Hierarchie-basierte 3D Visualisierung großer Softwarestrukturen

Michael Balzer
Fachbereich Informatik und Informationswissenschaft, Universität Konstanz
balzer@inf.uni-konstanz.de

Zusammenfassung: Moderne objektorientierte Softwaresysteme bestehen aus Tausenden miteinander in Verbindung stehenden Komponenten. Methoden der Visualisierung können eingesetzt werden, um Entwickler beim Verstehen dieser komplexen hierarchischen Systeme zu unterstützen. Die vorliegende Arbeit stellt eine dreidimensionale Visualisierung vor, bei der die statische Struktur objektorientierter Software durch die Anordnung von dreidimensionalen Objekten auf einer zweidimensionalen Fläche repräsentiert wird. Die visuelle Komplexität in der Darstellung wird durch die Verwendung von dynamischen Transparenzen in Abhängigkeit von der Position des Betrachter reduziert. Zur Repräsentation der Relationen zwischen den Komponenten des Softwaresystems wird der Ansatz eines ‚Hierarchischen Netzes' vorgestellt.

Stichworte: Softwarevisualisierung, Level-of-Detail, Voronoi-Diagramme

1 Einführung

Softwaresysteme gehören heute zu den komplexesten Artefakten die von Menschen geschaffen werden. In vielen Anwendungsbereichen werden objektorientierte Systeme entwickelt, die aus Millionen Zeilen Quelltext und vielen Tausend Komponenten bestehen. Die Zeiträume für die Entwicklung dieser Systeme sind dabei zum Einen sehr groß, zum Anderen übersteigen die Ausgaben für Wartung und Reengineering zumeist bei weitem die Kosten des ursprünglichen Designs und der Implementierung. Methoden der Visualisierung können eingesetzt werden, um existierende Softwaresysteme effizienter und genauer zu analysieren und zu verstehen.

Softwarevisualisierung, als ein Teilgebiet der Informationsvisualisierung, stellt die Verwendung verschiedener graphischer Repräsentationen zum besseren Verständnis von Softwaresystemen dar. Dabei existieren viele Ansätze, die jeweils verschiedene Aspekte von Programmen veranschaulichen, z.B. die statische Struktur, das Laufzeitverhalten oder den Entwicklungsprozess einer Software [Vis02, Sof03]. In dieser Arbeit beschränken wir uns auf die Visualisierung der statischen Struktur, die vor allem im Bereich der Qualitätskontrolle und des Reengineerings großer Softwaresysteme eine bedeutende Rolle spielt.

Das momentan wohl populärste Werkzeug im Bereich Softwarevisualisierung ist die Unified Modeling Language [Obj03]. UML ist jedoch nur sehr bedingt geeignet, um große Softwaresysteme anschaulich darzustellen. Eine andere Richtung schlagen Werkzeuge wie Rigi [MOTU93], SHriMP [SM95] und Portable Bookshelf [FHK+97] ein, die automatisch Softwarevisualisierungen

in Form von zweidimensionalen Graphen erstellen, die jedoch bei großen Systemen unüberschaubar werden. Nach Einführung von dreidimensionalen Graphen im Bereich Softwarevisualisierung durch Koike [Koi92] und Reiss [Rei95] wurden viele Möglichkeiten der Nutzung der dritten Dimension in Werkzeugen wie Narcissus [HDWB95], NestedVision3D [PFW98], ArchView [FdJ98] und CrocoCosmos [LN03] untersucht. Empirischer Studien, die die Effektivität von zweidimensionalen und dreidimensionalen Graphen verglichen haben, kommen jedoch zu unterschiedlichen Ergebnissen: In einigen Studien übertreffen 3D Graphen ihre 2D Vertreter deutlich [WF96], andere Studien geben eine genau gegensätzliche Meinung wieder [WC99]. Nach unserer Erfahrung treten bei 3D Graphen zum Einen in hohem Maße Verdeckungen auf, so dass einzelne Objekte oder auch ganze Regionen eines Graphen schwer erkennbar sind, zum Anderen ist eine Orientierung zumeist schwierig. So genannte 2,5-dimensionale Visualisierungen versuchen die Vorteile von 2D und 3D Visualisierungen zu verbinden. Dabei werden ähnlich einer Landschaft dreidimensionale Objekte auf einer zweidimensionalen Fläche verteilt. Die Informationsdichte dieser Darstellungen ist deutlich größer als in 2D Visualisierungen, während gleichzeitig die Unübersichtlichkeit von 3D Visualisierungen vermieden wird. Im Bereich Softwarevisualisierung machen von dieser Landschaftsmetapher die Werkzeuge THEMA [Plo97], Software World [KM00] und Component City [CKTM02] Gebrauch. Die von ihnen generierten Darstellungen sind jedoch sehr einfach und skalieren nicht auf realistische Systemgrößen.

2 Ein strukturelles Modell objektorientierter Software

Unser strukturelles Modell objektorientierter Software unterscheidet vier Typen von Softwarekomponenten: Pakete, Klassen, Methoden und Attribute. In Java und anderen objektorientierten Programmiersprachen können Klassen andere Klassen enthalten. Da diese enthaltenen Klassen zumeist sehr einfach und vor allem sehr eng an die enthaltende Klasse gekoppelt sind, vereinigen wir diese mit der enthaltenden Klasse, wodurch das strukturelle Modell und vor allem die Visualisierung deutlich klarer wird, ohne dass dabei relevante Strukturinformationen verloren gehen. Neben den hierarchischen Beziehungen unterscheidet das Modell weitere Relationen: Ableitungen von Klassen, Methodenaufrufe und Attributzugriffe. Ein Schema dieses Modells zeigt Abbildung 1.

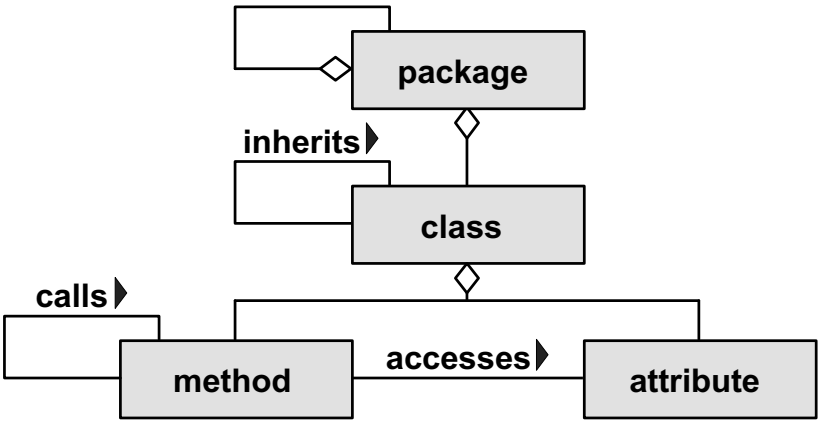


Abbildung 1: Strukturelles Modell objektorientierter Softwaresysteme

Diese strukturellen Modelle werden mit den Werkzeugen Sotograph [Sof] und SNiFF+ [Win] automatisch aus dem Quelltext existierender objektorientierter Softwaresysteme extrahiert und in Dateien im Rigi Standard Format [Won98] gespeichert. Durch diese Trennung von Extraktion und Visualisierung und der Verwendung eines standardisierten Austauschformats ist es möglich, Softwarestrukturdaten verschiedener Herkunft zu visualisieren, sofern ein geeignetes Analysewerkzeug zur Verfügung steht.

Als Datensatz für die verwendeten Abbildungen dient uns das Open Source System ‚Eclipse 3.0', dass mit seinen über 200 000 Komponenten und knapp 500 000 von uns ausgewerteten Relationen in mehr als 2 Millionen Zeilen Quelltext das größte frei verfügbare Softwaresystem darstellt.

3 Anordnung von Komponenten basierend auf ihrer Hierarchie mittels Voronoi Relaxierung

Die Anordnungen von Objekten in dieser Arbeit basieren auf der Hierarchie der Pakete, Klassen, Methoden und Attribute des visualisierten Softwaresystems. Die potentiell unendlich tiefe Hierarchie der Pakete wird durch verschachtelte Halbkugeln repräsentiert. Die äußere Halbkugel stellt die Wurzel des Hierarchiebaums dar. In ihr enthalten sind Halbkugeln die Pakete darstellen, welche direkt innerhalb dieses obersten Paketes enthalten sind. Diese Halbkugeln der zweiten Ebene der Pakethierarchie enthalten wiederum Halbkugeln für die dritte Ebene der Pakethierarchie usw. Die Größe der Halbkugeln wird an die Zahl der Methoden und Attribute in diesem Paket und allen untergeordneten Paketen angepasst. Nachdem die Pakete angeordnet wurden, werden nun die Klassen in der Mitte der Halbkugeln platziert. Jede Klasse wird dabei von einem Kreis repräsentiert, dessen Flächeninhalt wiederum mit der Anzahl enthaltener Methoden und Attribute korrespondiert. Innerhalb dieser Kreise werden die Methoden und Attribute als einfache quaderförmige Objekte platziert. Abbildung 2 illustriert dieses Anordnungsschema.

Für die Generierung dieser Anordnungen von Objekten verwenden wir die iterative Relaxierung von zweidimensionalen Voronoi Diagrammen basierend auf dem Lloyd Algorithmus [Llo82]. Eine detaillierte Beschreibung des Algorithmus findet man in [DHvS00, HKL$^+$99]. Das Ergebnis dieser Methode sind nicht-regelmäßige Verteilungen, die eine gegebene Fläche ähnlich einer Poisson-Disk-Verteilung füllen.

In den bisherigen Varianten der Voronoi Relaxierung wird stets eine feste absolute Größe für die Generatorobjekte des Voronoi Diagramms gewählt. Bei unserer Modifikation des Verfahrens bestehen im Gegensatz dazu nur feste Größenverhältnisse zwischen den Generatorobjekten, d.h. es ist nur festgelegt, dass ein Objekt *A* z.b. doppelt so gross ist wie Objekt *B*. Unser Ziel ist dabei, dass die gegebene Fläche möglichst gut von den Objekten ausgefüllt wird und gleichzeitig keine Überlappungen auftreten. Als Generatorobjekte verwenden wir Kreise, deren Flächeninhalt mit der Größe des repräsentierten Objektes korrespondiert. Die Euklidische Metrik als Distanzfunktion für die Berechnung des Voronoi Diagramms wird modifiziert zu

$$d = \sqrt{(x-x_c)^2 + (y-y_c)^2} - r_c,$$

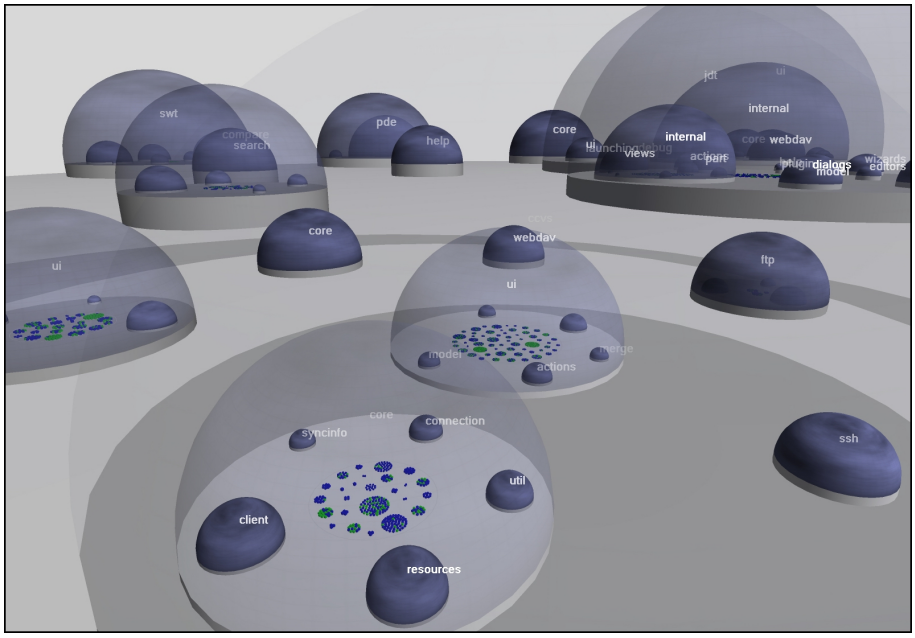


Abbildung 2: 3D Repräsentation eines Softwaresystems basierend auf dessen Hierarchie

wobei ein Kreis durch seinen Mittelpunkt (x_c, y_c) und seinen Radius r_c definiert ist. Wie schon erwähnt ist der Radius dabei nicht fest, sondern wird adaptiv in Abhängigkeit von der momentan vorliegenden Objektpositionen angepasst, wofür ein globaler maximaler Faktor m bestimmt wird, so dass keine zwei Kreise $K_1(x_1, y_1, r_1)$ und $K_2(x_2, y_2, r_2)$ existieren für die gilt:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - (r_1 + r_2) * m < 0.$$

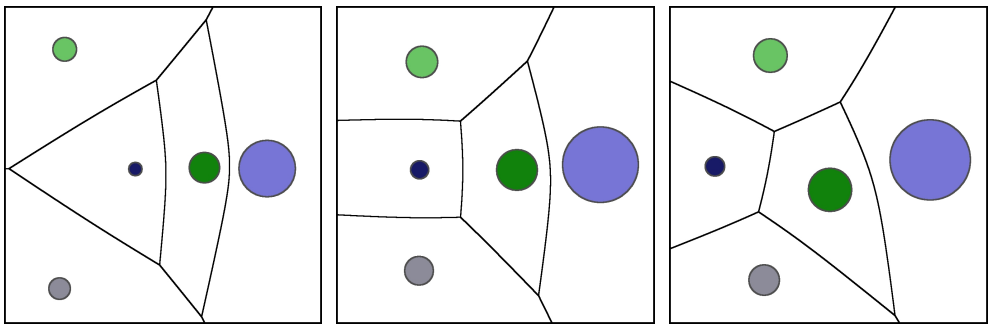
Zusätzlich sollte der Faktor m klein genug sein, so dass kein Kreis über die zur Verfügung stehende Fläche hinausgeht. Dieser Faktor m wird dann mit der relativen Größe eines jeden Kreises multipliziert und so deren absolute Größe ermittelt. Die mit diesen Bedingungen relaxierten Voronoi Diagramme generieren Kreisverteilungen unter dem Kriterium ‚Maximales Wachstum'.

Abbildung 3 illustriert den eben beschriebenen Algorithmus, wobei der Wert m mit einem zusätzlichen Faktor $f < 1$ multipliziert wird, so dass die Kreise sich nicht berühren, sondern stets in Abhängigkeit von ihrer jeweiligen Größe einen relativen minimalen Abstand zueinander halten.

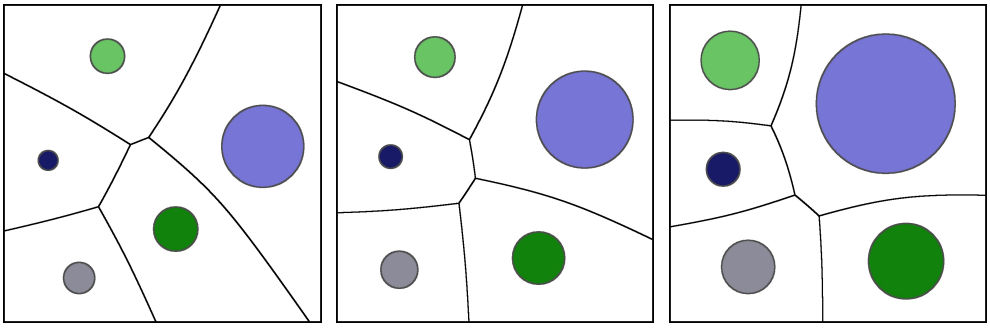
4 Hierarchische Netze

Neben der Hierarchie der Softwarekomponenten stellen die Beziehungen zwischen den Komponenten einen bedeutenden Teil der Struktur eines Softwaresystems dar. Wenn man diese Relationen als einfache direkte Linienverbindungen innerhalb einer zweidimensionalen Ebene darstellen würde, wäre es aufgrund der vielen Überlappungen und Verdeckungen quasi unmöglich einzelne Relationen zu differenzieren und näher zu untersuchen.

Die von uns vorgestellte Lösung zu diesem Problem nennt sich ‚Hierarchisches Netz'. Dabei werden die Relationen zwischen den Komponenten entsprechend der Hierarchie des Softwaresy-



(a) Initiale Verteilung (b) 2 Iterationen (c) 5 Iterationen



(d) 8 Iterationen (e) 10 Iterationen (f) Endzustand (27 Iter.)

Abbildung 3: Iterative Voronoi Relaxierung nach dem Kriterium ‚Maximales Wachstum‘

stems geführt. Wenn z.B. eine Relation zwischen Klasse *X* in Paket *A* und Klasse *Y* in Paket *B* besteht, und die Pakete *A* und *B* in Paket *C* enthalten sind, dann verläuft die Relation von Klasse *X* zu Paket *A*, zu Paket *C*, dann zu Paket *B* und schlussendlich zu Klasse *Y* (siehe auch Abbildung 4). Zu diesem Zweck wird über jedem Objekt in einem festen relativen Abstand zu dessen Mittelpunkt ein Punkt definiert, in dem sich alle Relation der untergeordneten Ebenen der Hierarchie sammeln und zur übergeordneten Ebene in der Hierarchie weitergeleitet werden. Durch den Umstand das die Objekte auf höheren Ebenen in der Hierarchie größer werden, wird ein dreidimensionaler Baum aus Relationen aufgespannt (Abbildung 5).

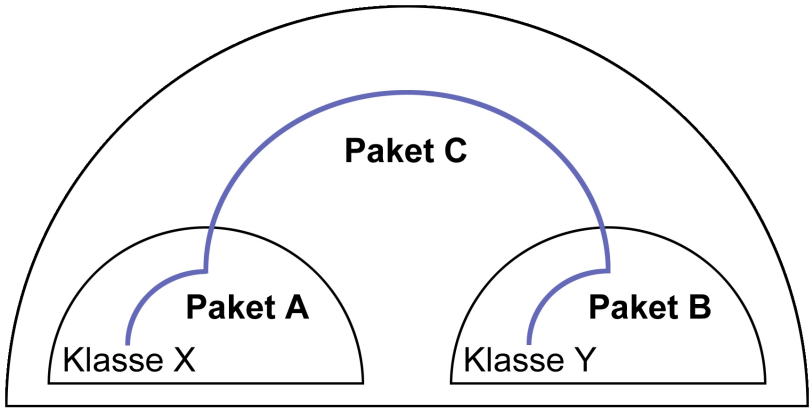


Abbildung 4: Routing von Verbindungen in einem ‚Hierarchischen Netz‘

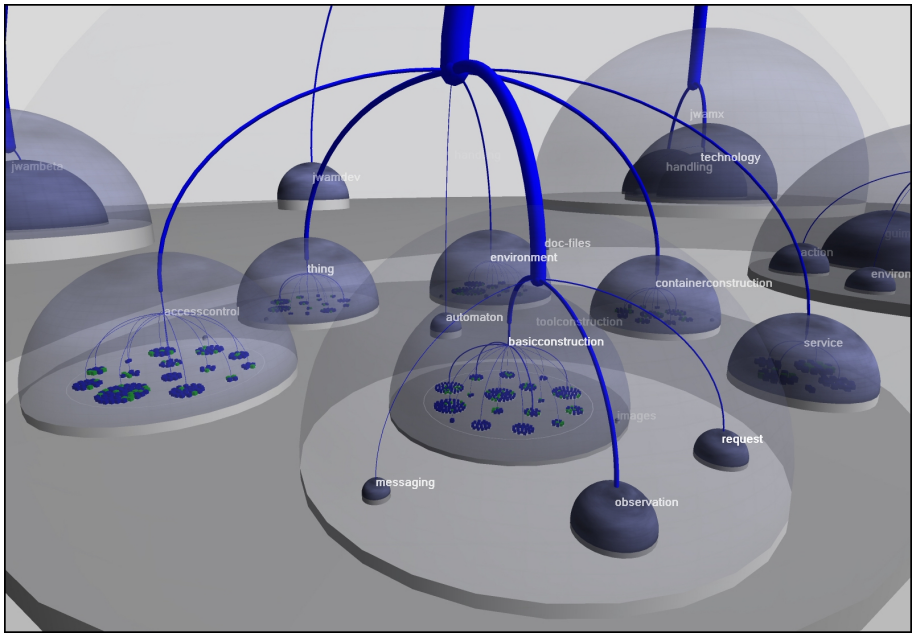


Abbildung 5: Visualisierung von Beziehungen zwischen Softwarekomponenten mittels eines ‚Hierarchischen Netzes‘

Die Art der Relationen wird durch die Farbe der Verbindungen repräsentiert. Relationen der gleichen Art und mit den gleichen Start- und Endpunkten werden zu einer Verbindung kombiniert, wobei die Anzahl der zusammengefassten Verbindungen durch den Durchmesser der neuen kombinierten Verbindung dargestellt wird, d.h. dickere Verbindung repräsentieren mehr Relationen.

Zusätzlich kann der Nutzer der Visualisierung auf die Darstellung der Relationen Einfluss nehmen. Bei der ersten Möglichkeit kann man explizit festlegen welche Arten von Relationen angezeigt werden – z.B. kann man sich ausschließlich alle Ableitungen zwischen Klassen anzeigen lassen. Die zweite Möglichkeit ist die Auswahl einer oder mehrerer Softwarekomponenten und die damit verbunden Anzeige von ausschließlich den Relationen, die eine Verbindung mit der oder den gewählten Komponenten besitzen. Diese beiden kombinierbaren Optionen ermöglichen eine klarere Darstellung und Verfolgung der ausgewählten Relationen.

5 Adaptive Transparenzen für Level-of-Detail

Wie schon in Abschnitt 3 erörtert, werden für die Repräsentation von Paketen verschachtelte Halbkugeln verwendet. In Anlehnung an [RG93] erfolgt die Darstellung dieser Halbkugeln mit Hilfe von Transparenzen. Dadurch lässt sich der Inhalt eines Systems bzw. eines Teils des selbigen betrachten, während gleichzeitig die Komplexität der Darstellung reduziert wird. Wenn man die Oberflächen der Halbkugeln komplett transparent darstellen würde, wäre eine Interpretation der Darstellung aufgrund der Fülle präsentierter Informationen sehr schwierig. Wenn im Gegensatz dazu die Oberflächen vollständig opak wären, würden Objekte innerhalb der Pakete nicht sichtbar sein. Die Verwendung von Transparenzen löst beide Probleme.

Im Gegensatz zu [RG93] ist der Grad der Transparenz nicht unveränderlich, sondern wird dynamisch an den Betrachterpunkt angepasst. Wenn der Abstand zwischen Betrachter und Halbkugel größer als das Fünffache ihres Radius ist, wird die Halbkugel opak dargestellt. Wenn der Abstand kleiner als das Doppelte des Radius ist, erfolgt die Darstellung der Halbkugeln vollständig transparent. Zwischen diesen beiden Zuständen erfolgt eine stufenlose Überblendung. Abbildung 6 veranschaulicht dies anhand eines Zooms in ein Softwaresystem.

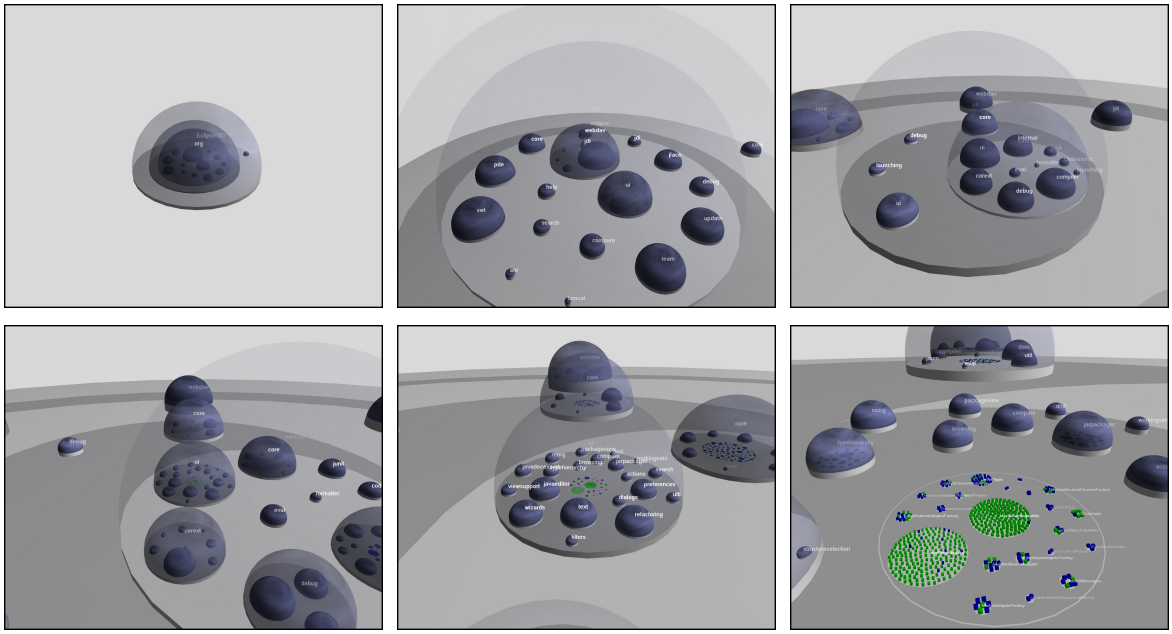


Abbildung 6: Zoom in das Open Source System ‚Eclipse 3.0‘

Das Ausblenden entfernter Hierarchieebenen ermöglicht so selbst eine Darstellung von Szenen mit einer sehr tiefen Hierarchie. Ein positiver Nebeneffekt ist, dass das Innere von vollständig opaken Halbkugeln nicht dargestellt werden muss, wobei normalerweise mehr als 90 % aller Halbkugeln opak sind.

6 Ausblick

Es sei angemerkt, dass die vorgestellte Visualisierung als interaktives Echtzeitsystem realisiert ist. Die Bildwiederholraten auf einem Rechner mit 3 GHz und einer Nvidia GeForce 5800 Grafikkarte bewegen sich zwischen 20 und 60 Bildern pro Sekunde.

In zukünftigen Arbeiten sollen weitere Möglichkeiten der Landschaftsmetapher im Rahmen der Softwarevisualisierung untersucht werden. Eine dabei verfolgte Richtung wird im Gegensatz zum momentan verwendeten auf der Hierarchie basierten Anordnungsschema die Untersuchung von Anordnungen auf Basis der Beziehungen zwischen den Komponenten sein. Desweiteren wird die Abbildung von Softwaremetriken auf die Objekte in der Visualisierung untersucht werden. So könnte z.B. die Höhe von Methoden repräsentierenden Objekten proportional zur Anzahl ihrer Zeilen im Quelltext sein.

Literatur

[CKTM02] Stuart M. Charters, Claire Knight, Nigel Thomas, and Malcolm Munro. Visualisation for informed decision making; from code to components. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 765–772. ACM, 2002.

[DHvS00] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3):40–51, 2000.

[FdJ98] Loe Feijs and Roel de Jong. 3d visualization of software architectures. *Communications of the ACM*, 41(12):73–78, 1998.

[FHK+97] Patrick J. Finnigan, Richard C. Holt, Ivan Kalas, Scott Kerr, Kostas Kontogiannis, Hausi A. Müller, John Mylopoulos, Stephen G. Perelgut, Martin Stanley, and Kenny Wong. The Software Bookshelf. *IBM Systems Journal*, 36(4):564–593, 1997.

[HDWB95] Robert J. Hendley, Nicholas S. Drew, Andrew Wood, and Russell Beale. Narcissus: Visualizing information. In *Proceedings of International Symposium on Information Visualization*, pages 90–96, 1995.

[HKL+99] Kenneth E. Hoff, John Keyser, Ming C. Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics (SIGGRAPH)*, pages 277–286. ACM, 1999.

[KM00] Claire Knight and Malcolm Munro. Virtual but visible software. In *Proceedings of the International Conference on Information Visualisation (IV)*, pages 198–205. IEEE Computer Society, 2000.

[Koi92] Hideki Koike. An application of three-dimensional visualization to object-oriented programming. In *Proceedings of the Workshop on Advanced Visual Interfaces (AVI)*, pages 180–192. World Scientific, 1992.

[Llo82] S. Lloyd. Least square quantization in PCM. In *IEEE Transactions on Information Theory*, volume 28, pages 129–137, 1982.

[LN03] Claus Lewerentz and Andreas Noack. CrocoCosmos – 3d visualization of large object-oriented programs. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, pages 279–297. Springer-Verlag, 2003.

[MOTU93] Hausi A. Müller, Mehmet A. Orgun, Scott R. Tilley, and James S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5(4):181–204, 1993.

[Obj03] Object Management Group Inc. OMG Unified Modeling Language Specification, 2003.

[PFW98] Greg Parker, Glenn Franck, and Colin Ware. Visualization of large nested graphs in 3d: Navigation and interaction. *Journal of Visual Languages and Computing*, 9(3):299–317, 1998.

[Plo97] Damien Ploix. Observation de programmes par la combinaison d'analogies. In *Actes de la conférence Intelligence Artificielle et Complexité*, pages 150–156, 1997.

[Rei95] Steven P. Reiss. An engine for the 3d visualization of program information. *Journal of Visual Languages and Computing*, 6(3):299–323, 1995.

[RG93] Jun Rekimoto and Mark Green. The Information Cube: Using transparency in 3d information visualization. In *Proceedings of the 3rd Annual Workshop Information Technologies & Systems (WITS)*, pages 125–132, 1993.

[SM95] Margaret-Anne Storey and Hausi Müller. Manipulating and documenting software structures using SHriMP views. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 275–284. IEEE Computer Society, 1995.

[Sof] Software-Tomography GmbH. http://www.softwaretomography.com.

[Sof03] *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)*. ACM, 2003.

[Vis02] *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE Computer Society, 2002.

[WC99] Ulrika Wiss and David A. Carr. An empirical study of task support in 3d information visualizations. In *Proceedings of the International Conference on Information Visualisation (IV)*, pages 392–399. IEEE Computer Society, 1999.

[WF96] Colin Ware and Glenn Franck. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics*, 15(2):121–140, 1996.

[Win] Wind River Systems Inc. http://www.windriver.com.

[Won98] Kenny Wong. *Rigi User's Manual, Version 5.4.4*, 1998. http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/.