informationen tatsächlich benötigt werden. Die letzte Stufe, die das Resultat beschreibt, blendet sämtliche, für die Darstellung des Resultats unnötigen Informationen aus. Zudem kann jede Stufe ihre eigenen Graph-, Knoten- und Kantenklassen, ausgehend von den Basisklassen in `sugi.admin`, spezialisieren. Um die Informationsweitergabe und die Erzeugung der korrekten Instanzen in den unterschiedlichen Stufen zu ermöglichen, wird ein erweiterbarer Konversionsalgorithmus bereits im Basispaket `sugi.admin` implementiert. Der Effizienzverlust, der sich durch die Konversion eines Eingabegraphen in einer Stufe ergibt, wird durch die einfache Erweiterbarkeit, die konzeptionelle Trennung der einzelnen Stufen und durch die, in allen Stufen geltende Trennung der Eingabe von der Ausgabe aufgehoben. Die Anwendung der einzelnen Stufen erfolgt durch sukzessive Instanziierung der Graph-Klassen der geeigneten Stufen in der gewünschten Reihenfolge. Diese Reihenfolge wird in einer Algorithmus-Klasse festgelegt und bleibt dem Endanwender i.d.R. verborgen.

4. Da Graphen, Knoten und Kanten unabhängig vom Layout-Prozess, in Abhängigkeit der jeweiligen Anwendung des Frameworks, eigene Informationen tragen können (im Fall von UML-Klassendiagrammen würde diese zusätzliche Information z.B. in einer Knotenklasse die graphische Darstellung einer UML-Klasse beschreiben) und Eingabeobjekte in einen konkreten Algorithmus von den Ausgabeobjekten unabhängig sind, ist es notwendig, diese Information in Form eigener Klassen, den Informationsklassen, auszulagern. Graph-, Knoten- und Kantenklassen tragen prinzipiell eine eigene Informationsklasse, die das entsprechende Interface aus `sugi.admin` implementiert. Eine Informationsklasse besitzt z.B. die Fähigkeiten, ihre graphischen Ausmaße zu berechnen, sich graphisch darzustellen oder sich in einen Datenstrom zu speichern. Zudem können applikationsabhängige Daten in Informationsobjekten abgelegt werden, die während des Layout-Prozesses invariant bleiben.
5. Die Erweiterung des Frameworks erfolgt neben dem Baukastenprinzip durch hot spots wie z.B. Fabrikmethoden, Schablonenmethoden, geeignete Datenklassen und Strategieklassen (siehe auch [2, 4]).

SugiBib besteht aus einer Menge von Kernpaketen, die die allgemeine Implementierung des Sugiama-Algorithmus enthalten. Andere Layout-Algorithmen können dem Framework durch zusätzliche Pakete problemlos hinzugefügt werden. Zudem sind weitere Vorverarbeitungsschritte, die spezielle Sichten eines Eingabegraphen ausfiltern oder Faltungen berechnen, denkbar. Dazu stellt das Framework einen allgemeinen Anfragemechanismus an Knoten und Kanten zu Verfügung, um beliebige Selektionen einfach berechnen zu können.
Die Darstellung von UML-Klassendiagrammen stellt eine Ausprägung des Basis-Frameworks dar, und wurde im Paket `sugi.uml.classDiagrams` implementiert. Der Seemann-Algorithmus ergibt sich durch das Zusammenspiel der folgenden Pakete des Frameworks (siehe auch Abbildung 6):

1. `sugi.preparation` enthält diverse Vorbereitungsschritte, wie z.B. Entfernung von Zyklen, Falten spezifizierter Kanten oder Knoten (graph folding).
2. `sugi.ranking` beinhaltet die Rangzuweisung durch den Netzwerk-Simplex-Algorithmus.
3. `sugi.ordering` approximiert die Minimierung der Kantenkreuzung zwischen benachbarten Rängen nach der Median- bzw. der Barycenter-Methode.
4. In `sugi.uml.classDiagrams.incrementalExtension` wurde die zusätzliche Stufe, die die inkrementelle Layout-Erweiterung nach Seemann durchführt, implementiert.
5. `sugi.coordinates` ermöglicht die Berechnung der Koordinaten von Knoten und Kanten. Initiale Koordinatenzuweisung ist durch die Strategien „Gansner“ (prioritätsgesteuerte Medianpositionierung) oder „spring embedder“ möglich.
6. `sugi.result` beschreibt das Resultat des Layout-Prozesses und blendet Layout-Informationen, die für das Zeichnen des Resultats unwichtig sind, aus.

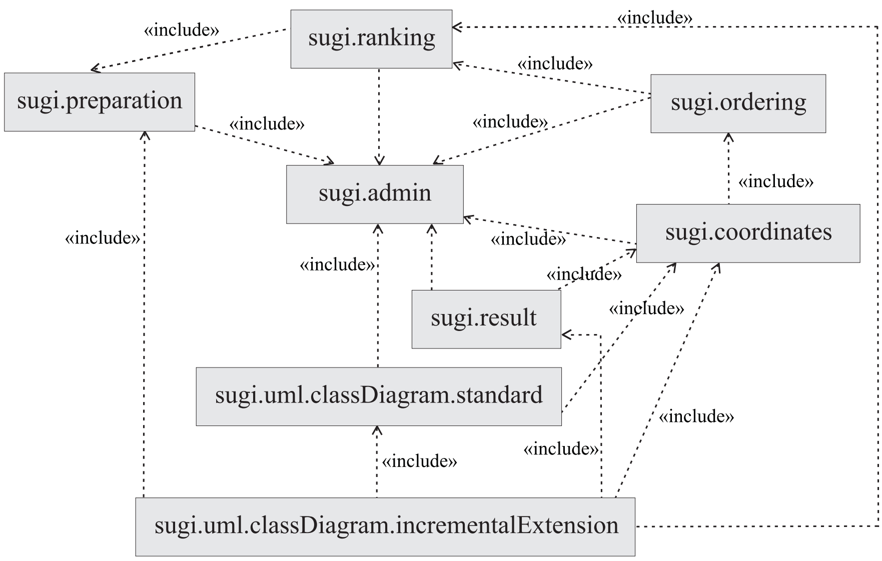

Abbildung 6. Abhängigkeiten der wichtigsten Pakete.

Implementiert wird der Seemann-Algorithmus in der Klasse `SeemannAlgorithmus`, die die Reihenfolge der auszuführenden Schritte festlegt. Konkrete Layout-Algorithmen können, insbesondere zur Laufzeit, konfiguriert oder rekursiv auf sich selbst angewendet werden. Das folgende Fragment zeigt einen Ausschnitt aus der Implementierung der Klasse `SeemannAlgorithmus`; es wird das Layoutergebnis des Eingabegraphen `input` berechnet:

```
CheckVirtualRootGraph vrgraph = new CheckVirtualRootGraph(input);
RankingGraph rgraph = new RankingGraph(vrgraph);
OrderingGraph ograph = new BarycentricOrderingGraph(rgraph,24,10,10);
IncrementalExtensionGraph igraph = new IncrementalExtensionGraph(ograph);
ICoordinatesGraph cgraph = new CoordinatesGraph(igraph);
return new ResultGraph(cgraph);
```

Die unten angegebenen Pakete wurden zur Unterstützung und Gliederung, insbesondere aber zur Ausprägung von SugiBib zum Layout von UML-Klassendiagrammen implementiert:

- `sugi.admin` ist das Paket mit den grundlegenden Klassen.
- `sugi.uml.classDiagrams.standard` stellt die Ausprägung des Frameworks für UML-Klassendiagramme dar. Die bisher einzige Eingabesprache für Klassendiagramme wurde in Form einer Erweiterung von UMLscript [7] durch einen generierten Parser bereitgestellt. Um die Informationen aus dem Parser in das Objektmodell von SugiBib zu übertragen, wurde eine Meta-Klassenhierarchie entworfen und nach Vorbild des Bridge-Musters an die Klassen im Paket `standard` angebunden.

Durch Variation der Darstellungsweise ergibt sich ein Layout, das eine entscheidende Verbesserung des Höhen-Seiten-Verhältnisses (aspect ratio) ermöglicht. Vielfach entstehen, z.B. durch die Beschreibung spezialisierter Klassen in Klassendiagrammen, breite Unterbäume, die durch keine oder nur wenige Assoziationskanten mit den restlichen Klassen verbunden sind. Diese Klassen kann man mit horizontalem oder vertikalem Versatz darstellen. Eine vertikale Ausrichtung verbessert das Höhen-Seiten-Verhältnis erheblich (siehe Abbildung 7). Um dies zu ermöglichen, werden innerhalb der inkrementellen Erweiterung Kandidaten ausgewählt, aus dem Graphen entfernt und in den Eltern-Knoten eingefügt. Knoten werden in SugiBib durch eine Aufteilung in Innenbereich (eigentlicher, sichtbarer Knoten) und Außenbereich (zusätzlich benötigter Platz) realisiert. Da die grafischen und textuellen Kantenattribute, dem Außenbereich zugeordnet werden, lassen sich die Kantenattribute überschneidungsfrei darstellen. Im Fall der Verbesserung des Höhen-Seiten-Verhältnisses liegt es nahe, den Außenbereich in geeigneter Weise zu vergrößern, um das gewünschte Unterlayout darzustellen. Die Technik des Außenbereichs kann insbesondere im Fall reflexiver Assoziationen eingesetzt werden. Die zyklischen Assoziationen, die, wie oben erwähnt, im Vorbereitungsschritt entfernt und in den zugeordneten Knoten transferiert werden, können als Kanten in der Knoteninformationsklasse gespeichert werden und, innerhalb des Außenbereiches eines Knotens, durch einen geeigneten Algorithmus fest einem Unterlayout zugeordnet werden. Analog kann man bei der Darstellung von Schablonenvereinbarungen (Templates) vorgehen.
Im Rahmen der Anpassungen und Erweiterungen der Urversion zeigte sich auch, wie nützlich die Eigenschaft, unregelmässige Knotenformen darzustellen, ist. Der Außen- und Innenbereich eines Knotens ist grundsätzlich rechteckig; die Darstellung innerhalb des Innenbereichs erfolgt durch entsprechende Methoden des Informationsobjekts des Knotens.

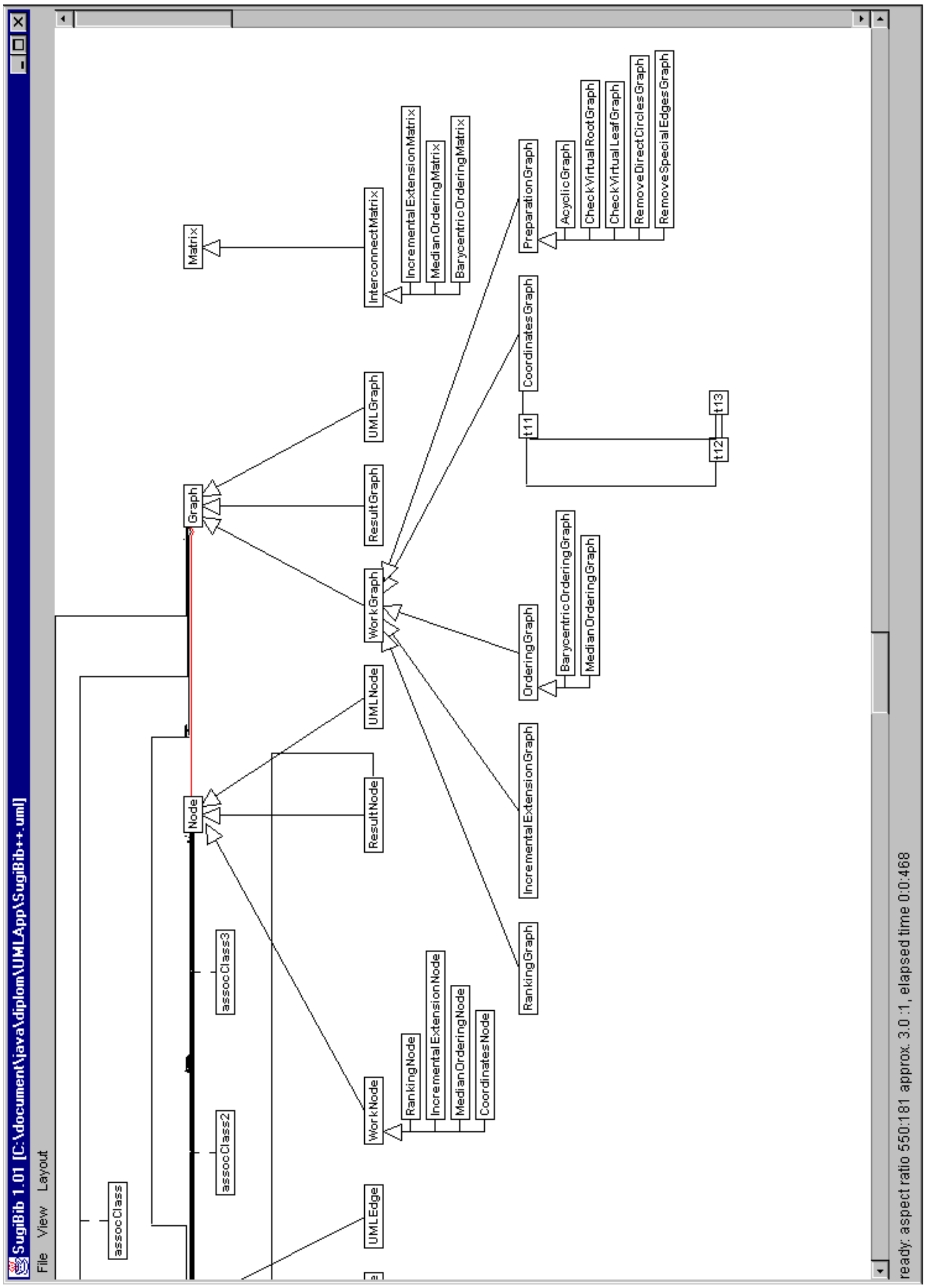

Abbildung 7. Screenshot der Beispielapplikation zur UML-Ausprägung von SugiBib. Dargestellt ist der Ausschnitt eines Teils der Klassenhierarchie von SugiBib und einiger zusätzlicher Diagrammelemente, die zu Testzwecken eingefügt wurden. Am Knoten `WorkNode` sieht man das Knoten-Unterlayout zur Verbesserung des Höhen-Seiten-Verhältnisses.

Dadurch können problemlos z.B. die Zentralknoten n-ärer Assoziationen (Raute) und Entwurfsmuster (unterbrochen gezeichnetes Oval) dargestellt werden. Durch die Kommunikation von Informationsobjekt und den Kanten, die dem Knoten zugeordnet sind, ist eine exakte Positionsbestimmung der Kantenkoordinaten in Abhängigkeit der Knotenform gewährleistet.
Weitere Probleme entstehen durch die Möglichkeit, Assoziationsklassen und Notizen in UML-Klassendiagrammen darzustellen. Assoziationsklassen werden in SugiBib in virtuellen Knoten als darstellbare Information gespeichert. Die Layoutalgorithmen, die in SugiBib verwendet werden, erkennen virtuelle Knoten, die einen Außenbereich besitzen, und erlauben damit die Darstellung von Assoziationsklassen. Die relative Positionierung an einer Kante muss aber unter Verwendung unvollständiger Information innerhalb des Layoutprozesses prädiziert werden. Dadurch entsteht eine Unsicherheit in der Qualität des Ergebnisses, die bisher zu keinerlei praktischen Problemen führte. Die allgemeine Verwendung von Notizen an beliebigen Modellelementen stellt aber weiterhin eine noch zu lösende Herausforderung dar.
Es wurde auch das Layout von Diskriminatoren und ähnlichen textuellen Attributen an Vererbungskanten untersucht. Es stellt sich hierbei ein sehr eingeschränktes ELP (edge label placement) Problem, das durch eine geeignete Sortierung und eine angemessene Vergrößerung des Zwischenraums benachbarter Ränge zufriedenstellend gelöst werden konnte.
Zur Zeit besteht SugiBib mit seiner UML-Ausprägung aus 129 Klassen in 13 Paketen und ca. 66.000 Zeilen Quellcode bei einem Kommentaranteil von 55%.

4 Anwendungen

Die Demonstrations-Applikation besitzt die Fähigkeit, UMLscript-Dateien zu lesen, das Resultat in verschiedenen Vergrößerungsstufen darzustellen und auszudrucken. Ein interaktiver Zoom-Mechanismus, der eine gewählte Klasse in ursprünglicher Größe, benachbarte Klassen etwas verkleinert und die restlichen Klassen eines Diagramms nur durch ihre Kurzform (keine Attribute, keine Methoden) darstellt, und die Kanten zwischen der ausgewählten und den direkt benachbarten Klassen farblich hervorhebt, untermauert die nachträgliche Erweiterbarkeit der Framework-Ausprägung. Eine Option, die Vererbungskanten, die in einem spitzen Winkel gegenüber der Horizontalen dargestellt werden müssten, orthogonal entsprechend der UML-Spezifikation darstellt, rundet die Applikation ab.
Der Sprachumfang von UMLscript, der in der Implementierung von SugiBib zum Einsatz kommt, stellt eine Erweiterung von [7] dar. Die Anwendung von Eigenschaftslisten (tag-values) wurde an den meisten Sprachkonstrukten ermöglicht, die Darstellung von Entwurfsmustern (*design patterns*) und einige Details, wie z.B. Klassen-Compartments zur Unterstützung der UML Version 1.3, hinzugefügt.
Das Framework wurde bisher zur Darstellung von Klassenstrukturen im Rahmen der Übungen zur Vorlesung „Objektorientierte Programmierung“ im Win-

tersemester 1999/2000 eingesetzt. Hierbei zeigte sich eine Besonderheit von UMLscript: Bei der Darstellung der Zusammenhänge typparametrisierter Klassen in C++ (z.B. zwischen einer typparametrisierten Listenklasse und ihren Iteratoren oder bei der Visualisierung der Grundstruktur der *Standard Template Library* STL) ergeben sich Einschränkungen in der Verwendung der Modellelemente: UML verbietet die Vererbung parametrisierter Klassen unter Verweis auf die Verwendung undefinierter Elemente in der wohldefinierten Semantik von UML. UMLscript hingegen erlaubt die freie Vererbung parametrisierter Klassen. Es wurde beschlossen, diese Eigenschaft von UMLscript unter dem Aspekt der anwendungsrelevanten Darstellung von Software-Entwurfsdiagrammen beizubehalten. Hierbei zeigt sich deutlich, dass SugiBib selbst keinerlei Prüfung auf semantische Korrektheit durchführt. Die UML-Ausprägung von SugiBib verlässt sich auf die Eingabesprache und den Autor der Eingabedatei, der die semantische Korrektheit selbst, oder durch ein anderes Tool, z.B. ein CASE-Tool, garantieren muss.
Bisher existierte die UML-Ausprägung nur innerhalb der Grenzen unseres Lehrstuhls als Tool, das ständig weiterentwickelt wird. Um möglichst vielen Interessenten Zugang zur jeweils aktuellen Implementierung zu ermöglichen, gibt es seit Januar 2000 eine WWW-Schnittstelle zur UML-Ausprägung, die als heterogene Lösung unter Verwendung von CGI und Perl realisiert wurde. Ein Einsatz als Servlet oder Java Server Page (JSP) ist ebenfalls denkbar. Unter `http://www2.informatik.uni-wuerzburg.de/SugiBib` finden Interessierte eine kurze Beschreibung des Projekts, die aktuelle Dokumentation des Frameworks, die Beschreibung von UMLscript und den Verweis auf das CGI-Skript. Der Benutzer kann im Skript eine UMLscript-Datei angeben; das Ergebnis wird in Form einer EPS-Datei an die angegebene email-Adresse gesendet. Unter Verwendung eines abgeleiteten, angepassten Graphik-Kontexts, der die Zeichenanweisungen von SugiBib nicht auf dem Bildschirm darstellt, sondern in eine Datei schreibt, und eines virtuellen X-Servers zur Erzeugung des Graphikkontexts, wurde die Server-Version *UMLNet* auf einem Linux-Server in Betrieb genommen.

5 Zukünftige Entwicklungen

SugiBib und die Ausprägung zur Darstellung von UML-Klassendiagrammen stellt kein abgeschlossenes Projekt dar. Das Layout von Klassendiagrammen beinhaltet unvorhersehbare Komplikationen durch interferierende, heuristikbasierte Algorithmen, die nur durch viele Tests erkannt und beseitigt werden können. Daher ist es wünschenswert, dass neben der Anpassung an die jeweils gültige UML-Version, möglichst viele Praxistests durchgeführt werden, um ein stabiles Tool bereitstellen zu können. Erweiterte Funktionalität kann durch eine graphische Benutzeroberfläche, die das Modifizieren von Klassen- und Kanteneigenschaften interaktiv ermöglicht und durch die Integration standardisierter Datenaustauschformate wie z.B. XML bzw. UML XMI DTD oder Tuple-Attribute-Language [8] erreicht werden. Sobald eine Implementierung vorliegt, ist auch an den Einsatz der UML-Ausprägung von SugiBib als Browser-Komponente

zu denken. Große Hoffnung legen wir auf die offenen Schnittstellen der aktuellen CASE-Tools, wie MID Innovator oder Rational Rose. Eine Integration von SugiBib in ein CASE-Tool ist damit möglich.
Die Layoutfähigkeiten könnten in zukünftigen Versionen in Hinblick auf die Höhen-Seiten-Verhältnisse bestehender Druckmöglichkeiten (z.B. DIN A4) optimiert werden. Die Darstellung von Kommentaren an beliebigen Modellelementen stellt weiterhin ein Entwicklungsziel dar. Zudem wäre es außerordentlich wünschenswert, Algorithmen und Verfahren zu entwerfen, die die Unabhängigkeit des Resultats von der Reihenfolge der Klassen des Eingabediagramms garantieren. Bisher fundieren die meisten Initialisierungsroutinen auf der gegebenen Reihenfolge der Elemente des Eingabegraphen. Selbst kleine Veränderungen in der Eingabe können das Ergebnis wesentlich beeinträchtigen.

Literatur

1. H. Eichelberger: Entwicklung eines Frameworks zum automatischen Zeichnen von Software-Entwurfsdiagrammen, Diplomarbeit Universität Würzburg, 1999
2. E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*, Addison Wesley, 1996
3. E.R. Gansner, E. Koutsofious, S.C. North, K.-P. Vo: A Technique for Drawing Directed Graphs, *IEEE Transactions on Software Engineering*, SE-19(3):214-230, March 1993
4. R. Pree: *Komponentenbasierte Softwareentwicklung mit Frameworks*, dpunkt-Verlag, 1997
5. K.G. Kakoulis, I.G. Tollis: On the Edge Label Placement Problem, *Lecture Notes in Computer Science*, LNCS 1190 S. North (Editor), 241-256, 1997
6. *OMG Unified Modeling Language Specification* Version 1.3, June 1999 via `http://www.rational.com`
7. J. Seemann, J. Wolff von Gudenberg: UMLscript: A Programming Language for Object-Oriented Design, *Ein Buch über Unified Modeling Language*, M. Schader, A. Korthaus (Hrsg.), 160-170, 1998
8. J. Seemann, J. Wolff von Gudenberg: Attributierte Graphen zur Metamodellierung mit UML, Vortrag zum 7-ten GROOM-Workshop „UML-Erweiterungen (Profile) und Metamodellierung“, 5.4.2000, Koblenz
9. J. Seemann: Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams, *Lecture Notes in Computer Science*, LNCS 1353 G. DiBattista (Editor), 414-423, 1998
10. K. Sugiyama, S. Tagawa, M. Toda: Methods for Visual Understanding of Hierarchical System Structures, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109-125, February 1981

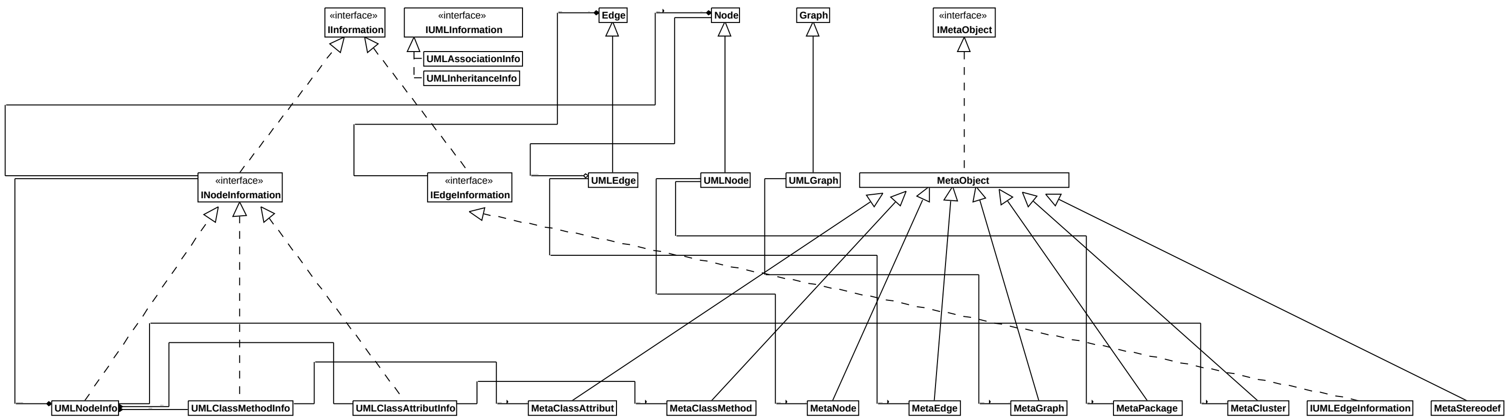

Abbildung 8. Ältere Darstellung der UML-Ausprägung von SugiBib, die orthogonales und direktes Layout der Kanten demonstriert.