Alternative Modellierungskonzepte in 3D

Daniel Vogtland

19. September 2004

Gutachter: Jens Schröder

1 Einleitung

Zum Verständnis großer Software-Systeme werden geeignete Visualisierungsmöglichkeiten benötigt. Der eigentliche Quellcode wird schnell zu umfangreich und unübersichtlich, als dass er noch als Basis für Konzeptionen dienen könnte. Visualisierungstechniken, die den strukturellen Aufbau, Abhängigkeiten und letztendlich auch das (gewünschte) Verhalten der Software möglichst intuitiv beschreiben, werden benötigt. Nur so können Entwurf und Wartung derartiger Projekte effektiv durchgeführt werden.

Als Standard zur Visualisierung von Modellen im Umfeld der Softwaretechnologie gilt die ***UML*** (Unified Modeling Language). Sie stellt u. a. standardisierte Diagrammtypen, wie etwa Klassen- und Objektdiagramme, Sequenzdiagramme und Kollaborationsdiagramme, bereit. Allerdings können UML-Diagramme mit zunehmender Komplexität des dargestellten Modells schnell unübersichtlich werden, wodurch der Einblick in Struktureigenschaften des Modells verhindert werden kann. Verbesserungen können nur erfolgen, wenn die (visuelle) Informationsdichte bei gleicher Information vergrößert wird oder die visuelle Informationsverarbeitung durch eine günstigere Visualisierungsform beschleunigt wird.

Im Folgenden sollen kurz drei mögliche dreidimensionale Visualisierungen von (Aspekten von) Softwaresystemen vorgestellt werden. Dabei wird auf die Darstellung technischer Details und Implementierungsbereiche verzichtet, sondern nur die grundsätzlichen Ideen und beispielhafte Anwendungsmöglichkeiten werden vorgestellt. Zunächst wird auf die Grundzüge von Geon-basierten Visualiserungen eingegangen. Danach werden Graph-basierten Darstellungen anhand zweier Beispielsysteme dargestellt. Abschließend wird noch kurz eine einfache dreidimensionale Erweiterung von UML vorgestellt.

2 Geon-basierte Darstellungen

Visuelle Repräsentation sollte so gestaltet sein, dass sie menschliche Erkennungsprozesse nutzt. Eine Visualisierung zu „verstehen“ bedeutet, das dargebotene Bild (die aktuelle Sicht) in Elemente zu zerlegen und diese gegebenenfalls zueinander in Beziehung zu setzen. Um mit den Elementen Informationen verknüpfen zu können, müssen diese eindeutig identifiziert bzw. klassifiziert werden. Anders ausgedrückt besteht die Aufgabe für den Betrachter darin, (bekannte) ***Objekte*** in dem Bild zu erkennen.

2.1 Menschliches Bild-Verstehen: Die Biederman Theorie

Biederman hat in seiner Arbeit [1] die Grundsteine für die Konzeption eines dreidimensionalen Diagrammtyps gelegt, der diesem Ansatz nachgeht. Seiner Meinung nach sind Farbe, Intensität und Textur eines Objektes eher sekundäre Attribute bezüglich der Identifikation. Auschlaggebend für die Identifikation ist vor allem die Gestalt bzw. Form, dies ist also ein primäres Attribut. Objekte sollten möglichst unabhängig vom Standort des Beobachters sein. Also sollten sie durch relative Koordinaten und Größenverhältnisse definiert sein, statt durch absolute Werte. Probleme bei fehlender Informationen (z.B. wenn das Objekt teilweise verdeckt ist) werden durch Inferenzmechanismen gelöst. Dies bedeutet, dass das zu analysierende Element dem Objekt mit der größten Übereinstimmung zugeordnet wird. Zusätzliches Wissen und wahrnehmungspsychologische Gesetze fließen in den Erkennungsprozess mit ein. Auf diese Weise ist auch des Abstrahieren von Karrikaturen o. ä. möglich.

Biederman gibt schematisch einen klar strukturierten Ablauf für die Erkennung eines Objekts an. Zunächst werden Kanten extrahiert, beispielsweise durch Unterscheidung verschiedener Texturen. Falls genügend Informationen vorhanden sind, werden konkave Regionen ermittelt. Diesen werden bestimmten Komponenten (Teilobjekte) zugeordnet. Sind nicht genügend Informationen vorhanden, wird stattdessen auf so genannte ***nonaccidental properties*** geprüft. Es handelt sich dabei um fünf Eigenschaften, die im zweidimensionalen Bild wahrgenommen in die dreidimensionale Vorstellung übertragen werden. So führt z. B. die Erkennung einer geraden Linie zu der Annahme, dass die entsprechende/n Kante/n des erzeugenden dreidimensionalen Objektes ebenfalls einen geraden Verlauf hat/haben. Diese fünf Eigenschaften zeichnen sich durch relative Unempfindlichkeit gegenüber Rausch- und (leichten) Verzerrungseffekten so wie der Perspektive des Betrachters aus. Vielleicht können nun direkt Komponenten entdeckt werden. Es besteht aber auch die Möglichkeit, dass sich neue Kanteninformationen ergeben (z. B. beim Erkennen von Symmetrie), dann wird zum Stadium der Kantenextraktion zurückgekehrt. Sind schließlich Komponenten gefunden, werden sie und ihre Kombinationen mit bekannten Objekten abgeglichen. Das Objekt mit grösster Übereinstimmung (eventuell unter Zuhilfenahme von zusätzlichem Wissen) wird identifiziert.

Diese Komponenten wurden von Biederman als ***Geonen*** bezeichnet,- einfache dreidimensionale Gebilde aus denen jedes mögliche Objekt zusammengesetzt werden kann. Aus-

gehend von primitiven Grundkörpern entwickelte Biederman nach diesem Ansatz durch unterschiedliche Ausprägungen der ***nonaccidental properties*** einen Geonensatz von 36

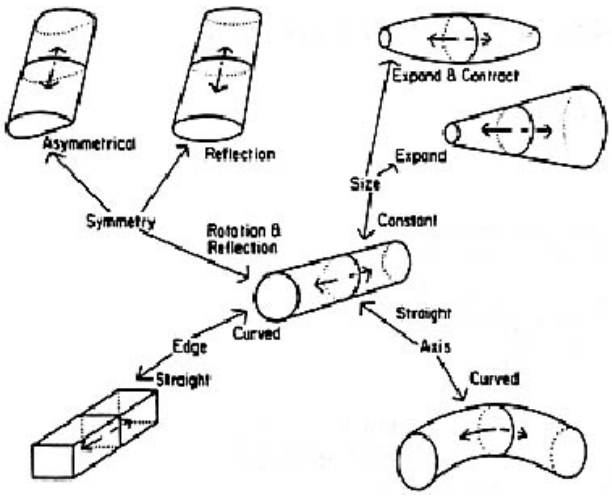

Abbildung 1: Entwicklung von Geonen anhand *nonaccidental properties*, Quelle: [1]

Geonen (Abbildung 1).

Jedes Objekt wird bei der Verarbeitung lückenlos in Komponenten aus diesem Satz zerlegt (Abbildung 2). Allerdings wird ein Objekt nicht nur durch die enthaltenen Geonen

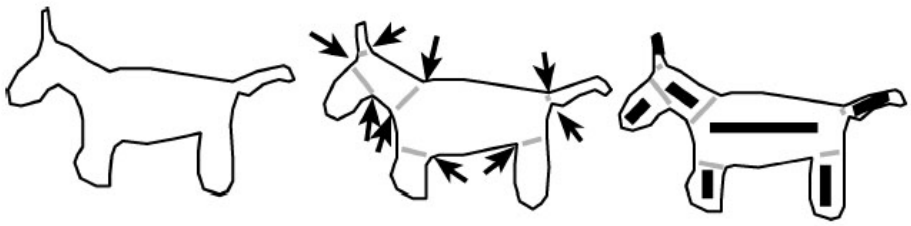

Abbildung 2: Objektzerlegung in Komponenten (Geonen), Quelle: [2]

bestimmt, auch die Anordnung und Größenverhältnisse sind entscheident. Sekundäre Attribute wie Textur oder Farbe werden zur Informationserweiterung genutzt, beispielsweise zur Unterscheidung von zwei Objekten mit gleicher Form.

2.2 Geonen und UML

In ihrer Arbeit [3] griffen die Autoren die Theorie von Biederman auf. Ihre Zielsetzung war eine verbesserte Visualisierung von Softwaresystemen, die jedoch genauso mächtig wie UML-Diagramme sein sollte. „Verbesserung“ bedeutet hier vor allem besseres intuitives Verständnis. Ausgangspunkt waren die bestehenden UML Notationen, wie sie in Klassen-

und Objektdiagrammen vorkommen.

Basierend auf Biedermans Arbeit mit einem zusätzlichen Einbettungs- bzw. enthalten-sein-Aspekt (containment) formulierten die Autoren folgende Darstellungs-Regeln:

1. Farbe und Textur (Color and Texture) sind Oberflächeneigenschaften von Geonen, die nur eine sekundäre Rolle bei der Objektklassifikation spielen. Sie können beim kognitiven Prozeß eine Hilfe darstellen, spielen aber nur eine untergeordnete Rolle bei der ersten und automatisiertesten Erkennungsphase.

2. Vertikalität (Verticality): Geon A kann ÜBER, UNTER oder NEBEN Geon B sein.

3. Zentrierung (Centering): Objekte können zentralisiert oder dezentralisiert sein. Beispielsweise befinden sich menschliche Beine links und rechts an der Unterseite des Torsos (dezentralisiert), menschliche Arme sind links und rechts seitlich oben am Torso angeordnet (zentralisiert).

4. Verbindung relativ zu Verlängerung (Connection relative to elongation): Die meisten Geonen werden verlängert. Ob sich die Verbindung zu einem anderen Geon an einer langen oder kurzen Fassette befindet, hat wichtige Wahrnehmungs-relevante Bedeutung. Als Beispiel führen die Autoren die Unterscheidung von Menschen und vierbeinigen Tieren an.

5. Relative Größe (Relative Size): Geon A kann größer, gleich groß oder kleiner als Geon B sein.

6. Einbettung (Containment): Im Kontext eines Softwaresystems müssen auch Objekte, die von größeren Komponenten eingeschlossen sind, identifiziert werden können. Diese Enthalten-sein Beziehung ist hierarchisch. Solche Mechanismen benötigen bei der Darstellung die Nutzung von Transparenz.

Zunächst versuchten die Autoren verschiedene Konzepte der Modellierung darzustellen: Generalisierung (A ist-ein B), Abhängigkeit (A ist abhängig von B), Beziehungsstärke (manche Beziehungen sind stärker als andere), Multiziplitäten von Beziehungen / Ordinalität (ein A kann zu vielen B in Beziehung stehen) und Aggregationen (A besitzt-ein B). Für jedes Konzept wurden alternative Visualisierungsformen entwickelt. Testpersonen sollten anschließend die besten Darstellungsformen für diese Konzepte ermitteln. Der Testpersonenkreis bestand aus zwei Gruppen: Experten (Erfahrung mit Software-Diagramm-Notationen, teilweise UML) und Laien (keine Erfahrung mit Software-Visualisierung). Abbildung 3 fasst alle ausgewählten Visualisierungen zusammen. Nach diesen Versuchsreihen formulierten Inrani, Tingley & Ware unter Einbeziehung früherer Ergebnisse und externer Arbeiten folgende Liste von Regeln zur wahrnehmungsunterstützenden Diagrammgestaltung:

- Die Hauptelemente eines Systems sollten durch Geonen dargestellt werden.

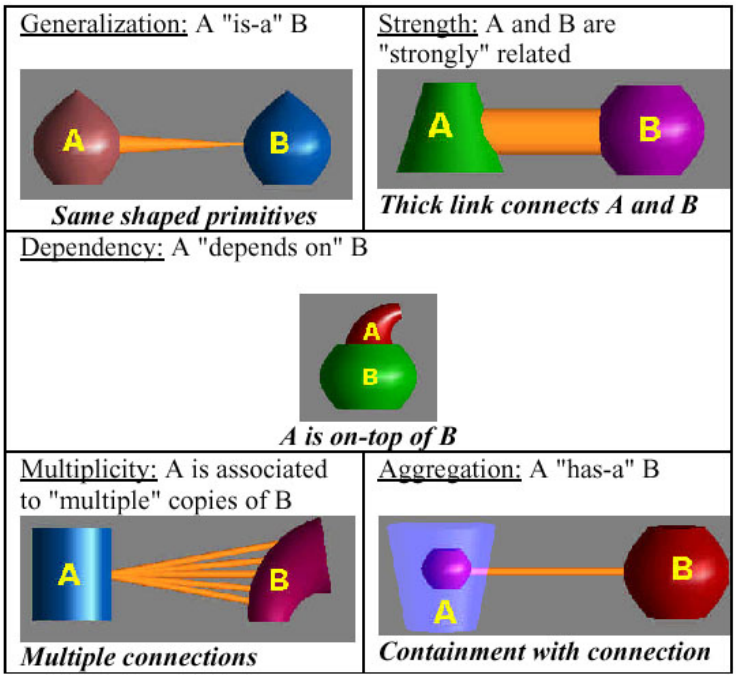

Abbildung 3: von Testpersonen bevorzugte Realisierungen, Quelle: [3]

- Verbindungen zwischen Elementen sind Verbindungen zwischen Geonen, Datenstrukturen werden durch ein Geonenskelett (Irani, Tingley & Ware bezeichnen mit Geonenskelett die Kombination von Geonen, die ein bestimmtes zusammengesetztes Element erzeugt) dargestellt.
- Untergeordnete Sub-Komponenten werden als Geonanhänge repräsentiert: kleinere Geonen werden an größeren angebracht.
- Geon Darstellungen sollten schattiert dargestellt werden, damit ihre 3D Gestalt besser zu erkennen oder überhaupt erst unterscheidbar ist (gleiche Silhouette).
- Sekundäre Attribute von Elementen und Beziehungen werden durch Farbe, Textur und Symbolen auf der Oberfläche eines Geons dargestellt.
- Alle Geonen sollten vom gewählten Beobachtungspunkt aus sichtbar sein.
- Das Geondiagramm sollte sich initial an der orthogonal zur Blickrichtung liegenden Ebene orientieren.
- Verbindungspunkte von Geonen sollten deutlich sichtbar gemacht werden.
- Ähnlichkeit, Generalisierung: Geonen mit gleicher geometrischer Komposition oder Form können Elemente gleichen Typs ausdrücken.

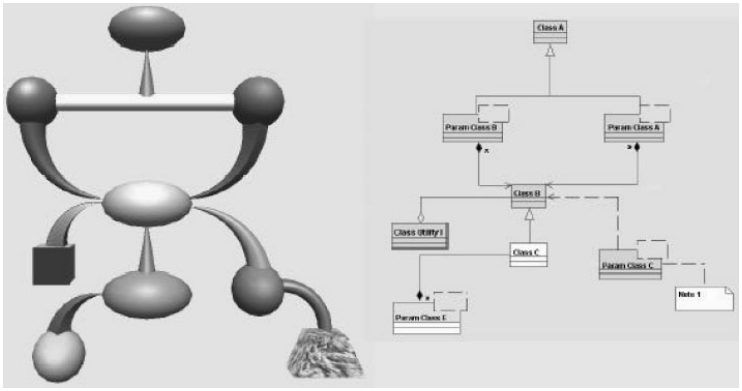

Abbildung 4: geonbasiertes Diagramm und äqivalentes UML Diagramm, Quelle: [2]

- Gravitation: Wenn Geon A sich über Geon B befindet, suggeriert dies, dass A von B unterstützt wird. Außerdem beeinflusst Gravitation den Eindruck, ob Strukturen stabil oder instabil sind.
- Einbettung zeigt, dass Geon A in Geon B enthalten ist. Syntaktisch kann dies dargestellt werden, indem eine innere Komponente an dem gleichen Geon außerhalb festgemacht wird.
- Ordinalität: Mehrfache Assoziationen zwischen zwei Elementen können am besten durch mehrfache Befestigungen visualisiert werden.
- Kräftigere Verbindungen suggerieren höhere Verbindungsstärken.
- Sequenz: Geonen, die in einer Linie angeordnet sind, werden eine Metapher für eine Operationskette oder andere lineare Strukturen.
- Symmetrie: Manche Informationsstrukturen besitzen Symmetrie und symmetrische Geon-Anordnungen sollten dazu genutzt werden, dies zu zeigen.
- Zentralisierung: Hat eine Komponente zentrale Bedeutung für eine Struktur, so kann dies durch ihre Position und Anordnungen von Wechselbeziehungen verdeutlicht werden.
- Größe: Größere Komponenten können übergeordneten Bereichen zugeordnet werden.

In [2] beschreiben Irani & Ware Versuche, welche die Vorteile der geonbasierten Darstellung bestätigen. Der verwendete Geonenensatz bestand aus 24 Elementen. Es wurden Erinnerung an Strukturen und Auffinden von Teilstrukturen getestet (Abbildung 4). Dabei wurden Geondarstellungen neben UML-Diagrammen auch mit zweidimensionalen Geon-Silhouetten Diagrammen verglichen. Dieser Ansatz lieferte somit auch Vergleichsergebnisse zwischen zweidimensionalen und dreidimensionalen Darstellungen, losgelöst von der UML-Notation (andernfalls könnte man annehmen, die UML-Notation sei einfach nur „schlecht“).

Geondiagramme führten durchweg zu den besten Ergebnissen.

In [4] wird eine in Java 3D implementierte Anwendung angesprochen. Diese unterstützt auch eine automatisierte Komposition der Elemente. Dies ist nach Dwyer eine notwendige Anforderung, da das Navigieren im dreidimensionalen Raum nicht so einfach ist wie in einem zweidimensionalen Editor, und der entsprechende Zeitaufwand bei der Modellierung ohne Automatisierung nicht vertretbar ist [5].

3 Graph-basierte Visualisierungen

Die Struktur und einzelne Momentaufnahmen eines Softwaresysteme können als ***Graph*** aufgefasst werden. Ein Graph besteht aus Knoten und (gerichteten oder ungerichteten) Verbindungen zwischen diesen Knoten. Diesen Knoten und Verbindungen (Kanten) können Eigenschaften zugeordnet werden, wie Farbe, Form oder auch Klassenzugehörigkeit. Im Kontext eines Softwaresystems können mögliche Knotentypen beispielsweise Klassen und Objekte, eventuell aber auch Methoden und Ressourcen sein. Kantentypen könnten „ist-ein“, „benutzt“, „führt aus“ oder „Instanz von“ sein.

Das Hauptziel besteht darin, die Darstellung so zu optimieren, dass sie für den Betrachter wichtige Beziehungen und strukturelle Merkmale in den Daten aufdeckt. Dies kann beispielsweise das ***Clustering*** beinhalten. Cluster sind hier Partitionen von Knoten, die z. B. dadurch gekennzeichnet sind, dass innerhalb eines Clusters alle Knoten stark miteinander (durch Kanten) verbunden sind, aber nur wenige Verbindungen hinausführen. Hierbei müssen Möglichkeiten gefunden werden, den Berechnungsaufwand zu reduzieren.

3.1 Walrus: Allgemeine Graphdarstellungen

Walrus ist ein in Java (mit Java 3D) programmiertes lauffähiges Visualisierungstool für Graphen. Es ist nicht auf Softwarevisualisierung spezialisiert, sondern dient der Darstellung von allgemeinen Graphen. Eigentlich handelt es sich um gerichtete Graphen, dies ist in der Visualisierung jedoch nicht erkennbar. Abbildungen 5 und 6 zeigen zwei Beispiele. Statt eines einfachen dreidimensionalen Raums wird ***hyperbolische Geometrie*** benutzt. In der hyperbolischen Geometrie entspricht ein Quadrat einer Kugel mit gleichem Mittelpunkt (vgl. [7, 8]). Zusätzlich wird ein Fischauge-Effekt (vgl. [9]) realisiert. Er bewirkt, dass die Detailstufe mit wachsendem Abstand vom Fokusmittelpunkt verringert wird. Der Nutzer kann die Darstellung interaktiv beeinflussen (z. B. Navigation und Anzeigeoptionen).

Neben der Definition von Knoten und gerichteten Kanten können in Walrus auch Attribute für diese individuell definiert werden, welche durch Walrus auslesbar sind. Diese Attribute haben in der Regel keine semantische Bedeutung, allerdings können sie zur Farbgebung benutzt werden. Auf diese Weise können Softwarevisualisierungsaspekte (welche ja beispielsweise in unterschiedlichen Kantentypen bestehen können; besitzt-ein vs. ist-ein)

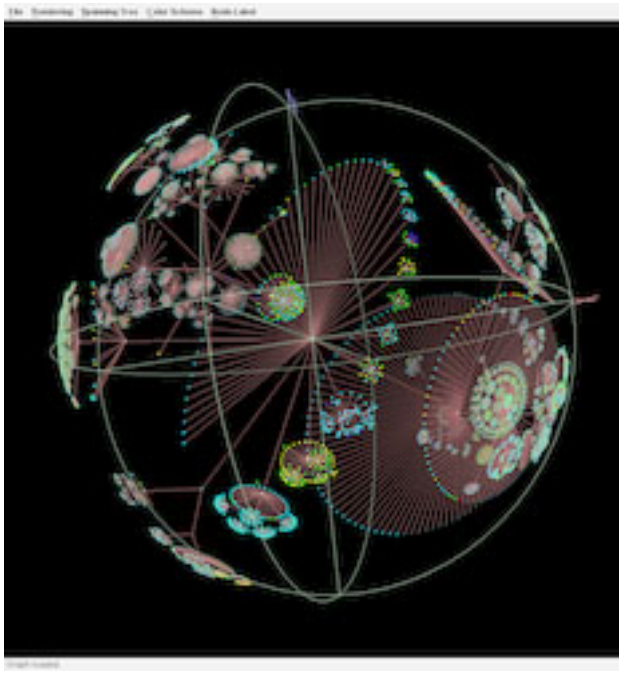

Abbildung 5: Walrus-Visualisierung einer Verzeichnisstruktur, Quelle: [6]

verwirklicht werden (Abbildung 7).

Um den Berechnungsaufwand zu reduzieren, wird ein ***Spannbaum*** (spanning tree) über die Eingabeknoten vorausgesetzt. Visualisierungen von Baumstrukturen haben erheblich geringere Komplexität [7]. Ausgelassende Kanten des Originalgraphen werden anschließend einfach wieder eingefügt (hier liegt auch der Nachteil dieser Methode,- Fehleranfälligkeit). Weitere Informationen zu Walrus sind auf der CAIDA-Website [6] zu finden.

3.2 CrocoCosmos: Software-spezifische Graphen

Bei ***CrocoCosmos*** handelt es sich um eine Komponente eines experimentellen Tools zur Analyse eines Softwaresystems. Dieser Abschnitt stützt sich bei der Beschreibung des Systems auf den Artikel [10].
CrocoCosmos dient der Visualisierung von Graphen, die zuvor aus Softwarestrukturen ermittelt wurden. Die Eingabe ist der gesamte Quellcode, der von der Applikation analysiert wird. Das Ergebnis ist ein hierarchischer Graph, der verschiedene Detailstufen ermöglicht. Die Programmentitäten wie Methoden, Attribute, Klassen, Dateien oder Subsysteme stellen Knoten dar. Die Containment-Hierarchie (z. B. Package - Klasse) wird durch die Graphstruktur behandelt, welche zusammenfassbare Untergraphen ermöglicht. Weitere Relationen wie „benutzt“, „ruft auf“ oder „ist-ein“ werden durch gerichtete Kanten erfasst. In den Knoten werden zusätzliche Informationen abgespeichert. Diese beruhen auf Metriken, welche Eigenschaften des Softwaresystems auf Zahlenwerte abbilden. Ein Beispiel für ei-

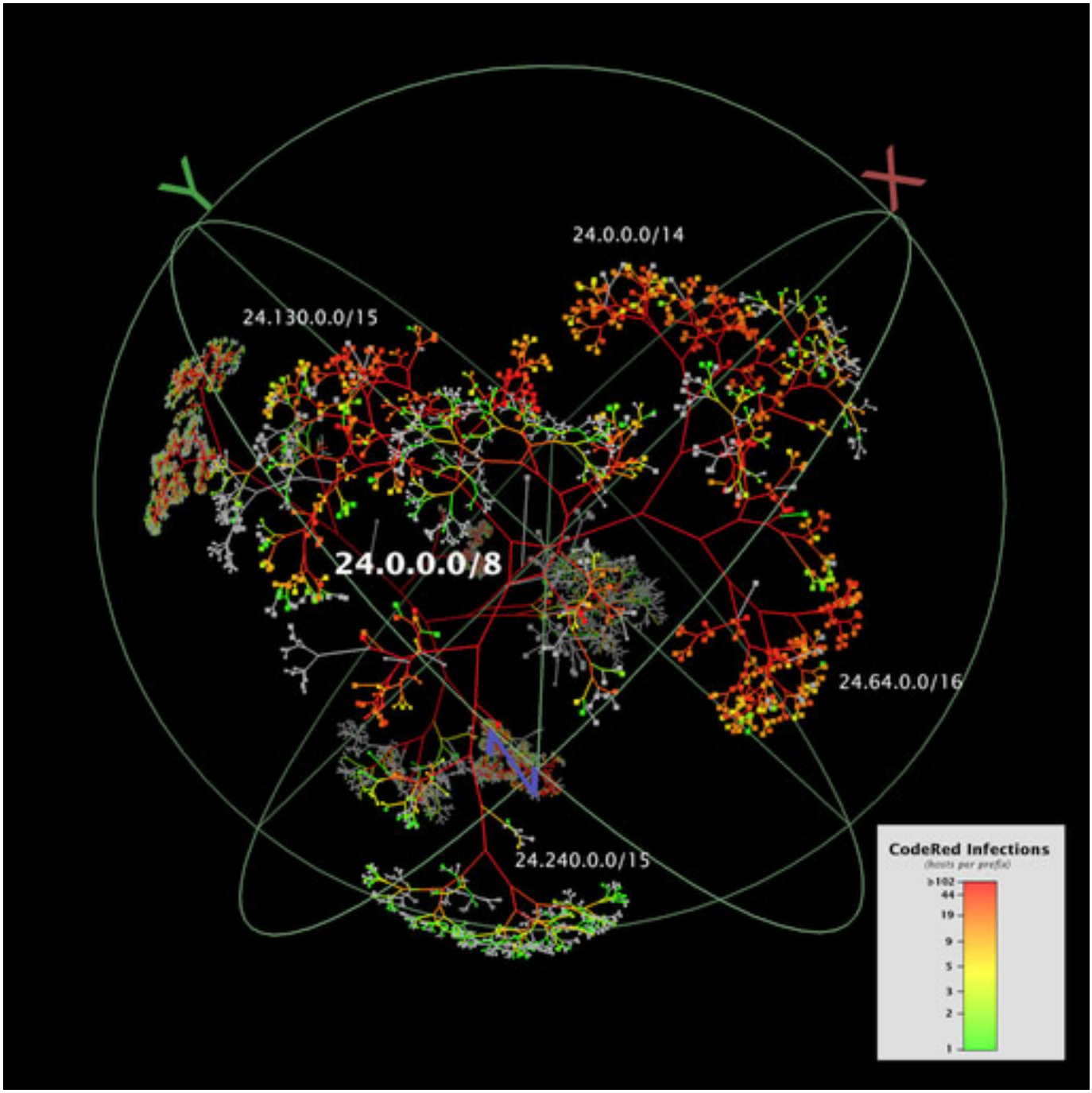

Abbildung 6: Walrus-Visualisierung einer Netzinfektion durch den Code-Red Wurm, Quelle: [6]

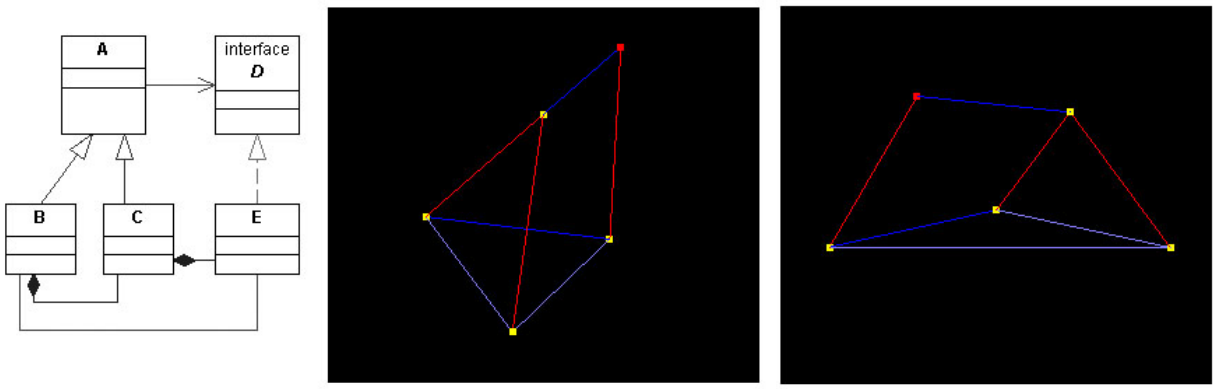

Abbildung 7: einfaches UML-Klassendiagramm (links) und zwei Walrus-Visualisierungen mit unterschiedlichen Spannbäumen.

ne solche Eigenschaft besteht in der Anzahl von ein- und ausgehenden Kanten bezüglich der „besitzt-ein“-Beziehung. Für jede Hierarchhieebene sind dabei individuelle Metriken definierbar. Die in den Knoten gespeicherten Metrikwerte werden bei der Visualisierung benutzt.

Knoten werden als Volumina (z. B. Quader oder Kugeln), Kanten als farbige Linien mit Farbverlauf entsprechend der Start- und Zielknoten dargestellt. Formzuordnungen (z. B. Kugeln für Klassen) und Knotenfärbungen (es können alle Subsysteme eines Systems die gleiche Farbe haben) sind durch den Nutzer beeinflussbar. Weitere visuelle Eigenschaften können mit Hilfe entsprechender Metriken von den Merkmalen der zu visualisierenden Softwareentität abhängig gemacht werden, beispielsweise kann der Durchmesser einer Klassen-Kugel durch die Anzahl der zu dieser Klasse gehörenden Methoden bestimmt sein. Der Nutzer hat weiterhin die Möglichkeit, Hierarchieebenen ein- oder auszublenden und frei im dreidimensionalen Raum zu navigieren.

Bei der Visualisierung eines solchen Graphen durch CrocoCosmos wird auf das Konzept der Energiemodelle zurück gegeriffen. Es handelt sich um Bewertungsfunktionen für Graphvisualisierungen, was zu einem Minimierungsproblem führt. In CrocoCosmos wird das ***LinLog Energy Model*** benutzt [10]. Auf dieses wird näher in [11] eingegangen (weitere Energiemodelle werden beipielsweise in [5, 12] beschrieben).

Ein Beispiel für eine schematische CrocoCosmos-Visualisierung ist in Abbildung 8 dargestellt. Abbildung 9 veranschaulicht die Möglichkeit der Hierarchieebenennutzung. Weitere Informationen zu CrocoCosmos sind auf [13] zu finden.

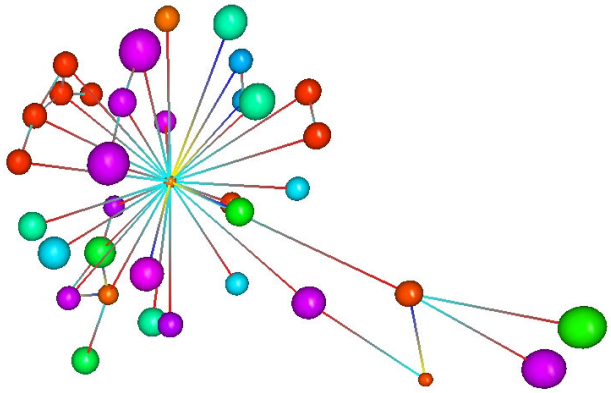

Abbildung 8: eine einfache CorocoCosmos-Visualisierung, Quelle: [10]

4 Animiertes 3D UML

Die UML hat sich als Visualisierungsnotation fest etabliert. Die vielleicht naheliegenste Idee, Softwarevisualisierungen zu verbessern, könnte also im Gegensatz zum Geonenansatz auch darin bestehen, UML „einfach“ um eine dritte Dimension zu erweitern (3D UML). Die zweidimensionalen Notationenselemente werden durch naheliegenden Transformationen zu dreidimensionalen Objekten: Assoziationslinien zu Röhren, Klassenrechtecke zu flachen Boxen usw. (vgl. Abbildung 10).

Gogolla & Radfelder weisen in ihrer Arbeit [14] auf einen entscheidenen Vorteil hin, der durch die zusätzliche Dimension in Verbindung mit Nutzerinteraktion entsteht. So werden 2D UML-Diagramme schnell sehr komplex. Interssieren nur die Beziehungen zwischen zwei an entgegengesetzten Rändern gelegenen Objekten, so sind diese meist nicht mehr zu erfassen. Die dritte Dimension kann jedoch diesem Problem entgegenwirken. Dazu werden die entsprechenden Elemente in den Vordergrund geholt. Die gewünschten Beziehungen sind vergrößert und überlagern unwichtige Diagrammelemente, welche ihrerseits unscheinbar und nicht störend im Hintergrund verbleiben.

Doch auch die zusätzliche Dimension allein ist gewinnbringend. Wird eine Dimension zur Darstellung von dynamischen Aspekten eines Softwaresystems benötigt, verbleibt beim 2D UML nur eine Dimension für verschiede Elemente, wie beispielsweise interagierende Objekte. Beim 3D UML können die Elemente dann statt nur nebeneinander auch hintereinander angeordnet sein. Ein gutes Beispiel sind Sequenzdiagramme (Abbildung 11).

Eine weitere Verbesserung der Visualisieriung besteht in der Nutzung von Animation. So können einerseits platzsparend dynamische Aspekte visualisiert werden (Abbildung 12), anderseits wird in [14] aber auch auf die Möglichkeit verwiesen, dynamische Aspekte vor statischen ablaufen zu lassen. So könnte beispielsweise im Hintergrund ein Klassendiagramm abgebildet sein, aus dem heraus im aktuellen Programmablauf neu instanziierte

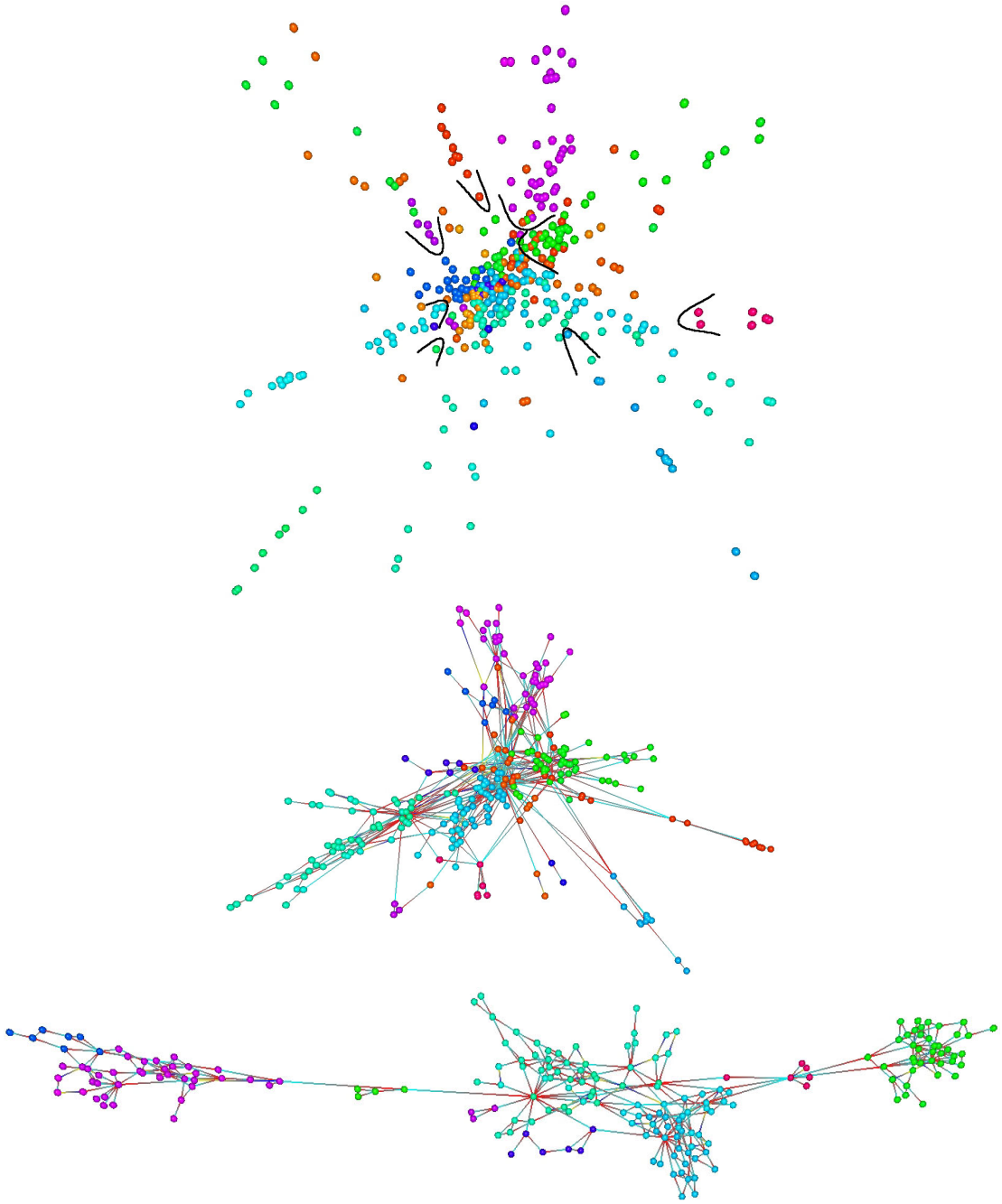

Abbildung 9: Repräsentation einer Programmlogik mit CrocoCosmos, suksessives Entfernen von unteren (detaillierteren) Schichten von oben nach unten, Quelle: [10]

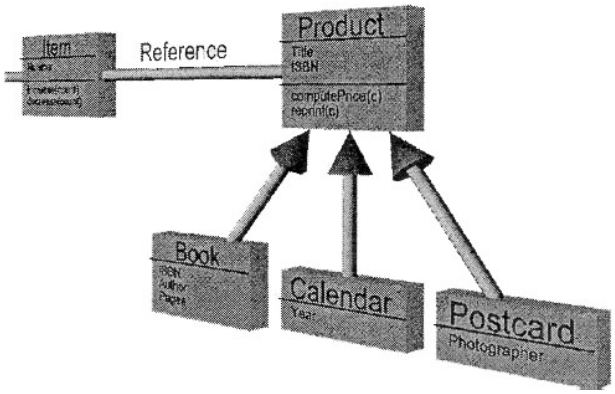

Abbildung 10: ein einfaches dreidimensionales UML-Diagramm, Quelle: [14]

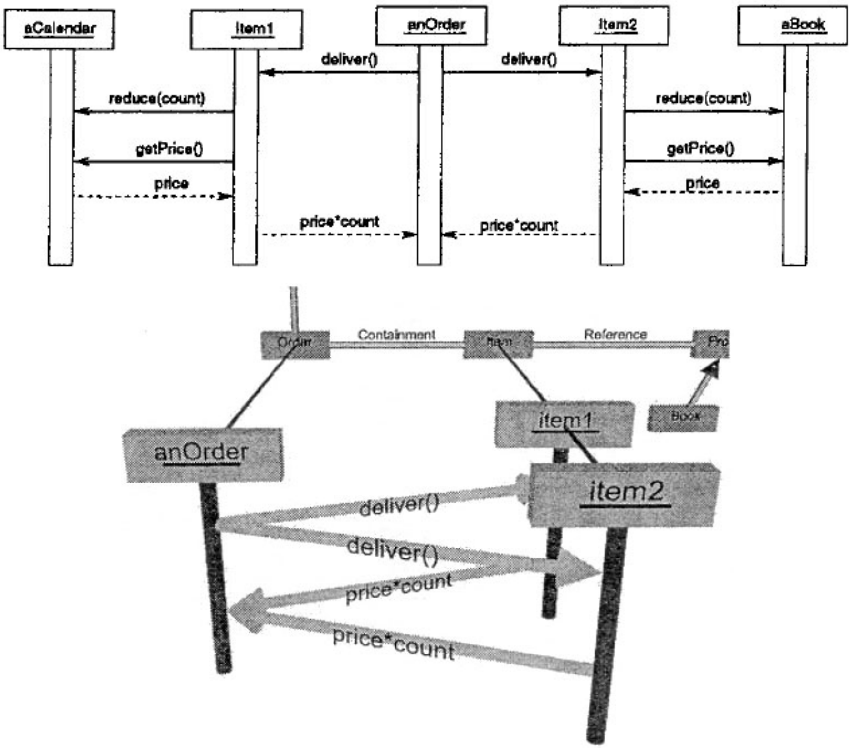

Abbildung 11: UML-Sequenzdiagramm (oben 2D, unten 3D), Quelle: [14]

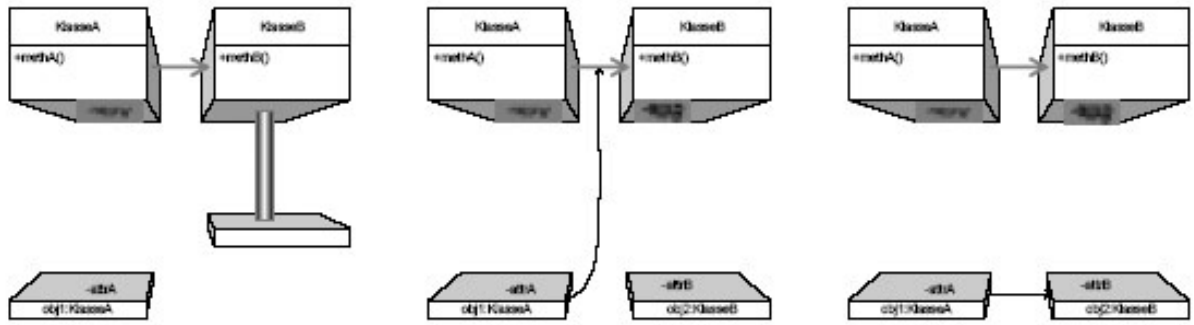

Abbildung 12: Schnappschüsse einer Instanziierungsanimation, Quelle: [15]

Objekte auf den Betrachter „zugeflogen“ kommen, um im Vordergrund zu interagieren.

5 Ausblick

Alle drei vorgestellten Konzepte verdienen einer weiteren Betrachtung. Gerade im Bereich der Graph-basierten Darstellung gibt es eine Vielzahl von Veröffentlichungen. Doch auch Geondiagramme haben im Bereich der Softwarekonstruktion hohe Beachtung gefunden, was wohl u. a. auf ihre Nähe zu den klassischen UML-Diagrammen zurückzuführen ist. Es gibt sogar eine Überschneidung dieser beiden Visualisierungskonzepte. Schließlich kann man Geondarstellungen als eine Art Spezialfall von Graphvisualisierung auffassen. Man könnte den Knotentypen (z. B. Klassen) und Kantentypen (z. B. Generalisierung oder Assoziation) entsprechende Geonendarstellungen zuordnen, dann handelt es sich bei einem Geondiagramm um eine spezielle Darstellung eines (z. B. Klassenbeziehungs-) Graphen.

Graphvisualisierungen sind zwar nicht unbedingt so intuitiv wie Geondiagramme, doch ist meist eine kompaktere Darstellung als bei Geondiagrammen möglich. Allgemein scheinen die Vorteile von Graphdarstellungen im Erfassen von (oftmals globaleren) Zusammenhängen zu liegen, während die Vorteile der Geondarstellung eher im intuitivem Verständnis von Beziehungen bestimmter Elemente liegen. Graphvisualisierungen sind somit wohl hauptsächlich für die Softwareanalyse einzusetzen, während Geondiagramme eher der Entwicklung dienlich sind. Dies ist jedoch immer von der aktuellen Aufgabe abhängig, ein Pauschalurteil wird wohl kaum möglich sein.

Die Stärke von animiertem 3D UML liegt dagegen eher in der „Einfachheit“. Es wird auf Bekanntes gesetzt und somit ergibt sich ein Wiedererkennungseffekt. Sollte UML jedoch durch eines der ersten beiden Konzepte ersetzt werden, so wird sicherlich auch dieser Ansatz schnell verworfen werden. Hat man sich mit Geondiagrammen angefreundet, sind animierte Geonen animierten 3D UML-Elementen bestimmt ebenbürtig, wenn nicht sogar überlegen. Bis dahin stellt 3D UML allerdings eine sinnvolle Bereicherung dar.

Zur Darstellung dynamischer Softwareaspekte können in allen drei Fällen bei umfangreicheren Projekten allenfalls vorgefertigte Animationen, die Annahmen über oder Aufzeichnungen des Verhaltens der Software visualisieren, genutzt werden. Eine wirkliche Realzeit-Darstellung (wie sie beispielsweise beim Debugging nützlich sein könnte) ist aufgrund des hohen Rechenaufwands zur Zeit wohl nicht möglich.

Literatur

[1] Biederman, Irving: ***Recognition-by-Components: A Theory of Human Image Understanding***. Psychological Review, 94(2):115–147, 1987.

[2] Irani, Pourang und Colin Ware: ***Diagramming Information Structures Using 3D Perceptual Primitives***. ACM Transactions on Computer-Human Interactions, 10(1):1–19, March 2003.

[3] Irani, Pourang, Colin Ware und Maureen Tingley: ***Using Perceptual Syntax to Enhance Semantic Content in Diagrams***. IEEE Computer Graphics and Applications, 21(5), September 2001.

[4] Casey, Ken und Dr. Chris Exton: ***A Java 3D Implementation of a Geon Based Visualisation Tool for UML***. In: ***Proceedings of the 2nd International Conference on Principles and Practise of Programming in Java***, Seiten 63–65, 2003.

[5] Dwyer, Tim: ***Three Dimensional UML Using Force Directed Layout***. In: Eades, Peter und Tim Pattison (Herausgeber): ***Conferences in Research and Practice in Information Technology***, Band 9, 2001.

[6] CAIDA: ***Homepage des Walrus Projekts***. URL, 2004. http://www.caida.org/tools/visualization/walrus/ (zuletzt gesichtet: 15.9.2004).

[7] Herman, Ivan, Guy Melançon und M. Scott Marshall: ***Graph Visualization and Navigation in Information Visualization: A Survey***. IEEE Transactions on Visualization and Computer Graphics, 6(1):24–43, January-March 2000.

[8] Munzner, Tamara: ***Interactive Visualization of large Graphs and Networks***. Doktorarbeit, Stanford University, June 2000.

[9] Furnas, G. W.: ***Generalized fisheye views***. In: ***Proceedings on the conference on Human Factors in Computing Systems***, Seiten 16–23, Boston, March 1986.

[10] Lewerentz, Claus und Andreas Noack: ***CrocoCosmos - 3D Visualization of Large Object-Oriented Programs***. In: Jünger, Michael und Petra Mutzel (Herausgeber): ***Graph Drawing Software***, Seiten 279–297. Springer-Verlag, 2003.

[11] Noack, Andreas: ***An Energy Model for Visual Graph Clustering***. In: Liotta, Guiseppe (Herausgeber): ***Graph Drawing***, Band 2912 der Reihe ***Lecture Notes in Computer Science***, Seiten 425–436. Springer, September 2003.

[12] Noack, Andreas: ***Energy Models for Drawing Clustered Small-World Graphs***. Computer Science Report, 7, 2003.

[13] Noack, Andreas: ***Homepage des CrocoCosmos Projekts***. URL, 2004. http://www.software-systemtechnik.de/crococosmos (zuletzt gesichtet: 15.9.2004).

[14] Radfelder, Oliver und Martin Gogolla: ***On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation***. In: ***Proceedings of the working conference on Advanced visual interfaces***, Seiten 292–295. ACM Press, 2000.

[15] Steimann, F., U. Thaden, W. Siberski und W. Nejdl: ***Animiertes UML als Medium für die Didaktik der objektorientierten Programmierung***. Modellierung 2002, 2002. GI Lecture Notes in Informatics.