TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Lehrstuhl für Rechnernetze

Bachelor-Arbeit

ENTWICKLUNG EINES KONZEPTES FÜR DAS INTERAKTIVE UND DIAGRAMMSPEZIFISCHE LAYOUT VON GRAPHBASIERTEN SOFTWAREDIAGRAMMEN

Lukáš Kubánek
Geboren am: 2. September 1991
Matrikelnummer: 3668863
Immatrikulationsjahr: 2010

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Betreuer
Dr. Thomas Springer

Betreuender Hochschullehrer
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Eingereicht am: 14. Oktober 2014

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Systemarchitektur, Lehrstuhl Rechnernetze

AUFGABENSTELLUNG FÜR DIE BACHELORARBEIT

Thema: **Entwicklung eines Konzeptes für das interaktive und diagrammspezifische Layout von graphbasierten Softwarediagrammen**

Name, Vorname	Kubanek, Lukas	Studiengang	Bachelor Informatik
Matrikel-Nr.	3668863	Projekt/Schwerpunkt	Mobile and Ubiquitous Comp.
Betreuer	Dr. Thomas Springer	Externer Betreuer	-
Beginn am	16. Juni 2014	Einzureichen am	7. September 2014

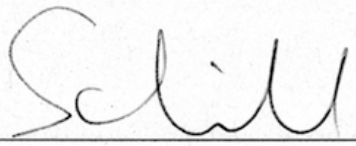
ZIELSTELLUNG

Die Entwicklung von Apps erfolgt häufig in kleinen Teams oder durch einzelne Entwickler. Im Vordergrund steht dabei die Implementierung des Codes, die Modellierung und Dokumentation mit UML-Werkzeugen wird in der Regel als zu aufwändig betrachtet und erfolgt gar nicht oder mit Papier oder Zeichenprogrammen. Dennoch ist ein gewisser Grad an Dokumentation durch Modelle für die Entwicklung vorteilhaft. Nach dem Ansatz des Agile Modeling werden leichtgewichtige Modelle für die Dokumentation während der Entwicklung erstellt. UML-Modelle und Werkzeuge sind für diese Zwecke aber zu schwergewichtig. Für den Ansatz des Agile Modeling werden Werkzeuge benötigt, die eine einfache Modellerstellung ermöglichen. Ein wesentlicher Aspekt ist dabei ein automatisches Layout.

Ziel der Bachelorarbeit ist die Entwicklung eines Konzeptes für ein interaktives, diagrammspezifisches Layout für graphenbasierte Softwarediagramme. Im Rahmen der Arbeit sollen existierende Ansätze zum automatischen Layout analysiert werden. Die Erkenntnisse der Analyse sollen dann bei der Konzeption interaktiver, diagrammspezifischer Layoutmechanismen verwendet werden. In der Bachelorarbeit soll der Schwerpunkt auf graphenbasierten Diagrammen und konkret auf Klassendiagrammen liegen. Die entwickelten Konzepte sollen prototypisch implementiert und in einer Nutzerstudie evaluiert werden.

SCHWERPUNKTE

- Recherche und Analyse existierender Ansätze für das interaktive und diagrammspezifische Layout
- Konzeption von Layoutmechanismen für graphbasierte Softwarediagramme mit Schwerpunkt auf Klassendiagrammen
- Prototypische Implementierung eines Werkzeugs für die agile Modellierung von Klassendiagrammen
- Evaluation der umgesetzten Layoutmechanismen mit einer Nutzerstudie



Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill (betreuender Hochschullehrer)

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Arbeit mit dem Titel *Entwicklung eines Konzeptes für das interaktive und diagrammspezifische Layout von graphbasierten Softwarediagrammen* selbstständig und ohne unzulässige Hilfe Dritter verfasst habe. Es wurden keine anderen als die in der Arbeit angegebenen Hilfsmittel und Quellen benutzt. Die wörtlichen und sinngemäß übernommenen Zitate habe ich als solche kenntlich gemacht. Während der Anfertigung dieser Arbeit wurde ich nur von folgenden Personen unterstützt:

Dr. Thomas Springer

Weitere Personen waren an der geistigen Herstellung der vorliegenden Arbeit nicht beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Hochschulabschlusses führen kann.

Dresden, 14. Oktober 2014

Lukáš Kubánek

INHALTSVERZEICHNIS

1. EINLEITUNG

1.1. MOTIVATION

Der Einsatz von Agilität in der Softwareentwicklung, bekannt unter dem Begriff **agile Softwareentwicklung**, erfreut sich heutzutage einer großen Beliebtheit. In der Umfrage „State of Agile" der Firma *VersionOne* aus dem Jahr 2013 haben 88% der befragten Personen aus nordamerikanischen und europäischen Softwareentwicklungsunternehmen bestätigt, dass in ihren Organisationen agile Softwareentwicklung eingesetzt wird. Die Umfrage zeigt weiterhin eine steigende Tendenz dieses Anteils in den letzten drei Jahren [Ver14].

Agile Softwareentwicklung basiert auf den Grundsätzen aus [BBB+01]. Im Vergleich zu den traditionellen und schwergewichtigen Softwareentwicklungsprozessen (wie z.B. *Unified Process* oder dem Wasserfallmodell) sind die agilen schlanker, flexibler und an die Bedürfnisse der heutigen Zeit angepasst. Des Weiteren stellt die agile Softwareentwicklung das Schreiben von Quellcode in den Fokus und unterstützt dies durch zahlreiche Praktiken wie z.B. testgetriebene Entwicklung, „Code-Refactoring" oder „Code-Reviews". Insbesondere bei kleinen Entwicklerteams und Einzelentwicklern, die Anwendungen für moderne Plattformen entwickeln[1], wird während der Entwicklung in der Regel auf Modellierung und umfassende Dokumentation mithilfe von UML-Diagrammen verzichtet.

Der agile Ansatz lässt sich jedoch auch auf die Softwaremodellierung übertragen. Die Grundlage dafür bildet die in [Amb02] von Scott W. Ambler präsentierte Methodik **Agile Modeling**. Diese Praktiken-orientierte Methodik zeigt, dass effektive Softwaremodellierung auch im agilen Umfeld Vorteile bringt. Des Weiteren wird empfohlen, mit den einfachsten Tools leichtgewichtige Modelle zu erstellen, die einen bestimmten Zweck erfüllen, wie z.B. einen Sachverhalt zu veranschaulichen oder die Kommunikation zwischen mehreren Teammitgliedern zu verbessern. Weiterhin soll keine zu komplizierte Notation[2] verwendet werden. Wenn der Inhalt der

[1] mobile Apps, Webanwendungen, Endbenutzer-Desktopanwendungen

[2] Der Autor rät dazu, eine hilfreiche Teilmenge der UML-Notation aber auch weitere Diagrammtypen und deren Notationen wie z.B. Flowcharts zu verwenden.

Modelle bereits im Code umgesetzt ist und die Modelle dem Projekt keinen Mehrwert mehr geben, sollen sie verworfen werden [Amb02].

Zum einen können die Modelle in Form von Skizzen auf **Papier oder Whiteboards** gezeichnet werden. Dies ist eine schnelle, praktische und unmittelbare Lösung. Leider ist sie für nachträgliche Änderungen und Korrekturen nicht geeignet, erschwert die Lesbarkeit und verfügt über keine syntaktische und semantische Überprüfung.

Zum anderen bietet sich für die Erstellung der Modelle der Einsatz eines **Zeichentools**[3] an. Die Zeichentools beheben die Probleme des statischen Charakters von Papier und bringen zusätzlich einen Vorteil durch die Digitalisierung mit sich. Jedoch unterstützen sie in der Regel keine Semantik und oft ist auch die Unterstützung der Syntax mangelhaft. So muss z.B. in einigen Zeichentools das Klassenelement für Klassendiagramme mit einem Rechteck und mehreren Textfeldern zusammengebaut werden.

Die **CASE-Tools**[4], die für Softwaremodellierung konzipiert sind und sowohl die Syntax als auch die Semantik der Modellierungssprache unterstützen, können ebenfalls für die Erstellung der Modelle verwendet werden. Obwohl die Nutzung von CASE-Tools in der Methodik *Agile Modeling* nicht untersagt ist, sind diese Werkzeuge für die Erzeugung von einfachen Modellen in der Regel oft zu komplex und schwergewichtig.

Es stellt sich heraus, dass verfügbare Software-Tools nur eine unbefriedigende Unterstützung für die agile Modellierung bieten. Ein für die agile Modellierung geeignetes Tool soll neben einer einfachen Modellerstellung und den Prinzipien aus [Amb02] auch die syntaktischen und semantischen Aspekte der Modellierungssprache unterstützen. Weiterhin ist es wichtig, dass der Inhalt des Diagramms im Fokus steht. Schließlich soll die Interaktion mit dem Diagramm gefördert werden, unter anderem durch eine intelligente Layout-Unterstützung, die den Gegenstand dieser Arbeit bildet.

1.2. ZIELSTELLUNG

In dieser Arbeit soll ein Konzept für eine interaktive und diagrammspezifische Layout-Unterstützung für visuelle Editoren aus dem Desktop-Bereich[5] erarbeitet werden. Der Schwerpunkt wird dabei auf graphbasierte Softwarediagramme gelegt, insbesondere wird auf Klassendiagramme eingegangen. Das Ziel ist es, einen möglichen Ansatz für die interaktive Layout-Unterstützung in leichtgewichtigen Softwaremodellierungswerkzeugen vorzustellen.

Im Rahmen der Arbeit sollen bestehende Ansätze für das Layout von Diagrammen untersucht und nach relevanten Kriterien kategorisiert werden. Die durchgeführte Analyse soll bei dem Entwurf des Konzeptes berücksichtigt werden. Die vorgestellten Layout-Mechanismen sollen

[3] Beispiel eines Zeichentools ist *OmniGraffle*, das in Abschnitt 3.2.1.1 näher behandelt wird.
[4] Computer-aided software engineering
[5] Die Desktop-Editoren zeichnen sich dadurch aus, dass sie auf Fenstern basieren und die Bedienung mit der Maus und Tastatur erfolgt. Andere Eingabemöglichkeiten wie z.B. Multi-Touch werden in dieser Arbeit nicht behandelt (siehe Abschnitt 1.3).

in Form eines Prototyps implementiert und anschließend mithilfe einer Nutzerstudie evaluiert werden.

1.3. RAHMENBEDINGUNGEN

Obwohl es bereits Ansätze für die Bearbeitung von Diagrammen mit fortgeschrittenen Bedienungsparadigmen wie Multi-Touch gibt (z.B. in [FHD10]), beschäftigt sich diese Arbeit ausschließlich mit Ansätzen für das Layout von Diagrammen im Bereich der Desktop-Anwendungen. Der Grund dafür ist, dass für die Erstellung und Bearbeitung von Diagrammen der Einsatz von Desktop-Systemen überwiegt.

1.4. AUFBAU DER ARBEIT

Die Arbeit ist in sieben Kapitel unterteilt. Nach diesem einführenden Kapitel wird zunächst in Kapitel 2 auf die grundlegenden Begriffe bezüglich des Layouts von Diagrammen und dessen ästhetischen Prinzipien eingegangen. In Kapitel 3 werden bestehende Ansätze für das Layout von Diagrammen aus kommerziellen Anwendungen, Software-Bibliotheken und Forschungsprojekten kategorisiert und im Detail beleuchtet. Dem neu entwickelten Konzept für das interaktive und diagrammspezifische Layout von graphbasierten Softwarediagrammen widmet sich Kapitel 4, in dem seine Funktionsweise erklärt und die einzelnen Bestandteile vorgestellt werden. In Kapitel 5 werden die umgesetzten Funktionen und die Architektur der prototypischen Implementierung beschrieben. Den Gegenstand von Kapitel 6 bildet die Evaluation des umgesetzten Konzeptes, die sich auf eine durchgeführte Nutzerstudie stützt und deren Ergebnisse präsentiert sowie auswertet. Anschließend wird die Arbeit in Kapitel 7 zusammengefasst und es werden Vorschläge für die weitere Entwicklung aufgeführt.

2. GRUNDLAGEN

Dieses Kapitel beschäftigt sich mit den Grundlagen, die für das Verständnis der Problematik der Layout-Berechnung in Diagrammen notwendig sind. Zunächst werden in Abschnitt 2.1 die grundlegenden Begriffe eingeführt und erläutert. Anschließend werden in Abschnitt 2.2 die ästhetischen Prinzipien für das Layout von allgemeinen Graphen und Klassendiagrammen der Notationssprache UML[6] zusammengefasst.

2.1. GRUNDLEGENDE BEGRIFFE

2.1.1. GRAPHBASIERTE DIAGRAMME

Viele Diagramme, die sich mit einer visuellen Sprache beschreiben lassen, basieren auf der Struktur von Graphen und werden durch Knoten (engl.: „node“) und Kanten (engl.: „edge“ bzw. „link“) gebildet [Eic05]. Die grafische Repräsentation der graphbasierten Diagramme (manchmal in der Literatur auch Node-Link-Diagramme genannt) besteht aus der Zusammensetzung von Textelementen und geometrischen Objekten wie Rechtecken, Ellipsen und Linien [Wyb08]. Die Struktur von Graphen kann um die Möglichkeit der Verschachtelung von Knoten und deren Verbindungen durch Kanten erweitert werden [Sie03, Wyb08]. Trotz des häufigen Einsatzes dieser Erweiterung in vielen Diagrammtypen wird sie für die Zwecke dieser Bachelor-Arbeit außer Acht gelassen.

Zu den graphbasierten Diagrammen gehört eine große Anzahl an Softwarediagrammen, wie z.B. Klassendiagramme[7], Objektdiagramme bzw. Use-Case-Diagramme der Notationssprache UML, Netzwerkdiagramme sowie Flowcharts.

[6] Unified Modeling Language (`http://www.uml.org`)

[7] Obwohl der UML-Standard eine Verschachtelung der Klassen in Pakete zulässt [Fow03], sind an dieser Stelle einfachere Varianten der Klassendiagramme gemeint, wie z.B. die konzeptuellen Klassendiagramme nach [Amb04].

2.1.2. LAYOUT

Durch die Spezifikationen der konkreten Diagrammtypen wird in der Regel ausschließlich die Struktur und die grafische Repräsentation festgelegt. Dagegen wird die Bestimmung der konkreten Eigenschaften der geometrischen Objekte wie z.B. ihre Anordnung nicht spezifiziert. Diese kann sich nach Konventionen, den Präferenzen der Nutzer bzw. nach ästhetischen Prinzipien richten [Mai12, Wyb08]. Durch eine Anpassung dieser Eigenschaften kann zum einen die Lesbarkeit des Diagramms verbessert und zum anderen können bestimmte Eigenschaften des Diagramms hervorgehoben werden [Mai12]. Weiterhin kann durch das Layout eine sekundäre Notation geschaffen werden, die dem Diagramm eine zusätzliche Bedeutung gibt [SGMMS03]. In den meisten Diagramm-Editoren ist die Erstellung des Layouts dem Nutzer überlassen. Das Layout kann aber auch durch einen automatischen Algorithmus berechnet werden [Mai12]. Die einzelnen Ansätze für das Layout von Diagrammen werden in Kapitel 3 vorgestellt.

Das Layout eines Diagramms wird als eine Zuordnung von Layout-Eigenschaften zu den Objekten des Diagramms definiert. Zu den bedeutsamen Layout-Eigenschaften der graphbasierten Diagramme (siehe Abschnitt 2.1.1) gehören vor allem die Positionen und Größen der Knoten und Routen der Kanten.

2.1.3. MENTALES MODELL

Während der Arbeit mit einem Diagramm wird ein mentales Modell (engl.: „mental map“) entwickelt, welches beschreibt, wie die Struktur und die Bedeutung des Diagramms durch den Nutzer wahrgenommen werden [Bra01, GSE+14].

Wenn der Inhalt bzw. das Layout des Diagramms geändert werden, muss sich der Nutzer an diese Änderung anpassen. Insbesondere in dem Fall einer automatischen Änderung ist es sehr wichtig, diese für den Nutzer möglichst nachvollziehbar durchzuführen, sodass das mentale Modell erhalten bleibt [Bra01, Mai12]. Dies kann auf zwei Wegen erreicht werden. Entweder wird versucht die Änderungen zu minimalisieren oder sie hervorzuheben (z.B. mit einer Animation oder weiteren visuellen Elementen) [Mai12].

2.2. ÄSTHETISCHE PRINZIPIEN

Die Qualität des Layouts von graphbasierten Diagrammen wird anhand von ästhetischen Prinzipien bestimmt [Mai12]. Sie beschreiben die Grundsätze für die grafische Darstellung der Elemente im Diagramm und deren Einhaltung führt zu übersichtlichen und verständlichen Zeichnungen [Sie03]. Eine besonders wichtige Rolle spielen sie in automatischen Layout-Algorithmen, die im statischen Kontext eingesetzt werden [Mai12].

Die ästhetischen Prinzipien werden zahlreich in der Literatur z.B. in [Sie03], [ES09], [Amb05], [Eic05] und [SKM93] beschrieben und nach verschiedenen Faktoren kategorisiert. Für die Zwecke dieser Arbeit wurde die Unterteilung der ästhetischen Prinzipien für allgemeine Graphen

und für Klassendiagramme ähnlich wie in [Sie03] gewählt. Eine Auflistung und kurze Zusammenfassung der bedeutsamsten Vertreter der beiden Gruppen folgt in den folgenden Abschnitten.

2.2.1. ÄSTHETISCHE PRINZIPIEN FÜR GRAPHEN

Wie bereits in Abschnitt 2.1.1 erläutert wurde, bauen die graphbasierten Diagramme auf der Struktur der Graphen auf. Für diese Struktur lassen sich ästhetische Prinzipien aufstellen, obwohl die Graphen an sich keine semantische Information besitzen [Sie03]. Sie können in der folgenden Liste nachgelesen werden:

ÄP.1 **Einhaltung der Abstände von Elementen** Die Elemente im Diagramm sollen nicht zu nahe aneinander liegen. Insbesondere sollen sich die Knoten nicht überlappen und die Kanten sollen keine Knoten schneiden. Die Abstände und Längen der Kanten sollen optimal gewählt werden, sodass das Diagramm übersichtlich bleibt [Sie03, Amb05].

ÄP.2 **Reduktion der Anzahl der Kantenkreuzungen** Da die allgemeinen Graphen nicht zwingend planar sind, lassen sich Kantenkreuzungen nicht vermeiden. Der Übersichtlichkeit halber soll versucht werden, die Anzahl der Kantenkreuzungen zu minimieren [Sie03, ES09].

ÄP.3 **Reduktion der Anzahl der Kantenknicke** Die Kanten können durch direkte Linien, Kurven oder eine Zusammensetzung von Liniensegmenten repräsentiert werden. Durch die Aufteilung der Kante in Segmente entsteht an jedem Verbindungspunkt von zwei Liniensegmenten ein Knick. Da der Verlauf der Kanten mit vielen Knicken schwer zu folgen ist, soll ihre Anzahl minimiert werden [Sie03].

ÄP.4 **Orthogonalität der Kanten** Durch die Zusammensetzung der Kanten aus ausschließlich horizontalen und vertikalen Segmenten wird trotz des Vorhandenseins von Knicken die Übersichtlichkeit gefördert [Sie03]. Somit macht der Einsatz von orthogonalen Kanten durchaus Sinn. Insbesondere vermeidet dieses Prinzip unzulässige Winkel an den Kantenknicken [Sie03] sowie Krümmungen der Kanten [Amb05].

ÄP.5 **Optimale Ausnutzung der Zeichenfläche** Um die Diagramme übersichtlicher zu gestalten, sollen die Knoten im Diagramm ausgewogen verteilt und die Gesamtfläche der Zeichnung minimiert werden [ES09, Sie03].

ÄP.6 **Symmetrie** Durch die symmetrische Anordnung von Elementen wird eine schnellere Erfassung einer Teilstruktur des Diagramms ermöglicht [Eic05, Amb05]. Obwohl dieses Prinzip oft mit anderen Prinzipien kollidiert, sollte es angestrebt werden [Sie03].

2.2.2. ÄSTHETISCHE PRINZIPIEN FÜR KLASSENDIAGRAMME

Die Klassendiagramme gehören zu häufig eingesetzten Diagrammtypen der Notationssprache UML [Fow03]. Sie beschreiben die Struktur eines objektorientierten Systems und visualisieren

Klassen und ihre Beziehungen [Sie03, Amb05]. Sie können für verschiedene Zwecke eingesetzt werden, u.a. für die Untersuchung der Konzepte für die Umsetzung von Anforderungen oder für den detaillierten Entwurf eines Systems [Amb04, Amb05]. Aufgrund der Vielseitigkeit dieses Diagrammtyps wurden die Klassendiagramme zum Schwerpunkt dieser Arbeit gewählt.

Die Problematik des Layouts von Klassendiagrammen wurde in vielen Arbeiten untersucht, z.B. in [Sie03], [Eic05] sowie [Eig04]. Die in diesen Arbeiten vorgestellten Layout-Algorithmen berücksichtigen ästhetische Prinzipien, die auf den semantischen Eigenschaften der Klassendiagramme basieren. Eine Auswahl der bedeutsamen Prinzipien wird im Folgenden aufgelistet:

ÄP.7 **Richtung von Kanten** Die Kanten in Klassendiagrammen lassen sich in hierarchische und nicht-hierarchische unterteilen [Eic05], was von der Art der Relation abgeleitet wird, die sie repräsentieren. Die hierarchischen Kanten (wie z.B. Vererbung) sollen vertikal verlaufen, während die nicht-hierarchischen Kanten (wie z.B. Assoziation) horizontal angeordnet werden sollen [ES09, Amb05]. Wichtig ist dieses Prinzip insbesondere für die Vererbungshierarchien, in den die Unterklassen unter den Oberklassen abgebildet werden, wodurch die Richtung der Kanten festgelegt wird.

ÄP.8 **Darstellung von Hierarchien** Die durch die hierarchischen Knoten dargestellte Rangordnung bildet den Gegenstand für weitere ästhetische Prinzipien. Zum einen sollen die Geschwisterknoten auf der gleichen Ebene platziert werden [Sie03]. Zum anderen sollen die Vaterknoten im Bezug zu den Kinderknoten möglichst zentriert ausgerichtet werden [ES09, Sie03]. Dies erweitert das Prinzip der Symmetrie [ES09] und wird insbesondere bei den Baumstrukturen eingesetzt [Sie03].

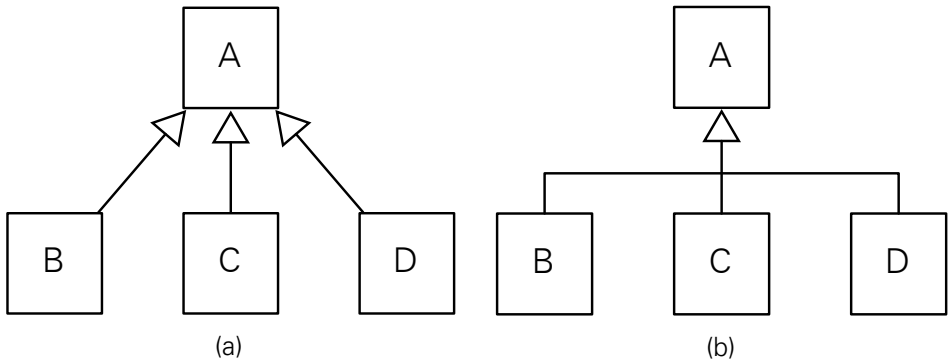


Abbildung 2.1.: Möglichkeiten der Darstellung von Vererbungsrelationen in Klassendiagrammen

ÄP.9 **Darstellung der Vererbungsrelationen** Die Vererbungsrelation kann auf zwei Arten dargestellt werden. Entweder werden wie in Abbildung 2.1(a) einzelne Linien für die jeweiligen Relationen verwendet oder sie werden in einer „Gabelform“ wie in Abbildung 2.1(b) dargestellt [Sie03]. Obwohl nach dem Standard beide Optionen zulässig sind, wird die zuletzt genannte Option trotz der Einführung von zusätzlichen Kantenknicken als übersichtlicher eingeschätzt [ES09, Sie03].

ÄP.10 **Platzierung von verwandten Elementen** Verwandte Elemente wie z.B. Pakete, Vererbungshierarchien, durch Relationen verbundene Elemente oder Design-Patterns sollen gruppiert und nah zueinander platziert werden [Eic05, Sie03]. Weiterhin sollen die einzelnen Gruppen und insbesondere die Hierarchien im Diagramm verteilt werden [Eig04].

ÄP.11 **Platzierung von nicht verbundenen Elementen** Die Elemente, die über keine Verbindung zu anderen Elementen verfügen, sollen an der Seite der Zeichnung positioniert werden [ES09].

3. BESTEHENDE ANSÄTZE FÜR DAS LAYOUT VON DIAGRAMMEN

Dieses Kapitel widmet sich der Vorstellung von bestehenden Ansätzen für das Layout von Diagrammen. In Abschnitt 3.1 werden die Ansätze zunächst kategorisiert. Anschließend werden in Abschnitten 3.2, 3.3 und 3.4 die einzelnen Hauptkategorien anhand von Beispielen ausführlich beschrieben und deren Eigenschaften erläutert.

3.1. KATEGORISIERUNG

Diese Arbeit setzt sich insbesondere mit der Interaktivität der Ansätze für das Layout von Diagrammen auseinander. Aus diesem Grund werden die bestehenden Ansätze nach dem Grad der Interaktivität und der Art der Layout-Unterstützung in drei Hauptkategorien unterteilt.

Zum einen gibt es das interaktive manuelle Layout, das auf der Freihand-Bearbeitung basiert und dem Nutzer ermöglicht, die Layout-Eigenschaften der Objekte im Diagramm wunschgemäß zu verändern, wobei dem Nutzer zusätzlich Layout-Vorschläge präsentiert werden. Das manuelle Layout wird in den meisten grafischen Editoren eingesetzt und wird in Abschnitt 3.2 näher erläutert.

Weiterhin gibt es Ansätze für das vollautomatische Layout, die optimale Layouts unter Berücksichtigung von ästhetischen Prinzipien produzieren. Sie haben einen statischen Charakter und können von dem Nutzer nur minimal beeinflusst werden. Somit sind sie nicht für interaktive Umgebungen geeignet. Mit diesen Ansätzen beschäftigt sich Abschnitt 3.3.

Schließlich gibt es Ansätze für das halbautomatische Layout, die das Layout automatisiert berechnen, lassen sich aber durch den Nutzer interaktiv beeinflussen. Demzufolge kombinieren sie die Vorteile der vorherigen beiden Kategorien. Diese Ansätze werden in Abschnitt 3.4 näher beschrieben.

3.2. MANUELLES LAYOUT

In Anwendungen zur Erstellung von Diagrammen wie z.B. Vektorgrafik-Software, Präsentationsprogrammen oder CASE-Tools wird ausschließlich das manuelle Layout unterstützt, d.h. die einzelnen Bestandteile können im Diagramm frei positioniert werden. Somit kann der Nutzer ein gewünschtes Layout erreichen, muss dafür aber viele manuelle Schritte tätigen [Eic05]. Da sich der Inhalt des Diagramms während der Erstellung ständig ändert, ist es notwendig, auch das Layout manuell anzupassen. Dies kann zur Frustration des Nutzers führen. Um einige Hürden des manuellen Layouts zu überwinden, bieten die Anwendungen eine Reihe an Funktionen, die dem Nutzer diese Tätigkeit bequemer machen.

Da das manuelle Layout auf direkter Interaktion des Nutzers mit dem Diagramm basiert, haben auch die Hilfsfunktionen einen interaktiven Charakter. Außerdem weisen sie ein temporäres Verhalten auf, d.h. bei ihrer Nutzung werden keinerlei Regeln oder Constraints erstellt, wie das meistens bei den halbautomatischen Ansätzen der Fall ist (siehe Abschnitt 3.4).

In Abschnitt 3.2.1 werden zunächst Anwendungen vorgestellt, die eine Grundlage für die Beschreibung der Hilfsfunktionen darstellen. Im Detail werden die Hilfsfunktionen in Abschnitt 3.2.2 behandelt. Schließlich werden die Eigenschaften der Ansätze für das manuelle Layout in Abschnitt 3.2.3 zusammengefasst.

3.2.1. ANWENDUNGEN ALS GRUNDLAGE

Im Folgenden werden drei Anwendungen vorgestellt, an denen die Hilfsfunktionen für das manuelle Layout gezeigt werden.

3.2.1.1. OMNIGRAFFLE

Die von *OmniGroup*[8] entwickelte kommerzielle Anwendung *OmniGraffle* ist eines der einfachsten Tools zur Erstellung von Diagrammen unter *Mac OS X* [Ols10]. *OmniGraffle* ist sehr flexibel und lässt sich für viele verschiedene Aufgaben einsetzen, z.B. von einem grafischen Entwurf einer Webseite bis hin zum Zeichnen von Klassendiagrammen. Das liegt vor allem daran, dass für die gezeichneten Diagramme kein semantisches Modell vorliegt und dass die Diagrammbestandteile als reine vektorgrafische Objekte repräsentiert werden. Detaillierte Informationen zu *OmniGraffle* können in [Omn08], [Ols10] oder auf der offiziellen Webseite[9] nachgelesen werden.

Die meisten in Abschnitt 3.2.2 beschriebenen Hilfsfunktionen für das manuelle Layout werden anhand von *OmniGraffle* 5[10] gezeigt. *OmniGraffle* bietet auch eine Unterstützung für automatisches Layout, die in Abschnitt 3.3.2.1 näher beschrieben wird.

[8] `http://omnigroup.com`

[9] `http://omnigroup.com/omnigraffle`

[10] Die aktuelle Version von *OmniGraffle* ist 6 (`http://www.omnigroup.com/blog/omnigraffle-6-is-here`), die Erkenntnisse für diese Arbeit wurden allerdings mit der Version 5 gewonnen.

3.2.1.2. KEYNOTE

Apple Keynote ist eine Anwendung zur Erstellung von Präsentationen für *Mac OS X* und *iOS*. Es bietet ähnliche Funktionen wie *Microsoft PowerPoint*[11] und bildet somit eine Alternative zu dem bekannten Präsentationsprogramm. Neben den Achsendiagrammen zur Visualisierung von Werten (wie Linien- oder Balkendiagramme) ermöglicht es *Keynote* mithilfe von vektorgrafischen Objekten einfache graphbasierte Diagramme zu erstellen. Weitere Informationen sind in [App11] oder auf der offiziellen Webseite[12] zu finden.

Keynote besitzt ähnliche oder sogar identische Hilfsfunktionen für das manuelle Layout wie *OmniGraffle*. In Abschnitt 3.2.2 werden einige Hilfsfunktionen anhand von *Keynote* 6 erklärt. Das automatische Layout wird in *Keynote* nicht unterstützt.

3.2.1.3. VISUAL PARADIGM

Visual Paradigm ist ein plattformunabhängiges CASE-Tool von der gleichnamigen Firma und unterstützt neben UML-Diagrammen auch weitere visuelle Sprachen [Vis14]. Das Tool basiert auf einer Freihandbearbeitung und bietet neben Hilfsfunktionen für das manuelle Layout (siehe Abschnitt 3.2.2) auch Unterstützung für automatische Layout-Algorithmen (siehe Abschnitt 3.3.2) [Fuh11]. Mit einer näheren Beschreibung von *Visual Paradigm* beschäftigen sich [Vis14], [Fuh11, S.313-314] und die offizielle Webseite[13].

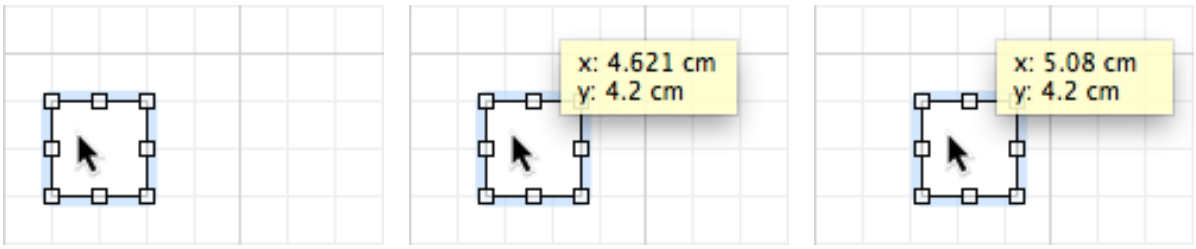
3.2.2. HILFSFUNKTIONEN FÜR MANUELLES LAYOUT

3.2.2.1. RASTER

Viele (unter anderem alle drei in Abschnitt 3.2.1 erwähnten) Visualisierungsprogramme unterstützen die Aufteilung des Diagramms in ein gleichmäßiges Raster, das durch Rasterlinien dargestellt werden kann. Diese Linien können zum Ausrichten von Objekten im Diagramm verwendet werden. Dies ist vor allem dann nützlich, wenn die Größen der Objekte eine wichtige Rolle spielen wie z.B. in Grundrissen [Omn08, Ols10, App11, Vis14].

Snap-to-Grid Die reguläre Verschiebungsaktion von Objekten im Diagramm funktioniert in der Regel wie folgt: Der Nutzer wählt mit dem Mauszeiger ein Objekt aus und schiebt das Objekt mit gedrückter Maustaste auf seine neue Position. Wenn die Funktion „Snap-to-Grid" nicht aktiviert ist, kann die Zielposition beliebig sein. Während der Verschiebung wird die Position des Objekts kontinuierlich angepasst, sodass der Nutzer ein visuelles Feedback bekommt. Dieser Vorgang ist in Abbildung 3.1 dargestellt.

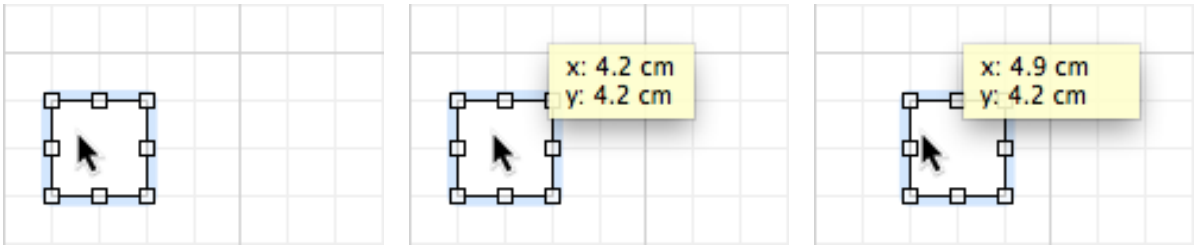
[11] `http://office.microsoft.com/en-us/powerpoint/`
[12] `http://www.apple.com/de/mac/keynote/`
[13] `http://www.visual-paradigm.com`



(a) (b) (c)

Abbildung 3.1.: Verschiebungsaktion mit nicht aktivierter Funktion „Snap-to-Grid“ in *OmniGraffle*

Wenn die Funktion „Snap-to-Grid“ nun aktiviert ist, wird im Unterschied zu dem vorherigen Vorgang die Zielposition so eingeschränkt, dass das verschobene Objekt an den Rasterlinien ausgerichtet wird. Der Mauszeiger kann dabei frei positioniert werden, das Objekt nimmt aber immer die nächste ausgerichtete Position an. Dieser Vorgang ist in Abbildung 3.2 dargestellt.



(a) (b) (c)

Abbildung 3.2.: Verschiebungsaktion mit aktivierter Funktion „Snap-to-Grid“ in *OmniGraffle*

Neben der Verschiebungsaktion wird auch die Aktion der Größenänderung eingeschränkt, indem das manipulierte Objekt nur eine solche Größe annehmen kann, die die Ausrichtung an die Rasterlinien nicht verletzt.

Ausrichten im Raster Das Ausrichten an Rasterlinien kann auch nachträglich durchgeführt werden. Dies passiert durch die Auswahl des auszurichtenden Objekts und dem Ausführen der Funktion *Ausrichten im Raster*. Die Position und Größe des Objekts werden so angepasst, dass das Objekt an den Rasterlinien ausgerichtet wird.

3.2.2.2. AUSRICHTEN UND VERTEILEN VON OBJEKTEN IN RELATION ZUEINANDER

Ähnlich wie das Raster (Abschnitt 3.2.2.1) werden die Funktionen zum Ausrichten und Verteilen von ausgewählten Objekten häufig in Visualisierungsprogrammen eingesetzt. Sie funktionieren wie folgt: Der Nutzer wählt im Diagramm Objekte aus, auf die die Funktion angewendet werden soll. Nachher wählt er eine konkrete Form der Funktion aus, die anschließend auf die ausgewählten Objekt angewendet wird [App11].

Ausrichten Die ausgewählten Objekte können in Relation zueinander ausgerichtet werden. Dafür muss nach der Auswahl von mindestens zwei Objekten die Form der Ausrichtung angegeben werden. Die Objekte können entweder vertikal (linke Kanten, Mittelachsen oder rechte Kanten) oder horizontal (obere Kanten, Mittelachsen oder untere Kanten) ausgerichtet werden. Nach dem Ausführen dieser Funktion werden die Positionen der ausgewählten Objekte so angepasst, dass die gewünschte Ausrichtung erfüllt ist [App11, Omn08]. Ein Beispiel ist in Abbildung 3.3 zu sehen.

(a) (b)

Abbildung 3.3.: Anwendung der horizontal zentrierten Ausrichtung in *Keynote*

Verteilen Wenn mindestens drei Objekte ausgewählt werden, kann die Funktion zum gleichmäßigen Verteilen der Objekte angewendet werden. Dabei wird die Position der äußeren Objekte fixiert und die Position von umschlossenen Objekten wird so angepasst, dass die vertikalen bzw. horizontalen Abstände zwischen den Objekten gleich sind (siehe Abbildung 3.4) [App11].

(a) (b)

Abbildung 3.4.: Anwendung der horizontalen Verteilung in *Keynote*

3.2.2.3. SMART GUIDES

Eine sehr hilfreiche Funktion, die in den meisten Grafik- und Präsentationsprogrammen unterstützt wird, sind Hilfslinien für Ausrichtung und relative Positionierung (engl.: „Smart Guides“) [App11]. Wenn diese Funktion aktiviert ist, werden während der Manipulation des ausgewählten Objekts farbige Linien eingeblendet. Diese Linien geben dem Nutzer ein visuelles Feedback zur geeigneten Ausrichtung des manipulierten Objekts relativ zu anderen Objekten im Diagramm. Außerdem wird ähnlich wie bei der Funktion *Snap-to-Grid* die freie Manipulation eingeschränkt, was in der Regel zu besseren Layout-Ergebnissen führt. Die Hilfslinien werden nach der Beendigung der Aktion wieder ausgeblendet und die Position bzw. Größe des manipulierten Objekts wird angepasst. Genauso wie auch bei allen anderen Hilfsfunktionen für das manuelle Layout handelt es sich bei den Hilfslinien um eine temporäre Layout-Funktion. Somit

wird die Ausrichtung der Objekte nur in deren Eigenschaften (Position und Größe) ausgedrückt und es wird keine explizite Regel erstellt.

Hilfslinien zur Ausrichtung (engl.: „Alignment Guides“) Die erste Art der Hilfslinien ist für die Ausrichtung des verschobenen Objekts an den Kanten bzw. Mittelachsen von anderen Objekten in Diagramm nützlich. Die Linien werden während der Verschiebungsaktion eingeblendet, wenn das verschobene Objekt an einem anderen ausgerichtet wird [App11].

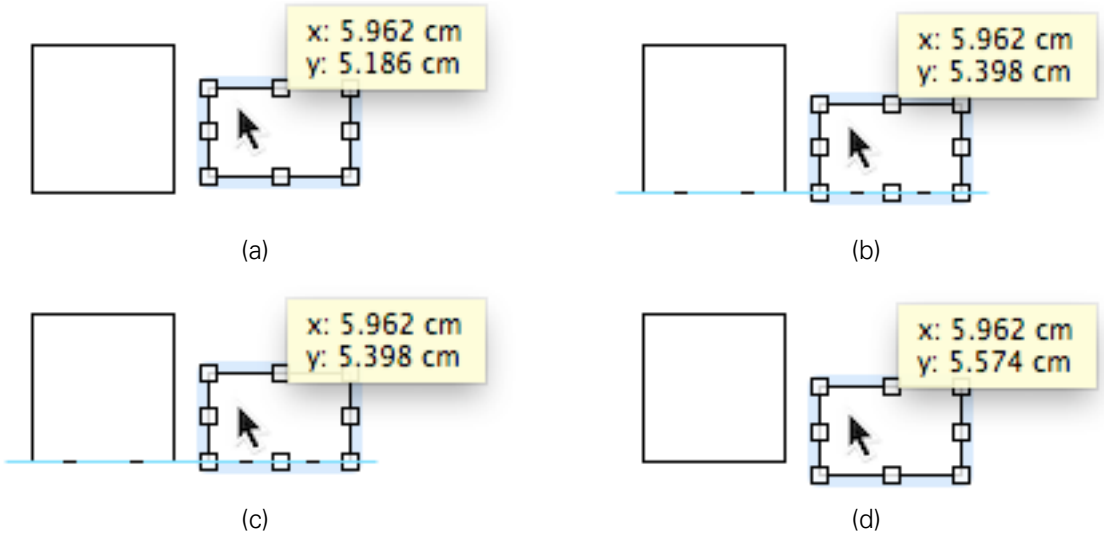


Abbildung 3.5.: Hilfslinien zur Ausrichtung während der Verschiebungsaktion in *OmniGraffle*

In Abbildung 3.5 wird ein Beispiel der Hilfslinien zur Ausrichtung der unteren Kanten von zwei Objekten während einer Verschiebungsaktion veranschaulicht. Zunächst wird in 3.5(a) das rechte Objekt ausgewählt und nach unten verschoben. Wenn sich die untere Kante des verschobenen Objekts der unteren Kante des inaktiven Objekts nähert, so wie es in 3.5(b) der Fall ist, wird eine Hilfslinie eingeblendet und das verschobene Objekt springt auf eine ausgerichtete Position. In 3.5(c) wird die Verschiebung weitergeführt. Dies kann anhand der Position des Mauszeigers erkannt werden. Das Objekt bleibt so lange ausgerichtet, bis der Abstand der Kanten eine bestimmte Schwelle in 3.5(d) überschreitet. Danach wird die Hilfslinie ausgeblendet und eine freie Positionierung ist wieder möglich.

Aus dem vorgestellten Beispiel folgt, dass sich um die untere Kante des inaktiven Objekts ein Bereich bildet, in dem das verschobene Objekt auf die ausgerichtete Position hinspringt. Dies passiert, wenn sich die Geraden in der Nähe befinden und der Abstand unter einer bestimmten Schwelle liegt. Dieser Bereich wird in Abbildung 3.6 visualisiert.

Solche Bereiche werden für alle Kanten und Mittelachsen von nicht ausgewählten Objekten in der horizontalen und vertikalen Richtung gebildet. Wenn der Nutzer ein Objekt verschiebt und sich eine der Kanten oder die Mittelachse des verschobenen Objekts in einem der Bereiche befindet, wird seine Position ausgerichtet, ohne den Mauszeiger zu beeinflussen. Das kann auch gleichzeitig für die horizontale und vertikale Richtung der Fall sein. Somit kann z.B. ein Objekt in einem anderen Objekt zentriert werden (siehe Abbildung 3.7).

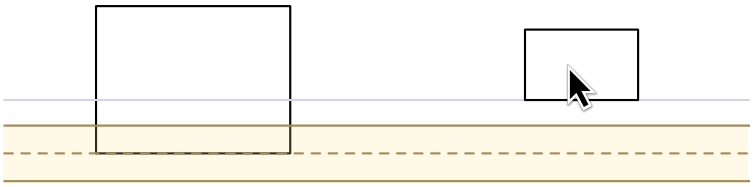


Abbildung 3.6.: Visualisierung des Bereichs für die Ausrichtung an die untere Kante des linken Objekts

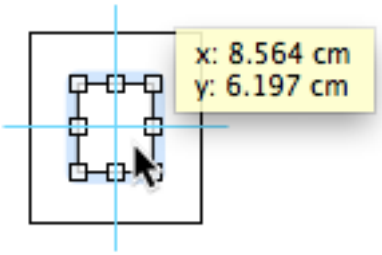


Abbildung 3.7.: Ausrichten eines Objekts in einem anderen mithilfe einer horizontalen und einer vertikalen Hilfslinie zur Ausrichtung in *OmniGraffle*

Eine Veranschaulichung dieser Funktion bietet in Form einer JavaScript-Anwendung das Open-Source-Projekt *Alignment-Guides*[14].

Abstandshilfslinien (engl.: „Distance Guides“) Die zweite Art der Hilfslinien dient dazu, die Abstände zwischen drei Objekten in einer horizontalen oder vertikalen Reihe gleich zu halten. Während der Verschiebung eines der drei Objekten wird ein Bereich gebildet, in dem das Objekt zu der Position hinspringt, in der die Abstände zwischen den nebeneinanderliegenden Objektpaaren gleich sind. Zusätzlich zu den Hilfslinien wird auch der Abstand eingeblendet [App11, Ols10]. Ein Beispiel dieser Funktion wird in Abbildung 3.8 gezeigt.

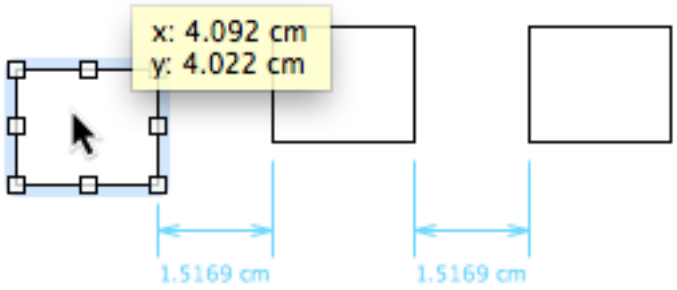


Abbildung 3.8.: Abstandshilfslinien während der Verschiebungsaktion in *OmniGraffle*

Größenhilfslinien (engl.: „Sizing Guides“) Im Unterschied zu den zwei vorher aufgeführten Arten der Hilfslinien wird die letzte Art durch die Aktion der Größenänderung eines Objekts

[14] Quellcode: `https://github.com/mrflix/Alignment-Guides`, Demo-Anwendung: `http://mrflix.github.io/Alignment-Guides`

hervorgerufen. Wenn sich die Größe des manipulierten Objekts der Größe eines anderen Objekts im Diagramm nähert und die Differenz unter einer bestimmten Grenze liegt, wird die Größe des anderen Objekts übernommen. Dies funktioniert getrennt für die Breite und Höhe (siehe Abbildung 3.9).

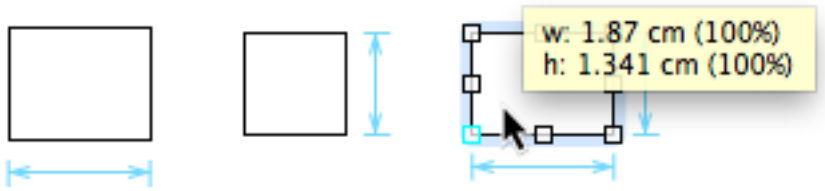


Abbildung 3.9.: Größenhilfslinien während der Aktion der Größenänderung in *OmniGraffle*

3.2.2.4. MANUELLE HILFSLINIEN

Weiterhin bieten *OmniGraffle* und *Keynote* die Funktion der Erstellung von manuellen Hilfslinien an [Omn08, App11]. Diese Funktion ist sehr ähnlich zu *Smart Guides* (siehe Abschnitt 3.2.2.3), unterscheidet sich aber wie folgt:

- Die Hilfslinien werden vom Nutzer manuell platziert - entweder horizontal oder vertikal.
- Die Hilfslinien sind global für das gesamte Diagramm und damit nicht relativ zu einem anderen Objekt im Diagramm.
- Die Hilfslinien sind permanent sichtbar, auch wenn keine Verschiebungsaktion stattfindet.[15]

Die Gemeinsamkeit besteht darin, dass das Ausrichten an die Hilfslinien identisch wie bei den *Smart Guides* funktioniert. Ein Bespiel der manuellen Hilfslinie in *OmniGraffle* wird in Abbildung 3.10 dargestellt.

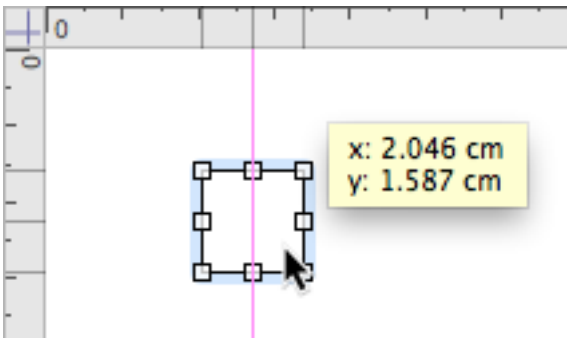


Abbildung 3.10.: Beispiel einer manuellen Hilfslinie in *OmniGraffle*

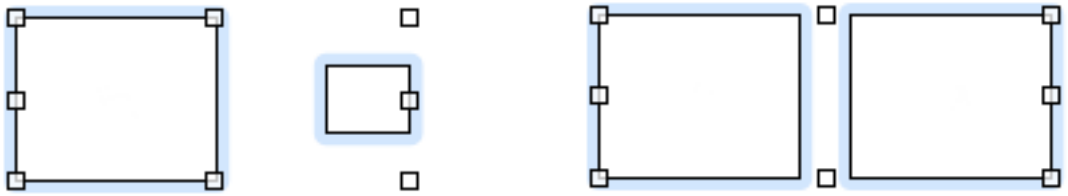
Wie bereits erwähnt, ist das Ausrichten an die manuellen Hilfslinien nicht persistent und eine Verschiebung der Hilfslinie verursacht keine Verschiebung der ausgerichteten Objekte. Dieses

[15] Die Anzeige der manuellen Hilfslinien kann sowohl in *OmniGraffle* als auch in *Keynote* global aus- und eingeschaltet werden.

Verhalten wird allerdings in den Tools *Microsoft Visio*[16] und *ConceptDraw*[17] unterstützt und wird mithilfe von Einweg-Constraints realisiert [Mai12, S.20].

3.2.2.5. GLEICHE GRÖSSE DER OBJEKTE

Sehr trivial aber dennoch nützlich ist die Funktion der Einstellung von gleichen Größen für ausgewählte Objekte. Nach der Auswahl von zwei oder mehreren Objekten im Diagramm hat der Nutzer die Möglichkeit, die Breite, die Höhe oder beide Dimensionen gleichzeitig auf die Werte des zuerst ausgewählten Objekts zu setzen (siehe Abbildung 3.11). Diese Funktion wird in *OmniGraffle* unterstützt [Ols10].

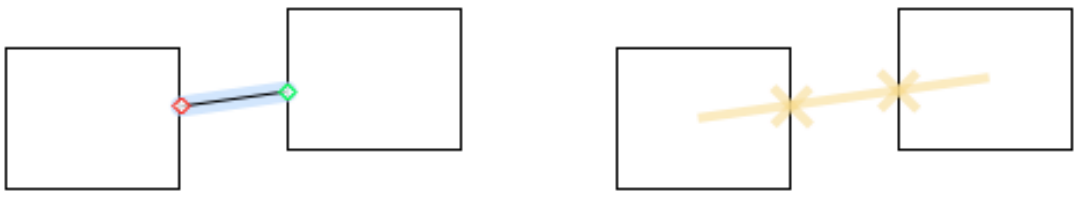


(a) (b)

Abbildung 3.11.: Anwendung der Funktion zur Einstellung von gleichen Größen in *OmniGraffle*

3.2.2.6. VERKNÜPFUNGSPUNKTE FÜR VERBINDUNGEN

Die bisher diskutierten Hilfsfunktionen haben sich ausschließlich mit der Positionierung und der Größenbestimmung der Objekte auseinandergesetzt. Für die graphbasierten Diagramme (siehe Abschnitt 2.1.1) sind neben den Knoten auch Kanten von Interesse, die in der Regel durch Verbindungslinien dargestellt werden. Wenn zwei Objekte mit einer Verbindungslinie verbunden werden, bleiben sie auch dann verbunden, wenn sich die Position oder Größe der Objekte ändert [App11]. In vielen Programmen (unter anderem in *Keynote* und *OmniGraffle*) werden die Verknüpfungspunkte der Verbindungslinie standardmäßig an den Stellen positioniert, an denen eine gedachte Strecke, die die Mittelpunkte beider Objekte verbindet, die Kanten der Objekte schneidet (siehe Abbildung 3.12) [Omn08].



(a) (b)

Abbildung 3.12.: Verknüpfungspunkte einer Verbindung in *OmniGraffle* (a) mit Veranschaulichung der Berechnung (b)

[16] http://visio.microsoft.com
[17] http://conceptdraw.com

OmniGraffle bietet die Möglichkeit, mithilfe des Magnetwerkzeugs manuell die Verknüpfungspunkte zu definieren. Dies funktioniert auf zwei verschiedene Arten. Entweder kann der Nutzer die Verknüpfungspunkte beliebig in dem Objekt positionieren oder er wählt eine vordefinierte Anordnung aus. Dazu gehört u.a. die gleichmäßige Verteilung einer festen Anzahl an Verknüpfungspunkten entlang aller Kanten oder die Positionierung eines Verknüpfungspunkts an jedem Eckpunkt. Ein Beispiel der Darstellung von Verknüpfungspunkten ist in Abbildung 3.13 zu sehen.

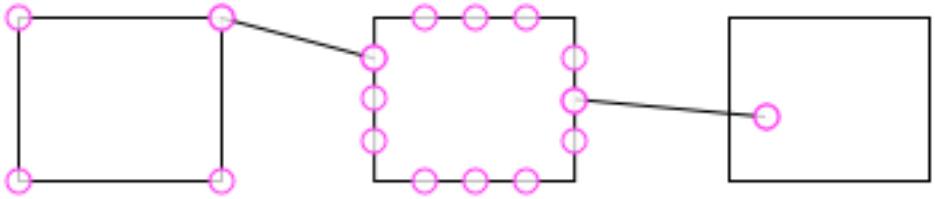


Abbildung 3.13.: Beispiel der manuellen Verknüpfungspunkte in *OmniGraffle*: Magnete an den Eckpunkten (links), drei Magnete pro Kante (Mitte), ein manuell positionierter Magnet (rechts)

Sobald mindestens ein Magnet dem Objekt hinzugefügt wurde, wird nicht mehr der Verknüpfungspunkt mithilfe des Mittelpunkts wie oben beschrieben berechnet, sondern die Verbindungslinie wird immer von dem nächsten Verknüpfungspunkt angezogen (siehe Abbildung 3.13). Alternativ kann der Nutzer ein Ende der Verbindungslinie an einen konkreten Verknüpfungspunkt binden. In dem Fall bleibt die Verbindungslinie an den gewählten Verknüpfungspunkt gebunden, auch wenn ein anderer Verknüpfungspunkt näher ist.

3.2.3. ZUSAMMENFASSUNG DER EIGENSCHAFTEN

3.2.3.1. VORTEILE

- Unterstützung der interaktiven Bearbeitung Das manuelle Layout ermöglicht freie Bearbeitung der Elemente im Diagramm. Insbesondere können deren Layout-Eigenschaften manuell angepasst werden, wofür zahlreiche Hilfsfunktionen eingesetzt werden können. Dadurch wird die Interaktion des Nutzers mit dem Diagramm in den Vordergrund gestellt.
- Hohe Flexibilität Aufgrund der Möglichkeit der beliebigen Anpassung der Layout-Eigenschaften kann das manuelle Layout sehr präzise gesteuert werden. Somit hat der Nutzer einen großen Einfluss auf das resultierende Layout.

3.2.3.2. NACHTEILE

- Aufwand der Layout-Erstellung Die Aktionen zur Erstellung des Layouts sind temporär, d.h. nach jeder Änderung des Inhalts eines Diagramms muss das erstellte Layout neu angepasst werden, was in der Regel mit viel Aufwand verbunden ist.

- **Fehlende Unterstützung der Diagrammtypen** Die Tools, die das manuelle Layout einsetzen, lassen sich in zwei Gruppen unterteilen: CASE-Tools, die die grafische Notation der Diagrammtypen unterstützen und Vektorgrafik-Tools, die eine geringe bis keine Unterstützung der grafischen Notation bringen. Die Syntax- und Semantikregeln für die Layout-Erstellung werden jedoch weder in den CASE-Tools noch in den Vektorgrafik-Tools berücksichtigt und können durch den Nutzer verletzt werden.

3.3. AUTOMATISCHES LAYOUT

Den graphbasierten Softwarediagrammen unterliegt die abstrakte Struktur des Graphen (siehe Abschnitt 2.1.1). Mit der grafischen Darstellung von Graphen beschäftigt sich die mathematische Disziplin des **Graphzeichnens**. Ihre Aufgabe besteht darin, Layout-Algorithmen zu entwerfen, die optimale Layouts in Hinsicht auf ästhetische Prinzipien (siehe Abschnitt 2.2) erzeugen, d.h. die Layout-Eigenschaften wie Positionen der Knoten und Routen der Kanten berechnen [Eic05, Arv02, Sie03, Mai12]. Unter dem automatischen Layout für graphbasierte Diagramme sind daher vollautomatische Algorithmen zu verstehen, die für ein gegebenes Diagramm die optimalen Layout-Eigenschaften berechnen und das Diagramm entsprechend anpassen [Fuh11].

Um während der Erstellung des Diagramms immer ein optimales Layout zu erhalten, muss der Layout-Algorithmus kontinuierlich nach jeder Änderung des Inhalts entweder automatisch oder manuell aufgerufen werden. Wenn eine Interaktion mit dem resultierenden Diagramm unterstützt wird, hat sie keinerlei Einwirkung auf den Algorithmus und wird bei dem nächsten Aufruf nicht berücksichtigt. In der Regel kann der Nutzer nur bestimmte Parameter des Algorithmus[18] anpassen und hat somit nur einen geringen Einfluss auf das Ergebnis des Layout-Prozesses.

Die Ansätze lassen sich in zwei Kategorien nach der Art der Sprache unterteilen, die als Eingabe für den Layout-Algorithmus verwendet wird[19]. Zum einen gibt es Ansätze, die aus einer Beschreibung des Diagramms in einer **textuellen Sprache** unter Anwendung des Layout-Algorithmus eine grafische Repräsentation erzeugen. Zum anderen gibt es Ansätze, die auf Diagramme angewendet werden, die in einer **visuellen Sprache** modelliert sind. Diese Ansätze verändern die Layout-Eigenschaften der Diagrammelemente direkt.

Mit den textbasierten Ansätzen beschäftigt sich Abschnitt 3.3.1. Die visuellen Ansätze werden in Abschnitt 3.3.2 behandelt. Den speziellen Algorithmen für Klassendiagramme widmet sich Abschnitt 3.3.3. Abschließend werden die Eigenschaften der Ansätze für das automatische Layout in Abschnitt 3.3.4 zusammengefasst.

[18] z.B. minimale Länge der Kanten im zirkulären Layout-Algorithmus `circo` aus der Bibliothek *Graphviz* [NGEH14]

[19] In der Literatur werden Ansätze für das automatische Layout meistens nach den Layout-Algorithmen kategorisiert [Fuh11, Eic05]. Wie bereits in Abschnitt 3.1 erläutert wurde, richtet sich die Kategorisierung in dieser Arbeit danach, wie die Ansätze zu bedienen sind und welche Art der Interaktion sie unterstützen.

3.3.1. TEXTBASIERTE ANSÄTZE

Die textbasierten Ansätze für das automatische Layout erfordern als Eingabe eine **textuelle Beschreibung** des Diagramms, die in einer allgemeinen Auszeichnungssprache oder einer domänenspezifischen Sprache formuliert werden kann. Diese Beschreibung wird eingelesen und intern in ein abstraktes Modell umgewandelt, auf das der Layout-Algorithmus angewendet wird. Als Ausgabe wird eine statische Repräsentation des Diagramms in Form eines Bildes geliefert, bei der jegliche Möglichkeit der Interaktion fehlt. Durch die Entkopplung der Eingabe von der Ausgabe lässt sich das Diagramm nur durch eine Änderung des Quelltexts und einen wiederholten Aufruf des Layout-Algorithmus verändern.

3.3.1.1. GRAPHZEICHNEN-TOOLS

Es existiert eine Menge an Bibliotheken, die verschiedene automatische Algorithmen für das Graphzeichnen implementieren, u.a. handelt es sich um hierarchische, kräftebasierte und orthogonale Algorithmen [Mai12]. Sie stellen die Funktionalität der automatischen Layout-Berechnung für Graphen bereit und können durch andere Programme verwendet werden. Beispiele für solche Bibliotheken sind *Graphviz*[20], *yFiles*[21] oder *Kieler*[22] [Mai12]. In dieser Arbeit wird *Graphviz* näher vorgestellt. Insbesondere werden seine Aspekte des textbasierten und visuellen Ansatzes gezeigt.

Graphviz ist ein in der Programmiersprache C geschriebenes Tool für die Visualisierung von Graphen und wurde ursprünglich von *AT&T* entwickelt. Es beinhaltet folgende Komponenten:

- die Domänenspezifische **Sprache Dot**[23] für die Beschreibung von Graphen
- einen Satz von **Layout-Algorithmen**: u.a. *dot*, *neato*, *fdp* oder *circo* [Gan14, NGEH14]
- eine **Software-Bibliothek**, die die Funktionalität der Layout-Algorithmen und der grafischen Ausgabe bereitstellt und sich in andere Programme einbinden lässt [Gan14]
- einen Satz von **Kommandozeilen-Tools** für die Anwendung der Layout-Algorithmen und für die grafische Ausgabe [NGEH14]
- eine **GUI-Anwendung**, die über dieselben Funktionen wie die Kommandozeilen-Tools verfügt

Um einen Layout-Algorithmus auf einen Graphen mithilfe der Komandozeilen-Tools oder der GUI-Anwendung anwenden zu können, muss der Graph in der Sprache Dot beschrieben werden. Ein Beispiel eines einfachen Graphen mit vier Knoten und drei Kanten ist im Quelltext 3.1 gegeben.

[20] `http://graphviz.org`
[21] `http://www.yworks.com/en/products_yfiles_about.html`
[22] `http://www.informatik.uni-kiel.de/en/rtsys/kieler/`
[23] Der Aufbau der Sprache wird unter `http://www.graphviz.org/content/dot-language` erläutert. Ein Beispiel der Beschreibung eines Graphen ist im Quelltext 3.1 gegeben.

```
graph DotExampleGraph {
    rankdir = LR
    A -- C
    B -- C
    C -- D
}
```

Quelltext 3.1: Beschreibung eines Graphen in Dot (`graphviz-dot-example.dot`)

Durch einen Aufruf des Kommandozeilen-Tools *dot* wird die oben aufgelistete Dot-Quelldatei eingelesen und der beschriebene Graph wird in Form eines internen Modells instanziiert, worauf der Layout-Algorithmus *dot* angewendet wird. Anschließend wird mit einem „Renderer" die graphische Repräsentation in einer Datei erzeugt. Dies wird in Abbildung 3.14 dargestellt [Gan14].

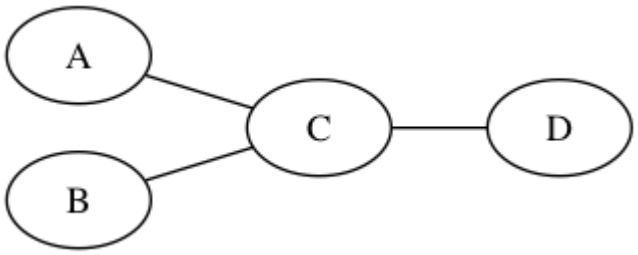


Abbildung 3.14.: Das Resultat des Aufrufs des Kommandozeilen-Tools *dot*

Alternativ kann die Dot-Quelldatei in der GUI-Anwendung geöffnet werden. In dem Fall wird das Ergebnis nicht direkt in eine Datei gespeichert, sondern als PDF in einem Fenster der Anwendung angezeigt.

Das oben diskutierte Beispiel zeigt, dass das resultierende Diagramm nicht interaktiv ist. Eine Änderung des Graphen ist nur in der Quelldatei möglich und mit einem wiederholten Aufruf des Kommandozeilen-Tools oder Laden der Datei in der GUI-Anwendung verbunden. Diese Schwachstelle wird durch 3rd-Party Editoren mit der integrierten grafischen Ausgabe wie etwa *Leonhard*[24] oder *WebGraphviz*[25] verbessert. In *Leonhard* wird die Übersetzung nach jeder Änderung des Quelltexts sogar automatisch gestartet.

Trotz der fehlenden Interaktivität des resultierenden Diagramms kann das Layout von dem Nutzer beeinflusst werden, indem der Layout-Algorithmus gewählt wird bzw. seine Parameter in der Quelldatei angepasst werden [NGEH14]. Im oben genannten Beispiel wurde der Layout-Algorithmus *dot* verwendet und die Richtung des Graphen in der Dot-Quelldatei mit dem Befehl `rankdir = LR` angepasst, sodass der Graph von links nach rechts gezeichnet wird. Die konkreten in *Graphviz* unterstützten Layout-Algorithmen werden in [Gan14, S.22] beschrieben und in Abschnitt 3.3.2.1 zum Teil visualisiert.

[24] *Leonhard* ist ein grafischer Editor für *Graphviz*, der unter älteren Versionen von *Mac OS X* funktioniert. Weitere Information sind unter `http://algorithmique.net/leonhard.html` und `https://github.com/glejeune/Leonhard` zu finden.

[25] `http://webgraphviz.com`

3.3.1.2. TEXTBASIERTE UML-TOOLS

Neben den textbasierten Tools für das Graphzeichnen gibt es Tools, die für spezifische Domänen ausgelegt sind. An dieser Stelle werden kurz zwei textbasierte UML-Tools vorgestellt, die aus einer textuellen Beschreibung in einer speziellen Sprache grafische UML-Diagramme erzeugen und intern für die Layout-Berechnung die oben vorgestellte Bibliothek *Graphviz* verwenden:

- PlantUML[26] ist eine Java-Bibliothek, die für die Beschreibung von UML-Diagrammen die gleichnamige Sprache verwendet. Diese Sprache wird in [Roq10] näher behandelt. Neben den vielen Anwendungen[27] bietet *PlantUML* einen Online-Editor[28] an, der es ermöglicht, UML-Diagramme direkt im Browser zu erstellen und in Form von Bildern zu exportieren.
- yUML[29] ist ein Online-Editor zur Erstellung von UML-Diagrammen im Browser. Im Unterschied zu *PlantUML* verwendet *yUML* eine anschauliche zeichenbasierte Sprache[30]. Die Eingabe erfolgt über ein Textfeld und wird kontinuierlich in ein Bild übersetzt, das jederzeit exportiert werden kann. Alternativ kann eine URL generiert [Fuh11] werden, unter der das gezeichnete Diagramm online verfügbar ist.

3.3.2. VISUELLE ANSÄTZE

Die visuellen Ansätze für das automatische Layout unterscheiden sich von den textbasierten darin, dass der Layout-Algorithmus auf eine visuelle Sprache angewendet wird und dass das berechnete Layout direkt die Eingabe verändert. Diese Ansätze werden in der Regel in visuellen Editoren eingesetzt, die eine unmittelbare Bearbeitung des Diagramms unterstützen.

Der Ablauf ist wie folgt: Zunächst wird ein Diagramm in einer bestimmten visuellen Sprache modelliert (z.B. ein Klassendiagramm in der Sprache UML). Danach wird ein Layout-Algorithmus (in der Regel manuell) ausgeführt, der das neue Layout für alle Diagrammbestandteile berechnet. Anschließend wird das modellierte Diagramm dermaßen angepasst, sodass es das berechnete Layout annimmt.

Die Entkoppelung der Ein- und Ausgabe, wodurch sich die textuellen Ansätze auszeichnen (siehe Abschnitt 3.3.1), entfällt an dieser Stelle und da das resultierende Diagramm interaktiv bleibt, kann es durch den Nutzer weiterhin verändert bzw. erweitert werden. Um nach jeder Änderung des Diagramms ein automatisch berechnetes Layout zu erhalten, muss allerdings der Layout-Algorithmus jedes Mal erneut gestartet werden.

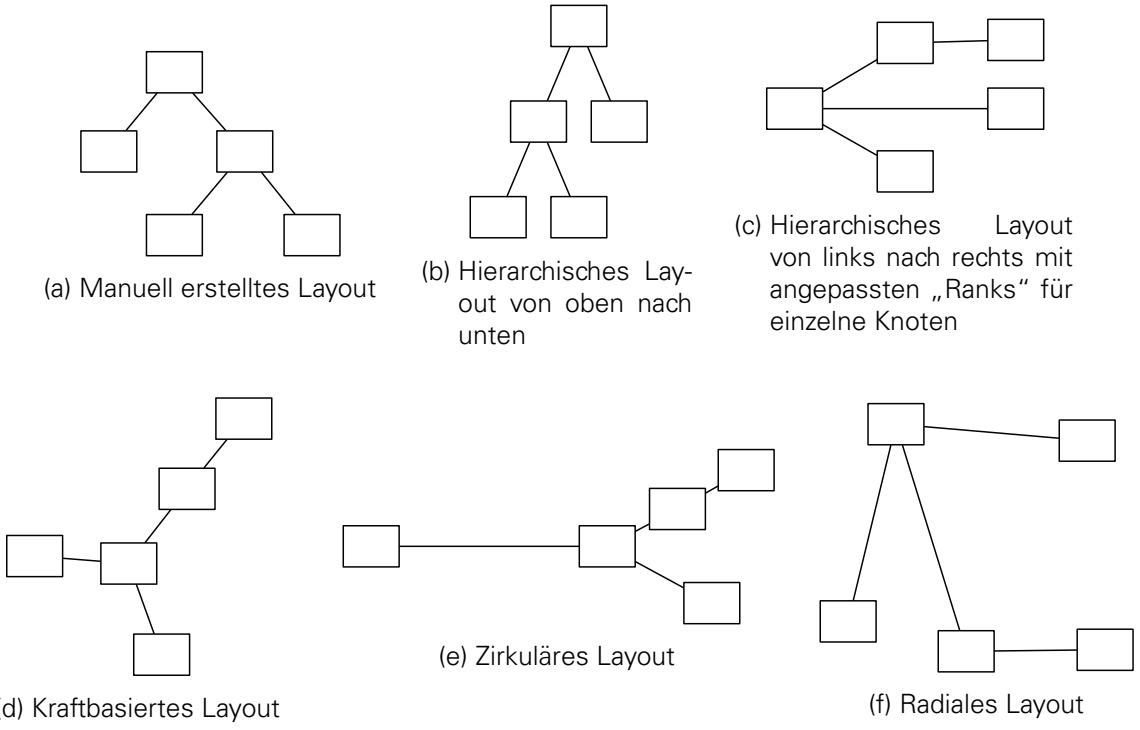
Die Algorithmen für das automatische Layout sind im Allgemeinen nicht ideal und die Nutzer neigen dazu, das berechnete Layout für persönliche Präferenzen anzupassen, um ein mentales

[26] `http://plantuml.sourceforge.net`
[27] Die bekannten Anwendungen sind unter `http://plantuml.sourceforge.net/running.html` aufgelistet.
[28] *PlantUML* Server: `http://www.plantuml.com/plantuml`
[29] `http://yuml.me`
[30] Eine Übersicht der Syntax für Klassendiagramme: `http://yuml.me/diagram/scruffy/class/samples`

Modell (siehe Abschnitt 2.1.3) bzw. eine sekundäre Notation (siehe Abschnitt 2.1.2) zu verwalten. Alle nachträglich manuell getätigten Layout-Änderungen werden bei einem erneuten Aufruf des Layout-Algorithmus verworfen und somit steht der Nutzer vor der Entscheidung, ob er nach jedem Aufruf des Algorithmus die Anpassungen wiederholt durchführt oder auf das automatische Layout komplett verzichtet [Eig04, S.119ff].

3.3.2.1. AUTOMATISCHES LAYOUT IN OMNIGRAFFLE

Wie bereits in Abschnitt 3.3.1.1 beschrieben wurde, bietet *Graphviz* eine Software-Bibliothek an, die sich in andere Programme einbinden lässt. Das Visualisierungsprogramm *OmniGraffle* (siehe Abschnitt 3.2.1.1) macht sich dies zunutze und verfügt über eine Funktion für das automatische Layout, die für die Layout-Berechnung Algorithmen aus *Graphviz* verwendet, nämlich den hierarchischen, kraftbasierten, zirkulären und radialen Algorithmus [Ols10]. Nachdem die Funktion des automatischen Layouts für ein Diagramm aktiviert wird, kann der Layout-Algorithmus ausgewählt und seine Parameter eingestellt werden[31]. In Abbildung 3.15 wird ein Beispiel der Anwendung von allen verfügbaren Layout-Algorithmen auf ein Diagramm illustriert.



(a) Manuell erstelltes Layout

(b) Hierarchisches Layout von oben nach unten

(c) Hierarchisches Layout von links nach rechts mit angepassten „Ranks“ für einzelne Knoten

(d) Kraftbasiertes Layout

(e) Zirkuläres Layout

(f) Radiales Layout

Abbildung 3.15.: Beispiele der Anwendung von automatischen Layout-Algorithmen in OmniGraffle

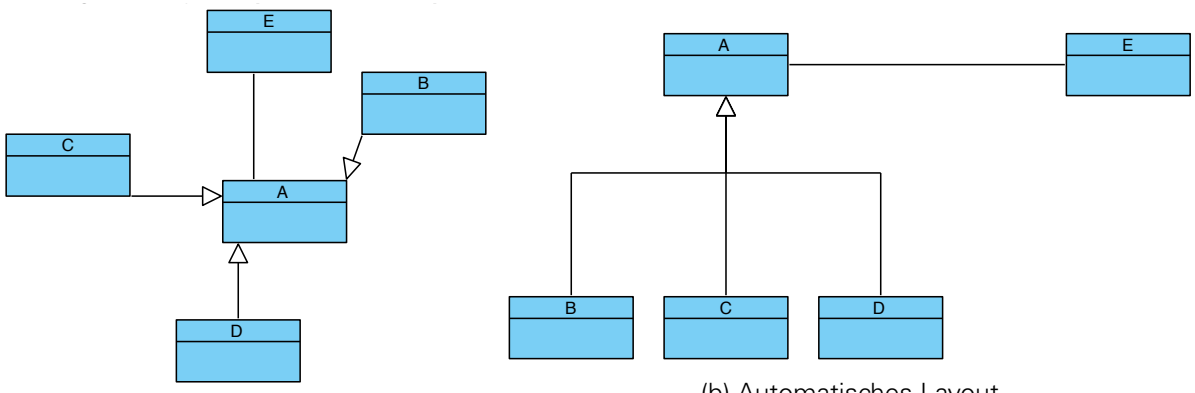
Der eingestellte Layout-Algorithmus kann entweder durch die manuelle Auswahl im Menü, die Veränderung der Parameter des Algorithmus oder das Hinzufügen sowie Löschen einer Kante zwischen zwei Knoten erneut aufgerufen werden [Wyb08, S.43]. Danach wird das Layout berechnet und mithilfe einer Animation auf das Diagramm angewendet, wobei die durchge-

[31] Die konkreten möglichen Parameter der Algorithmen sind in [Omn08, S.74] nachzulesen.

führten manuellen Anpassungen verworfen werden. *OmniGraffle* erweitert die Funktion des automatischen Layouts um weitere Tools. So kann z.B. die Struktur des Diagramms parallel in einer Outline bearbeitet werden [Omn08].

3.3.2.2. AUTOMATISCHES LAYOUT IN VISUAL PARADIGM

Ähnlich wie *OmniGraffle* basiert das CASE-Tool *Visual Paradigm* (siehe Abschnitt 3.2.1.3) auf der freien Positionierung der Elemente im Diagramm und verfügt über die Funktion der automatischen Layout-Berechnung [Vis14]. Dem Nutzer steht eine Vielzahl von Layout-Algorithmen zur Verfügung, deren Eigenschaften wunschgemäß eingestellt werden können [Fuh11]. Im Unterschied zu der Aktivierung der automatischen Layout-Berechnung für das gesamte Diagramm in *OmniGraffle* (siehe Abschnitt 3.3.2.1) ist die Berechnung in *Visual Paradigm* einmalig und kann auch auf eine Untermenge des Diagramms angewendet werden. Sie wird manuell durch eine Auswahl im Kontextmenü gestartet, wobei der konkrete Layout-Algorithmus entweder manuell oder automatisch gewählt werden kann. Aus diesem Grund versagt diese Funktion im Bereich der Interaktivität. Eine weitere Schwachstelle dieser Funktion ist die mangelhafte bis fehlende Unterstützung der Semantik der Elemente. Ein Beispiel der Anwendung der automatischen Layout-Berechnung auf ein einfaches Klassendiagramm ist in Abbildung 3.16 dargestellt.



(a) Manuelles Layout

(b) Automatisches Layout

Abbildung 3.16.: Ein Beispiel der automatischen Layout-Berechnung für ein einfaches Klassendiagramm in *Visual Paradigm*

3.3.3. SPEZIELLE ALGORITHMEN FÜR KLASSENDIAGRAMME

Die Berechnung des automatischen Layouts von Klassendiagrammen bildet den Gegenstand für viele Forschungsarbeiten wie z.B. [Eic05], [Sie03] sowie [Eig04]. Die darin vorgestellten Algorithmen haben den gleichen statischen Charakter wie die Algorithmen zum Graphzeichnen, sind allerdings für Klassendiagramme spezialisiert, indem deren Syntax und unter Umständen auch Semantik bei der Layout-Berechnung berücksichtigt werden. Eine grundlegende Rolle spielen für die automatischen Layout-Algorithmen die ästhetischen Prinzipien [Mai12]. Da die

ästhetischen Prinzipien für Graphen an dieser Stelle nicht ausreichend sind [Eic05, S.79], werden sie in den aufgeführten Arbeiten für Klassendiagramme erweitert. Ihre Zusammenfassung wurde in Abschnitt 2.2.2 aufgeführt.

Die genannten Arbeiten machen sich bestehende Algorithmen zum Graphzeichnen zunutze, passen diese für die Klassendiagramme an und setzen sie in der Regel in Form einer Bibliothek um. Zu den verfügbaren algorithmischen Ansätzen gehören Topology-Shape-Metrics [Sie03, S.33], hierarchische Algorithmen und kraftbasierte Algorithmen [Eic05, S.32ff]. Die Grundlage für [Eic05] bildet der hierarchische Sugiyama Algorithmus. Dagegen basieren [Sie03] und [Eig04] auf dem Topology-Shape-Metrics Ansatz, der den Prozess der automatischen Layout-Berechnung in mehrere Schritte unterteilt.

Obwohl sich die Algorithmen in den erwähnten Arbeiten intern unterschiedlich verhalten, besteht ihre Funktion in der automatischen Berechnung der Layout-Eigenschaften für die Elemente eines Klassendiagramms, welches in der Regel anhand einer Instanz des UML-Metamodells beschrieben wird. Einerseits lassen sie sich in visuelle Editoren einbauen und können somit den visuellen Ansätzen für das automatische Layout (siehe Abschnitt 3.3.2) zugeordnet werden. Anderseits kann das Modell textuell in Form einer XMI-Datei[32] repräsentiert werden. Daher ist die Einordnung in die textuellen Ansätze für das automatische Layout (siehe Abschnitt 3.3.1) ebenfalls möglich.

Trotz der in [Eig04] beschriebenen möglichen Erweiterung des Algorithmus für einen interaktiven Einsatz sind die in allen oben genannten Arbeiten vorgestellten Algorithmen prinzipiell nicht für einen interaktiven Einsatz konzipiert und fokussieren sich eindeutig auf die Umsetzung der ästhetischen Prinzipien. Insbesondere sind sie für Szenarien geeignet, in denen das Klassendiagramm nicht durch den Nutzer erstellt wird, sondern als ein Modell vorliegt, wofür das Layout berechnet werden soll. Beispiele für solche Anwendungsfälle sind Dokumentationswerkzeuge (z.B. *Doxygen*[33]), Werkzeuge zur Generierung von Diagrammen oder Analyse-Werkzeuge zum „Reverse Engineering“ [Eig04].

3.3.4. ZUSAMMENFASSUNG DER EIGENSCHAFTEN

3.3.4.1. VORTEILE

- Automatische Layout-Berechnung Das automatische Layout wird, wie bereits der Name sagt, automatisch berechnet und verringert somit den Aufwand an manueller Layout-Erstellung durch den Nutzer.
- Einhaltung der ästhetischen Prinzipien Die Algorithmen für das automatische Layout von Diagrammen liefern in der Regel optimale Layouts im Bezug auf die ästhetischen Prinzipien [Mai12]. Durch die Spezialisierung der Algorithmen kann sogar das Einhalten der syntaktischen Strukturregeln der konkreten Diagrammtypen erreicht werden.

[32] XML Metadata Interchange (`http://www.omg.org/spec/XMI`)
[33] `http://www.stack.nl/~dimitri/doxygen/manual/diagrams.html`

- **Berechnung des initialen Layouts** Des Weiteren sind die Algorithmen für das automatische Layout von Diagrammen für die Berechnung des initialen Layouts ausgelegt. Dies ist insbesondere in den Anwendungsfällen hilfreich, in denen nur der Inhalt aber kein Layout des Diagramms vorliegen.
- **Möglichkeit der Einbindung in automatisierte Prozesse** Aufgrund des statischen Charakters, der definierten Eingabesprachen und der Eignung für die Berechnung des initialen Layouts sind die Ansätze für das automatische Layout für die Einbindung in automatisierte Prozesse geeignet.

3.3.4.2. NACHTEILE

- **Fehlende Interaktivität** Das automatische Layout ist für eine interaktive Bearbeitung der Diagramme nicht konzipiert, weil die direkte Interaktion des Nutzers mit dem Diagramm mangelhaft ist oder gar nicht unterstützt wird. Obwohl sich die visuellen Ansätze in interaktiven Umgebungen wie etwa visuellen Editoren einsetzen lassen, sind sie dafür nicht geeignet [Mai12, S.22ff] [DMW08, S.4]. Insbesondere zeichnet sich dies dadurch aus, dass der Prozess der Erstellung eines Diagramms nicht gefördert wird. Die interaktiven Änderungen des Inhalts bzw. des Layouts des Diagramms werden von dem Layout-Algorithmus nicht berücksichtigt und somit kann bei dem Aufruf des Algorithmus das mentale Modell (siehe Abschnitt 2.1.3) bzw. die sekundäre Notation (siehe Abschnitt 2.1.2) zerstört werden [Eig04]. Des Weiteren ist es notwendig, den Algorithmus manuell nach jeder Änderung erneut zu starten. Im Unterschied zu den visuellen Ansätzen bieten die textuellen Ansätze keine direkte Form der Interaktion an, was durch die unterschiedlichen Eingabe- und Ausgabesprachen und der damit zusammenhängenden Entkoppelung der Eingabe und Ausgabe bedingt ist.
- **Geringer Einfluss auf das resultierende Layout** Da das automatische Layout auf statischen Algorithmen basiert, die sich in der Regel nur durch die Anpassung der Parameter beeinflussen lassen, werden die Layout-Präferenzen des Nutzers nicht berücksichtigt. Die mögliche Kontrolle des Ergebnisses des Layout-Prozesses ist somit sehr mangelhaft [GSE+14].
- **Mangelhafte Berücksichtigung der Syntax- und Semantikregeln** Die syntaktischen und semantischen Layout-Regeln der konkreten Diagrammtypen werden von den allgemeinen Algorithmen für das Layout von graphbasierten Diagrammen nicht berücksichtigt und können dadurch verletzt werden. Diese Schwachstelle wird durch die Anpassung der Algorithmen für konkrete Diagrammtypen beseitigt.

3.4. INTERAKTIVES HALBAUTOMATISCHES LAYOUT

Wie bereits in diesem Kapitel präsentiert wurde, unterstützen die meisten Editoren zur Erstellung von Diagrammen Hilfsfunktionen für das manuelle Layout (siehe Abschnitt 3.2) und inte-

grieren eventuell zusätzlich auch automatische Layout-Algorithmen (siehe Abschnitt 3.3). Das manuelle Layout ist sehr intuitiv, zeichnet sich durch die direkte Interaktion des Nutzers mit dem Diagramm aus, lässt aber eine automatisierte Layout-Berechnung und Berücksichtigung der ästhetischen Prinzipien vermissen. Dieses Problem wird durch das automatische Layout angegangen, was aber im Bereich der Interaktivität versagt [GSE+14]. Die Vorteile der Ansätze aus beiden genannten Kategorien lassen sich kombinieren und bilden eine neue Kategorie der Ansätze für die halbautomatische Layout-Unterstützung.

Diese Ansätze sind für interaktive Umgebungen ausgelegt, die eine sequenzielle Modifizierung des Diagramms durch den Nutzer unterstützen [Arv02, GSE+14, Wyb08]. Das Layout wird während der Bearbeitung des Diagramms mithilfe von dynamischen Algorithmen kontinuierlich berechnet und inkrementell angepasst. Neben dem Inhalt kann der Nutzer auch einige Aspekte des Layouts beeinflussen, z.B. durch direkte Positionierung von Knoten bzw. Kanten oder durch eine andere Form von Feedback [Arv02].

3.4.1. STRUKTURBASIERTE BENUTZERGESTEUERTE ANSÄTZE

Die erste Kategorie der Ansätze für das interaktive halbautomatische Layout bilden die strukturbasierten benutzergesteuerten Ansätze, die dem Nutzer ermöglichen, das Layout des Diagramms durch Erstellung und Verwaltung von Strukturregeln[34] zu beeinflussen. Diese Strukturregeln erinnern an die Hilfsfunktionen für das manuelle Layout (siehe Abschnitt 3.2.2), sind aber dahingegen persistent [Wyb08]. Sie werden bei der Berechnung des Layouts durch einen dynamischen Layout-Algorithmus berücksichtigt und eingehalten.

Die strukturbasierten benutzergesteuerten Ansätze lassen sich in zwei Gruppen unterteilen: in Constraint-basierte und Pattern-basierte Ansätze. Beide Gruppen werden im Folgenden vorgestellt.

3.4.1.1. CONSTRAINT-BASIERTE ANSÄTZE

Die Beschreibung der Strukturregeln kann mithilfe von **Constraints** erfolgen. Dies macht sich der in [Wyb08] beschriebene Ansatz zunutze und stellt einen Algorithmus für die kontinuierliche Layout-Berechnung in einem interaktiven Editor anhand von Constraints vor. Die Grundidee basiert darauf, dass der Nutzer persistente Constraints für die Beziehungen von ausgewählten Knoten erstellt (z.B. Verteilung oder Ausrichtung der Knoten) und somit deren Layout beschreibt. Die Einhaltung der Constraints wird durch einen **Constraintlöser**[35] gewährleistet, der das valide Layout anhand der Zusammensetzung von allen erstellten Constraints berechnet.

Die freie Positionierung der Elemente im Diagramm ist grundsätzlich möglich, wird aber durch die erstellten Constraints eingeschränkt. Das zeichnet sich z.B. dadurch aus, dass bei der Bewegung eines Knotens, der mit einem oder mehreren Constraints mit anderen Knoten in Be-

[34] Ein Beispiel für eine solche Strukturregel ist die in Abschnitt 3.2.2.2 beschriebene gleichmäßige Verteilung der ausgewählten Objekte in Relation zueinander.
[35] Für eine Übersicht der Typen von Constraintlösern ist [Mai12, S.18ff] nachzulesen.

ziehung gesetzt ist, auch die zusammenhängenden Knoten automatisch mitbewegt werden, sodass die Constraints eingehalten bleiben. Weiterhin können auch die Parameter der erstellten Constraints durch den Nutzer eingestellt werden. Somit zeichnet sich dieser Ansatz durch eine große Flexibilität aus.

In Abbildung 3.17 ist ein Screenshot des Editors *Dunnart*[36] dargestellt, welcher den Ansatz aus [Wyb08] implementiert. Die rechte Sidebar beinhaltet eine Palette mit Buttons für die Erstellung von Constraints. In der Mitte befindet sich die Zeichenfläche, in der ein hierarchisches Diagramm modelliert ist. Die blauen Linien um das eigentliche Diagramm dienen der Visualisierung von erstellten Constraints und ermöglichen deren Manipulation.

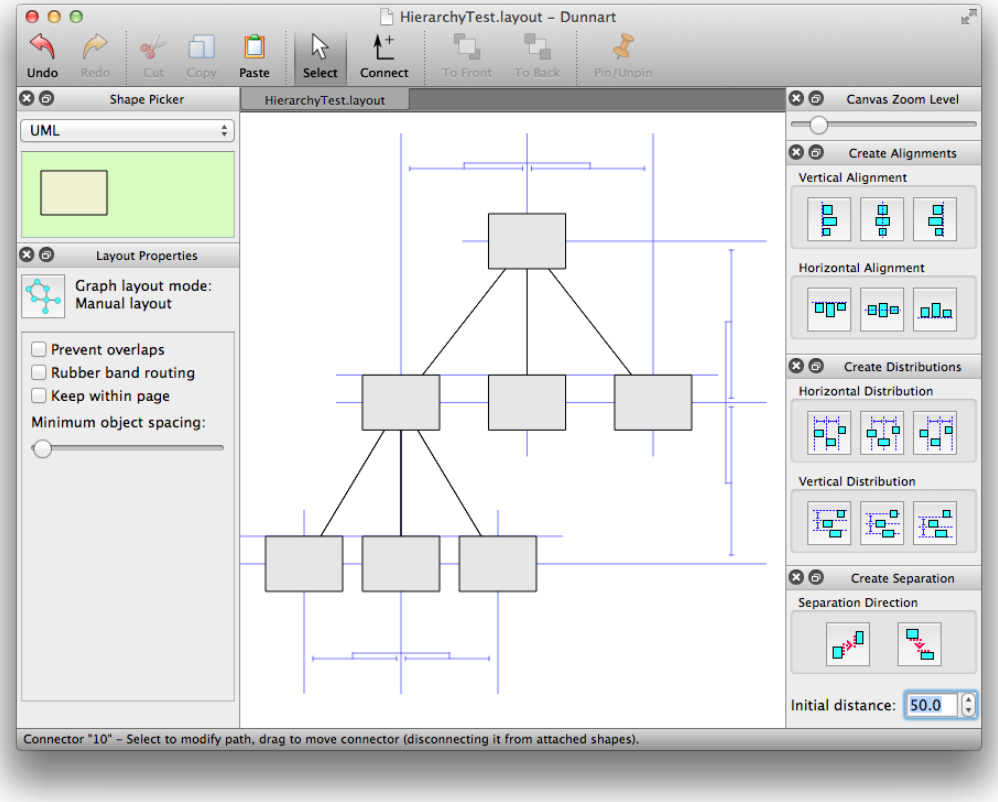


Abbildung 3.17.: Ein Screenshot des Constraint-basierten Editors *Dunnart* mit dem Beispiel eines hierarchischen Diagramms

Die Constraint-basierte Ansätze sind darüber hinaus mit Problemen verbunden. Zum einen kann die Menge der Constraints unvollständig sein oder es werden unzulässige Constraints erstellt. Dies muss im Editor entsprechend behandelt werden, indem ein mögliches Layout gewählt wird bzw. die unzulässigen Constraints mit einer Form von Feedback versehen werden. Zum anderen entstehen Probleme mit der Performance, da der Algorithmus und damit auch der Aufruf des Constraintlösers nach jeder kleinen Änderung ausgeführt wird [Mai12].

[36] `http://dunnart.org`

3.4.1.2. PATTERN-BASIERTER ANSATZ

Der in [Mai12] und [MM10] präsentierte Pattern-basierte Ansatz für das Layout von Diagrammen drückt die Strukturregeln in Form von **Layout-Patterns** aus. Dieser Ansatz kann für beliebige visuelle Sprachen eingesetzt werden, die mit Metamodellen beschrieben werden können. Dabei besteht das sprachenspezifische Metamodell aus zwei Teilen, nämlich den Metamodellen für die abstrakte und konkrete Syntax. Das zuletzt genannte Metamodell beschreibt die visuellen Eigenschaften der Sprache und wird daher durch die Layout-Berechnung beeinflusst.

Die Beschreibung der Layout-Patterns erfolgt jedoch nicht mithilfe der sprachenspezifischen Metamodellen, sondern mit sprachenunabhängigen **Pattern-spezifischen Metamodellen**. In [Mai12] werden u.a. Pattern-spezifische Metamodelle für Mengen von Elementen, für geordnete Listen von Elementen und für Graphen präsentiert. Zu den darauf aufbauenden Layout-Patterns gehören u.a. horizontale bzw. vertikale Ausrichtung (Menge von Elementen), gleicher horizontaler bzw. vertikaler Abstand (geordnete Liste von Elementen) und die Algorithmen zum Graphzeichnen für das baumbasierte, hierarchische oder zirkuläre Layout (Graph). Eine komplette Liste der vorgestellten Layout-Patterns ist unter [Mai12, S.55] zu finden.

Bei der Instanziierung eines Layout-Patterns wird das sprachenspezifische Modell auf ein Pattern-spezifisches Modell abgebildet und auf gegebene bzw. alle Elemente des Diagramms angewendet[37]. Dadurch wird eine Wiederverwendung der Layout-Patterns für diverse visuelle Sprachen gewährleistet.

Weiterhin enthalten die Layout-Patterns Prädikate, die für ein valides Layout des Diagramms erfüllt sein müssen. Die Erfüllung der Prädikate erfolgt durch die Anwendung von Regeln, die die beteiligten Layout-Variablen anpassen und ebenso von den Layout-Patterns gekapselt werden. Die Regeln können bestehende Algorithmen für das Layout von Diagrammen wiederverwenden, u.a. auch Algorithmen zum Graphzeichnen (siehe Abschnitt 3.3.1.1) und Constraints-basierte Algorithmen (siehe Abschnitt 3.4.1.1).

Die instanziierten Layout-Patterns werden nach jeder Änderung des Diagramms durch einen Kontroll-Algorithmus ausgewertet, dessen Aufgabe es ist, alle Prädikate der Layout-Constraints zu erfüllen und damit ein valides Layout für das Diagramm zu berechnen.

Dieser Ansatz ist insbesondere für interaktive Umgebungen wie z.B. visuelle Editoren geeignet. Die Erstellung des Layouts wird dem Nutzer überlassen, indem er die Layout-Patterns instanziiert. Diese können entweder global oder für ausgewählten Knoten spezifiziert sein. Wenn keine Instanzen der Layout-Patterns verfügbar sind, können die Knoten im Diagramm frei positioniert werden. Anderseits wird die Interaktion durch die instanziierten Layout-Patterns beeinflusst und möglicherweise eingeschränkt.

[37] Dieser Sachverhalt wird mithilfe eines Beispiels in [Mai12, S.59ff] anschaulich gemacht.

Der beschriebene Ansatz wurde in Form eines Layout-Frameworks[38] implementiert und in visuellen Editoren eingesetzt, die mithilfe von *DiaMeta*[39] bzw. *Graphical Editing Framework*[40] erzeugt wurden [Mai12].

In Abbildung 3.18 wird ein Screenshot des *DiaMeta Graph Editors* zur Erstellung von gerichteten Graphen gezeigt. Die rechte Sidebar enthält Buttons zur Instanziierung von Layout-Patterns. In der rechten unteren Ecke sind die durch den Nutzer erstellten Layout-Patterns aufgelistet. In dem mittleren Bereich des Fensters befindet sich die Zeichenfläche, die neben der Darstellung der Knoten und Kanten auch die einzelnen instanziierten Layout-Patterns visualisiert. In dem aufgeführten Beispiel wurden Layout-Patterns für die horizontale und vertikale Ausrichtung, gleichmäßige Verteilung und hierarchisches Layout verwendet. Für eine bessere Vorstellung der Interaktion im Editor sind die offiziellen Screencasts[41] anzuschauen.

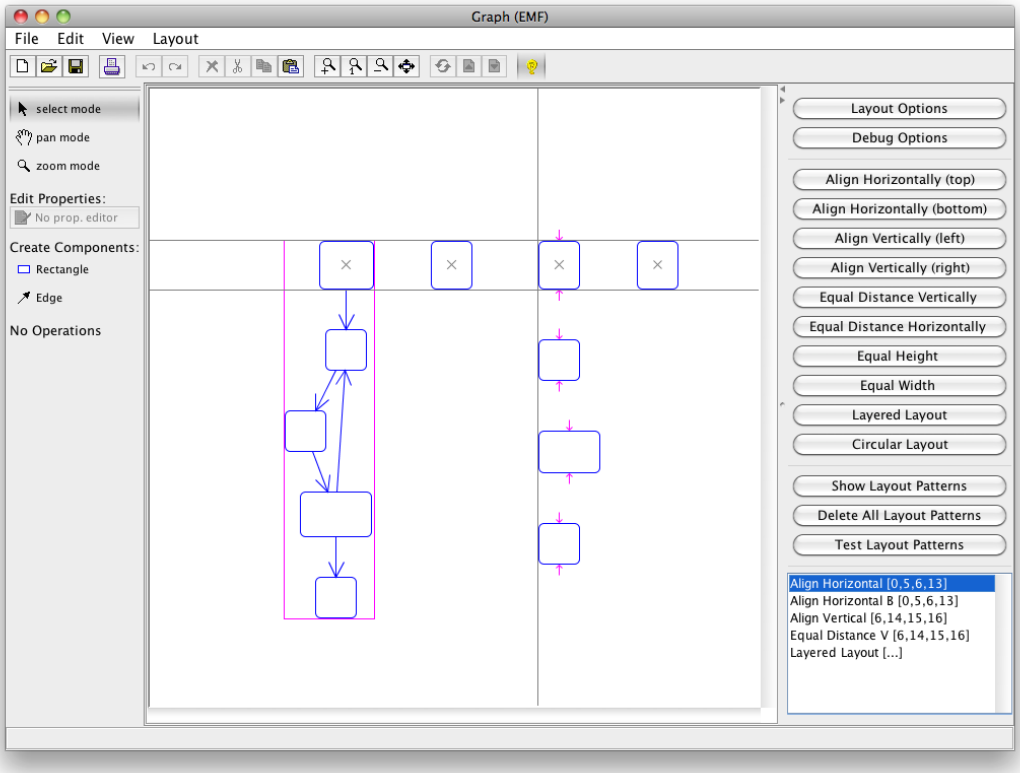


Abbildung 3.18.: Ein Screenshot des *DiaMeta Graph Editors* (Quelle: `http://www.unibw.de/inf2/Personen/Wissen_Mitarbeiter/sonja/research/layoutframework/graphEditor.png`, Aufruf: 20.09.2014)

[38] `http://www.unibw.de/inf2/Personen/Wissen_Mitarbeiter/sonja/research/layoutframework`
[39] Ein Framework zur Generierung von Editoren für visuelle Sprachen basierend auf Spezifikationen mittels Metamodellen. `http://www.unibw.de/inf2/DiaGen`
[40] `http://www.eclipse.org/gef`
[41] *DiaMeta Graph Editor*: `http://www.sonjamaier.de/dyndraw/screencasts/graphEditor.mov`
DiaMeta Ecore Editor: `http://www.sonjamaier.de/dyndraw/screencasts/ecoreEditor.mov`

3.4.2. ANWENDUNGSSPEZIFISCHE ANSÄTZE

Die strukturbasierten benutzergesteuerten Ansätze für das halbautomatische Layout, die in Abschnitt 3.4.1 beschrieben wurden, sind sehr universell und können in Editoren für verschiedene visuellen Sprachen eingesetzt werden. Aus diesem Grund können die syntaktischen und semantischen Eigenschaften der Sprachen nicht gründlich in den Algorithmen berücksichtigt werden. Dahingegen gibt es anwendungsspezifische Ansätze, deren Algorithmen für konkrete visuelle Sprachen entworfen sind. Im Folgenden wird ein Beispiel vorgestellt.

3.4.2.1. SMART LAYOUT IN MINDNODE

MindNode[42] ist eine benutzerfreundliche Desktop-Anwendung zur Erstellung von Mindmaps für *Mac OS X*. Die Mindmaps haben eine baumbasierte Struktur mit einem oder mehreren zentralen Knoten. Weiterhin gilt, dass jeder Knoten mehrere Unterknoten besitzen kann und alle Knoten außer den zentralen Knoten genau einen Oberknoten haben. Die hierarchischen Relationen werden mit farbigen Zweigen dargestellt. Zusätzlich bietet *MindNode* die Möglichkeit der Erstellung von Querverbindungen zwischen Knoten aus unterschiedlichen Teilen der Mindmap [Ide14]. Dies wird mit gestrichelten Pfeilen gekennzeichnet. In Abbildung 3.19 ist ein Beispiel einer Mindmap zu sehen.

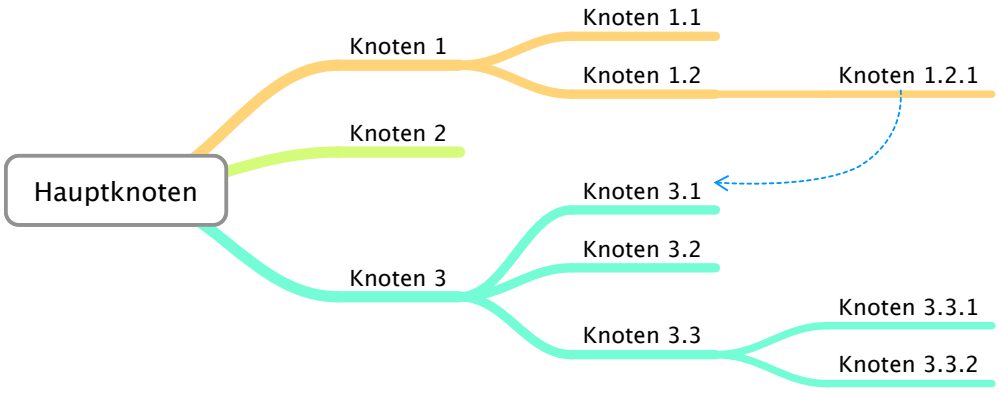


Abbildung 3.19.: Beispiel einer Mindmap in *MindNode*

MindNode stellt eine interaktive Funktion für eine halbautomatische Layout-Unterstützung namens „Smart Layout“ bereit [Ide14]. Wenn diese Funktion ausgeschaltet ist, können einzelne Knoten der Mindmap frei positioniert werden. Dahingegen nimmt die Mindmap, wenn die Funktion eingeschaltet ist, eine vorgerechnete Baumstruktur an und die Verschiebungsaktion eines Knotens drückt in diesem Fall die Absicht einer Layout-Modifikation aus.

In Abbildung 3.20 wird die Funktionsweise der Funktion „Smart Layout“ anhand der Verschiebungsaktion gezeigt. Zunächst wird in 3.20(a) ein Knoten angeklickt, dessen Position angepasst werden soll. Mit „Drag and Drop“ wird in 3.20(b) der Knoten auf die gewünschte Position

[42] `http://mindnode.com`

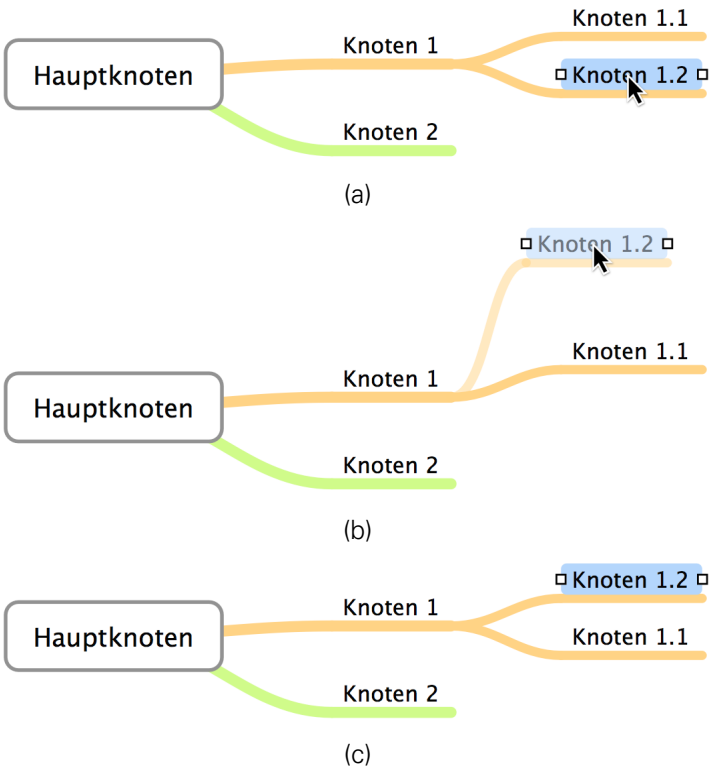


Abbildung 3.20.: Die Verschiebungsaktion mit der eingeschalteten Funktion „Smart Layout“ in *MindNode*

verschoben. Die Verbindung zu dem Oberknoten wird dabei mit einem helleren Zweig veranschaulicht. Nach dem Loslassen der Maustaste in 3.20(c) wird die gewünschte Position des verschobenen Knotens ausgewertet, ein neues Layout der Mindmap anhand des Hinweises durch die Verschiebungsaktion berechnet und anschließend auf die Mindmap angewendet. Dabei werden alle beeinflussten Knoten von ihren aktuellen Positionen zu ihren neuen Positionen mithilfe einer Animation bewegt. Somit kann der Nutzer das Layout beeinflussen, wird aber in den Möglichkeiten eingeschränkt. Die integrierte Layout-Berechnung führt zu einem optimalen Layout und ist vor allem durch die Struktur der visuellen Sprache für Mindmaps möglich.

3.4.3. ZUSAMMENFASSUNG DER EIGENSCHAFTEN

3.4.3.1. VORTEILE

- **Unterstützung der interaktiven Bearbeitung** Die Ansätze für das halbautomatische Layout sind für interaktive Umgebungen geeignet und ermöglichen eine unmittelbare Bearbeitung des Diagramms.
- **Persistente Strukturbeschreibung** Die durch den Nutzer ausgeführten Änderungen haben einen persistenten Charakter und werden während der weiteren Bearbeitung des Diagramms nicht verworfen. Außerdem erfolgt die Beschreibung der Struktur mithilfe von Regeln, die dem Nutzer zur Verfügung stehen. Dadurch führt die halbautomatische Layout-Unterstützung zur Vereinfachung der Layout-Erstellung.

- Erhaltung des mentalen Modells Da die Wahrnehmungsorganisation in den interaktiven Umgebungen eine große Bedeutung hat [SKM93, Mai12], zeichnen sich die halbautomatischen Ansätze durch den Erhalt des mentalen Modells während der Ausführung von inkrementellen Änderungen aus [GSE+14].

3.4.3.2. NACHTEILE

- Fehlende Einhaltung der ästhetischen Prinzipien Da der Nutzer für die Strukturbeschreibung verantwortlich ist, können die ästhetischen Prinzipien verletzt werden. Insbesondere die allgemeinen Ansätze besitzen eine mangelhafte bis keine Unterstützung der syntaktischen und semantischen Regeln von konkreten Diagrammtypen.

4. ANSATZ FÜR DAS INTERAKTIVE UND DIAGRAMMSPEZIFISCHE LAYOUT VON GRAPHBASIERTEN SOFTWAREDIAGRAMMEN

In diesem Kapitel wird ein Ansatz für das interaktive und diagrammspezifische Layout von graphbasierten Softwarediagrammen präsentiert. Zunächst werden in Abschnitt 4.1 die Kriterien für den Entwurf des Ansatzes aufgestellt. Eine kurze Übersicht der Funktionsweise wird in Abschnitt 4.2 gegeben. In Abschnitt 4.3 werden die eingesetzten Mechanismen der Interaktion näher erläutert. Eine detaillierte Beschreibung der Layout-Patterns und deren Rolle in dem präsentierten Ansatz folgt in Abschnitt 4.4. In Abschnitt 4.5 wird erklärt, wie die Layout-Berechnung im Detail funktioniert. In Abschnitt 4.6 wird der präsentierte Ansatz von den bestehenden Ansätzen abgegrenzt und mit ihnen verglichen. Das Kapitel wird mit einer Zusammenfassung in Abschnitt 4.7 abgeschlossen.

4.1. KRITERIEN

Im Folgenden werden Kriterien für den präsentierten Ansatz aufgestellt. Die Wahl der Kriterien stützt sich in erster Linie auf die Prinzipien für die agile Modellierung aus [Amb02]. Weiterhin orientieren sich die Kriterien an den in Kapitel 3 vorgestellten Ansätzen, deren positive Eigenschaften in die Auswahl einfließen. Schließlich haben auch die ästhetischen Prinzipien (siehe Abschnitt 2.2) sowie die Rahmenbedingungen dieser Arbeit (siehe Abschnitt 1.3) einen Einfluss auf die gewählten Kriterien.

K.1 **GUI** Wie bereits in Abschnitt 1.3 erwähnt wurde, beschäftigt sich diese Arbeit ausschließlich mit Mechanismen für das Layout von Diagrammen, die in Tools für klassische grafische Benutzeroberflächen eingesetzt werden. Daher soll der entwickelte Ansatz für klassische grafische Benutzeroberflächen ausgelegt sein und von üblichen GUI-basierten Bedientechniken Gebrauch machen.

K.2 **Interaktivität** Die Manipulation von Diagrammen soll interaktiv erfolgen. Der Nutzer soll in der Lage sein, mit dem Diagramm unmittelbar interagieren zu können und die Layout-Anpassungen sollen eine direkte Auswirkung auf das manipulierte Diagramm haben. Demzufolge soll die Eingabe mit der Ausgabe fest gekoppelt sein[43].

K.3 **Unmittelbares Feedback** Bei einer direkten Manipulation des Diagramms soll dem Nutzer ein unmittelbares Feedback gegeben werden, sodass das Ergebnis der Manipulation während ihrer Ausführung sichtbar wird [Wyb08, S.69].

K.4 **Förderung des Prozesses der Diagramm-Erstellung** Der Nutzer soll den Inhalt des Diagramms bearbeiten und inkrementell modifizieren können [GSE+14]. Dabei soll das Layout nach jeder Änderung des Inhalts angepasst werden. Weiterhin soll es möglich sein, das Layout des Diagramms bis zu einem bestimmten Grad nach den Präferenzen des Nutzers beeinflussen zu können.

K.5 **Erhaltung des mentalen Modells** Die Layout-Änderungen im Diagramm sollen möglichst intuitiv erfolgen, insbesondere soll das mentale Modell erhalten werden (siehe Abschnitt 2.1.3).

K.6 **Förderung der Konzentration auf den Inhalt** Der Inhalt eines Diagramms soll eine wichtigere Rolle als seine Repräsentation haben [Amb02, S.38ff]. Diese Tatsache soll in dem Ansatz berücksichtigt werden, indem der Aufwand für das Erzeugen des Layouts verringert wird. Außerdem sollen die Aktionen zur Anpassung der visuellen Eigenschaften möglichst eingeschränkt sein, ohne einen negativen Einfluss auf die Ästhetik des Layouts zu haben (siehe Kriterium K.7).

K.7 **Berücksichtigung der ästhetischen Prinzipien** Die Layout-Berechnung soll unter Berücksichtigung der in Abschnitt 2.2 beschriebenen ästhetischen Prinzipien für graphbasierte Softwarediagramme erfolgen.

K.8 **Berücksichtigung der Syntax und Semantik** Die Syntax- und Semantikregeln der einzelnen Diagrammtypen sollen berücksichtigt werden und die Möglichkeit der Verletzung der Regeln soll verhindert werden.

K.9 **Benutzerfreundlichkeit** Der Ansatz soll möglichst benutzerfreundlich sein. Insbesondere ist wichtig, die Manipulation des Diagramms intuitiv zu gestalten und eine potenzielle Frustration der Nutzer durch die Einschränkungen der Layout-Möglichkeiten zu verhindern.

[43] Die Kopplung bezieht sich auf den Mauszeiger und die darunter liegende Repräsentation des Diagramms. Aus der Sicht der Mensch-Computer-Interaktion ist die Nutzung der Maus als Eingabegerät von der Ausgabe am Monitor stets entkoppelt.

4.2. FUNKTIONSWEISE

Der präsentierte Ansatz operiert auf dem Modell der konkreten Syntax einer graphbasierten visuellen Sprache und berechnet die Layout-Eigenschaften für Knoten und Kanten eines Diagramms (siehe Abschnitt 2.1.1). Die Berechnung erfolgt **halbautomatisch** und erfordert eine interaktive Umgebung, denn der Layout-Prozess wird durch die **Interaktion** des Nutzers mit dem Diagramm beeinflusst. Dabei wird versucht, ein optimales Verhältnis zwischen der Möglichkeit der flexiblen Modifizierung der Layout-Eigenschaften und einer automatisierten Layout-Berechnung zu erreichen. Der Ansatz kombiniert die Vorteile des manuellen und automatischen Layouts und lässt sich daher in die Gruppe der Ansätze für das interaktive halbautomatische Layout einordnen (siehe Abschnitt 3.4).

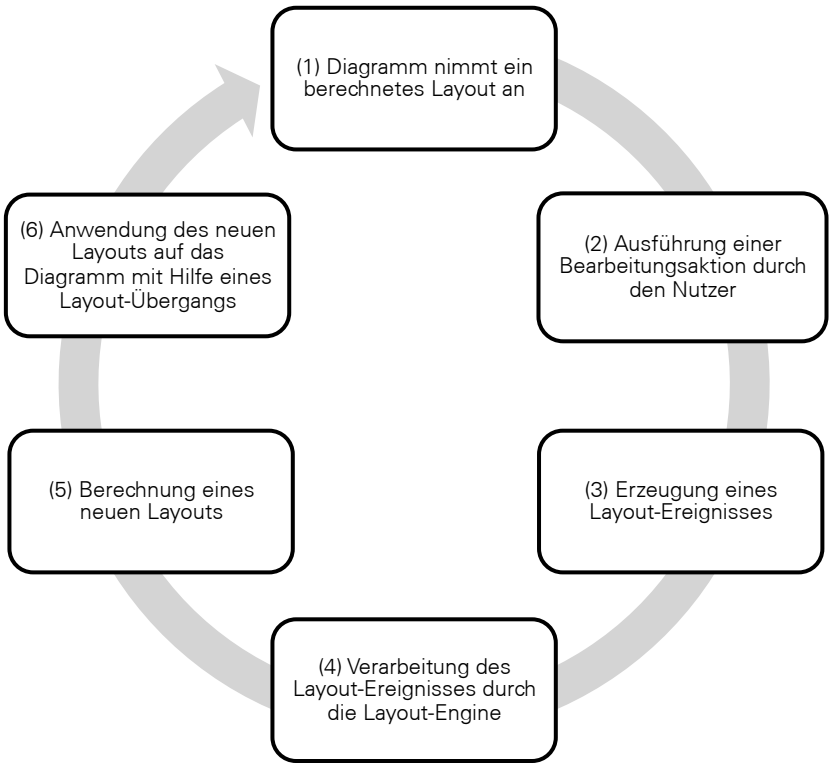


Abbildung 4.1.: Die Interaktionsschleife des präsentierten Ansatzes

Den Kern des Ansatzes bildet eine Interaktionsschleife, die sich aus mehreren Schritten zusammensetzt und in Abbildung 4.1 illustriert ist. Zunächst befindet sich das bearbeitete Diagramm in einem stabilen Zustand und nimmt ein berechnetes Layout an (Schritt 1). Der Nutzer kann die Interaktion auslösen, indem er eine **Bearbeitungsaktion** ausführt, die mithilfe von **Bedienungsmechanismen** im Diagramm realisiert und veranschaulicht wird (Schritt 2). Für jede Bearbeitungsaktion werden ein oder mehrere **Layout-Ereignisse** erzeugt, die die bewirkte Änderung im Diagramm beschreiben (Schritt 3). Diese Layout-Ereignisse werden durch eine **Layout-Engine** verarbeitet, die für den Typ des Diagramms speziell angepasst ist (Schritt 4). Die Layout-Engine verwaltet intern Instanzen von **Layout-Patterns**, die die Struktur des Diagramms beschreiben. Neben den instanziierten Layout-Patterns bilden auch der aktuelle Inhalt und das Layout des Diagramms die Grundlage für die Layout-Berechnung. Diese wird im

nächsten Schritt durchgeführt (Schritt 5). Anschließend wird das neu berechnete Layout auf das Diagramm angewendet und mithilfe eines **Layout-Übergangs** visualisiert (Schritt 6). Durch die Wiederholung dieser Schritte entsteht der gesamte Bearbeitungsprozess. Die einzelnen genannten Bestandteile und Phasen der Layout-Berechnung werden in weiteren Abschnitten im Detail erläutert.

4.3. MECHANISMEN DER INTERAKTION

Das Kriterium K.4 erfordert eine Möglichkeit der Bearbeitung des Diagramms. Der Nutzer soll in der Lage sein, neue Knoten und Kanten hinzuzufügen und bestehende Knoten und Kanten zu manipulieren bzw. zu löschen. Durch das Kriterium K.2 wird gefordert, dass die Bearbeitung interaktiv erfolgt. Dies wird durch die Unterstützung von **Bearbeitungsaktionen** gewährleistet, die die Grundlage der Interaktion bilden. Die Bearbeitungsaktionen werden in Abschnitt 4.3.1 im Detail beschrieben.

Weiterhin sollen die Möglichkeiten der Manipulation von Objekten eingeschränkt werden, um das Ablenken durch manuelle Layout-Anpassungen zu minimieren (Kriterium K.6). Dies wird durch die Einschränkung der aus den manuellen Layout-Ansätzen (siehe Abschnitt 3.2) bekannten freien Positionierung der Knoten und den Einsatz eines **neuen Bedienungsmechanismus** für die Verschiebungsaktion erreicht, dem sich Abschnitt 4.3.2 widmet.

Die Erhaltung des mentalen Modells (Kriterium K.5) wird durch eine **Animation der Layout-Übergänge** gewährleistet. Dies wird in Abschnitt 4.3.3 thematisiert.

4.3.1. BEARBEITUNGSAKTIONEN

Die interaktiven Ansätze werden in Editoren eingesetzt, die in der Regel über einen Canvas[44] verfügen. Der Canvas ist ein rechteckiger Bereich des Editor-Fensters, in dem das Diagramm dargestellt wird und in dem die Interaktion mit dem Diagramm stattfindet. Die Interaktion setzt sich aus Bearbeitungsaktionen zusammen, die durch den Nutzer ausgeführt werden. Um das Kriterium K.1 zu erfüllen, sollen alle Bearbeitungsaktionen auf GUI-Techniken basieren.

In dieser Arbeit werden die grundlegenden Bearbeitungsaktionen präsentiert, die insbesondere für die Entwicklung des Prototyps (siehe Kapitel 5) eine wichtige Rolle spielen. Natürlich ist eine Erweiterung der unterstützten Bearbeitungsaktionen möglich.

Je nach der Art der Modifizierung lassen sich die Bearbeitungsaktionen in zwei Gruppen unterteilen. Zum einen sind das Aktionen, die den Inhalt des Diagramms modifizieren und zum anderen sind das Aktionen, die das Layout des Diagramms modifizieren, ohne den Inhalt zu verändern.

[44] Der „Canvas“ ist ein alternativer Begriff für die Zeichenfläche, der in dieser Arbeit verwendet wird.

4.3.1.1. BEARBEITUNGSAKTIONEN ZUR MODIFIZIERUNG DES INHALTS

Zu den Bearbeitungsaktionen, die den Inhalt verändern, gehört das Hinzufügen und Löschen von Knoten bzw. Kanten und die Veränderung ihrer Eigenschaften wie z.B. das Umbenennen der Knoten. Da die Intention für die Ausführung dieser Aktionen nicht die Anpassung des Layouts ist, weisen diese Aktionen ein verzögertes Feedback auf, d.h. das Layout wird erst nach der Beendigung der Bearbeitung angepasst[45].

4.3.1.2. BEARBEITUNGSAKTIONEN ZUR MODIFIZIERUNG DES LAYOUTS

Der einzige Vertreter der Bearbeitungsaktionen, die ausschließlich das Layout modifizieren, ist die Verschiebung eines Knotens. Aufgrund der Einschränkung der freien Positionierung wird durch die Verschiebung eine Empfehlung für die Position eines Knotens angegeben. Diese Aktion macht sich einen neuen Bedienungsmechanismus zunutze, der ein unmittelbares Feedback aufweist und in Abschnitt 4.3.2 näher beschrieben wird. Dieser Mechanismus bildet die Grundlage des gesamten Ansatzes.

4.3.2. MECHANISMUS DER TEMPORÄREN SCHICHT

Alle interaktiven Ansätze für das Layout von Diagrammen aus Kapitel 3 bis auf das „Smart Layout“ in *MindNode* (siehe Abschnitt 3.4.2.1) unterstützen **freie Positionierung** der Objekte im Diagramm. Diese Art der Bedienung wird im manuellen Layout zusätzlich um temporäre Hilfsfunktionen (siehe Abschnitt 3.2) und in strukturbasierten benutzergesteuerten Ansätzen um Möglichkeit der Erzeugung von Strukturregeln (siehe Abschnitt 3.4.1) erweitert. Durch diese Erweiterungen wird die freie Positionierung eingeschränkt, wodurch in der Regel ein besseres Layout des Diagramms erreicht werden kann. Dahingegen ist die freie Positionierung in „Smart Layout“ in *MindNode* und dem in dieser Arbeit präsentierten Ansatz **nicht gestattet**. Stattdessen wird durch die Verschiebung der Objekte ihre ungefähre Position angegeben, die intern ausgewertet wird und eine Anpassung des Layouts bewirkt.

Dieses Verhalten wird für die Verschiebungsaktion der Knoten eingesetzt, indem bei der Verschiebung eines Knotens eine temporäre Schicht erzeugt wird, in der sich der Knoten frei positionieren lässt. Im Diagramm wird der manipulierte Knoten durch einen Platzhalter repräsentiert, der nur zulässige Positionen annehmen kann. Dieser Mechanismus basiert auf der Technik **„Drag and Drop“** und ist in drei Phasen aufgeteilt. Im Folgenden wird er an einem Beispiel detailliert erklärt.

Bevor die Verschiebungsaktion beginnen kann, müssen sich im Canvas einige Knoten befinden, so wie es in Abbildung 4.2(a) der Fall ist.

In der ersten Phase wird ein Knoten im Canvas angeklickt, was in Abbildung 4.2(b) durch einen Mauszeiger gekennzeichnet ist. Dabei wird der angeklickte Knoten in eine neu eingefügte **tem-**

[45] Eine Ausnahme bildet das Hinzufügen eines neuen Knoten, das sich ähnlich wie die Verschiebungsaktion aus Abschnitt 4.3.1.2 verhält.

poräre Schicht ausgelagert, die vor dem Canvas positioniert ist, so wie es in Abbildung 4.3 skizziert ist. Im Canvas wird die ursprüngliche Darstellung des verschobenen Knotens durch einen **temporären Platzhalter** ersetzt, welcher dem Nutzer die Zielposition des Knotens andeutet. Der Platzhalter ist in Abbildungen 4.2(c) und 4.3 durch ein Rechteck repräsentiert, das mit einer grauen gestrichelten Linie umrandet ist.

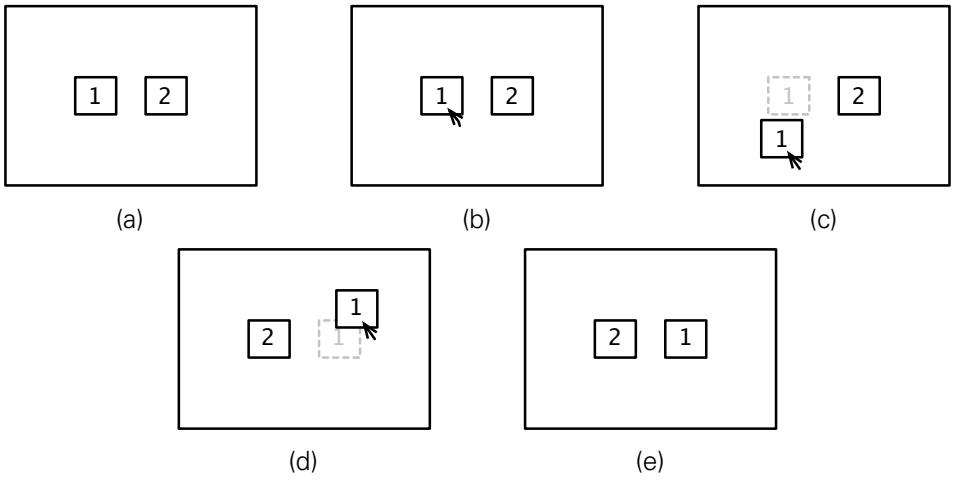


Abbildung 4.2.: Beispiel einer Verschiebungsaktion

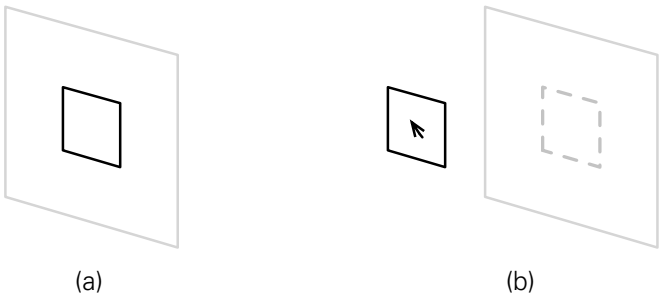


Abbildung 4.3.: Darstellung der temporären Schicht vor dem Canvas

In der zweiten Phase findet eine Bewegung des Mauszeigers statt. Da die temporäre Schicht an den Mauszeiger gebunden ist, wird die ausgelagerte Darstellung des Knotens frei mit dem Mauszeiger bewegt, wohingegen der Platzhalter im Canvas seine Position zunächst nicht ändert. Dies ist in Abbildung 4.2(c) illustriert.

Die Position des Mauszeigers wird nach jeder Bewegung ausgewertet und sobald sich der Mauszeiger in einem bestimmten Bereich des Diagramms befindet, für den das Layout geändert werden soll, wird das Layout im unterliegenden Canvas mithilfe einer Animation (siehe Abschnitt 4.3.3) angepasst. Dabei wird die Änderung der Layout-Eigenschaften des verschobenen Knotens auf den Platzhalter angewendet, so wie es in Abbildung 4.2(d) dargestellt ist. Durch die Anpassung des Layouts während der Interaktion wird ein unmittelbares Feedback erreicht (Kriterium K.3). Dagegen wird das Layout in *MindNode* (siehe Abschnitt 3.4.2.1) erst nach dem Loslassen der Maustaste angepasst und damit das Feedback verzögert.

Dieser Abschnitt beschäftigt sich ausschließlich mit der Beschreibung der Interaktion, die kon-

krete Funktionsweise der Layout-Berechnung kann in Abschnitt 4.5 nachgelesen werden.

Die dritte Phase beginnt mit dem Loslassen der Maustaste. Danach springt der verschobene Knoten auf seine Zielposition, nämlich die Position des Platzhalters. Anschließend werden der Platzhalter sowie die temporäre Schicht entfernt und die Verschiebungsaktion wird beendet. Die Abbildung 4.2(e) zeigt den Endzustand.

4.3.3. LAYOUT-ÜBERGÄNGE

In vorhergehenden Abschnitten wurde beschrieben, dass nach der Änderung des Inhalts bzw. während der Verschiebung eines Knotens das Layout angepasst wird. Konkret handelt es sich um einen Übergang vom aktuellen zu einem neu berechneten Layout. Um diese Übergänge für den Nutzer nachvollziehbar zu machen, wird eine Animation eingesetzt, die insbesondere für die Erhaltung des mentalen Modells sorgt (Kriterium K.5).

4.4. LAYOUT-PATTERNS

Die Layout-Patterns dienen der Beschreibung der Struktur des Layouts eines Diagramms. Ähnlich wie in [Mai12] handelt es sich um allgemeine wiederverwendbare Lösungen für Layout-Probleme. Die Auffassung der Layout-Patterns in dieser Arbeit ist im Vergleich zu dem Konzept aus [Mai12] (siehe Abschnitt 3.4.1.2) sehr vereinfacht. Zum einen werden die Layout-Patterns nicht an einem separaten Metamodell definiert, sondern direkt auf dem Metamodell der konkreten Syntax des gewählten Beispiels. Zum anderen besitzen die Layout-Patterns keine komplexe Struktur der Regelausführung wie in [Mai12].

Die präsentierten Layout-Patterns wurden von den in Abschnitt 2.2 aufgeführten ästhetischen Kriterien abgeleitet und orientieren sich insbesondere an der Unterstützung der Interaktion u.a. durch das Konzept der Variation. Des Weiteren führt diese Arbeit eine Aufteilung auf implizite und explizite Patterns ein. Die Problematik der Layout-Patterns wird im Folgenden detailiert behandelt.

4.4.1. IMPLIZITE LAYOUT-PATTERNS

Die impliziten Layout-Patterns beschreiben allgemeine Eigenschaften für die Layout-Berechnung. Dafür werden keine expliziten Strukturregeln erzeugt, wie das der Fall bei den expliziten Layout-Patterns (siehe Abschnitt 4.4.2) ist, sondern sie werden von dem Algorithmus der Layout-Berechnung als geltend angenommen.

4.4.1.1. GRÖSSE DER KNOTEN

In allen interaktiven Ansätzen für das Layout von Diagrammen aus Kapitel 3 hat der Nutzer die Möglichkeit die Größe der Objekte zu verändern. Weiterhin werden in dem allgemeinen Ansatz

aus [Mai12] Layout-Patterns präsentiert, die Regeln für die Größe der Objekte beschreiben. Somit ist die Größe der Objekte eine Eigenschaft, die zum einen von dem Nutzer und zu anderem von dem Layout-Algorithmus geändert werden kann.

Dagegen ist die Größe der Knoten in dieser Arbeit ausschließlich durch den Inhalt des Knotens bestimmt. Sie nimmt zunächst eine minimale Größe an und wenn der Inhalt[46] in den Knoten mit der minimalen Größe nicht passt, wird der Knoten entsprechend erweitert. Somit kann die Größe der Knoten weder durch den Nutzer noch durch die Layout-Berechnung geändert werden. Dieses implizite Layout-Pattern fördert insbesondere die Konzentration auf den Inhalt (Kriterium K.6) und die ästhetische Gestaltung (Kriterium K.7).

4.4.1.2. VERHINDERUNG DER KNOTEN-ÜBERLAPPUNG

Dadurch, dass die Größe der Knoten bekannt ist und die Position der Knoten berechnet wird, kann die Knoten-Überlappung verhindert werden. Dies trägt zur Ästhetik des Diagramms (Kriterium K.7) und insbesondere der Einhaltung der Abstände zwischen den Knoten (Prinzip ÄP.1) bei.

4.4.1.3. ZENTRIERUNG DES DIAGRAMM-INHALTS

Die Zentrierung des Diagramm-Inhalts ist ein weiteres ästhetisches Layout-Pattern, das eine eindeutige Bestimmung des Layouts ermöglicht. Die Berechnung des Layouts erfolgt relativ und wird so auf das Diagramm angewendet, dass der Diagramm-Inhalt immer in die Mitte des Canvas platziert wird und dadurch die Zeichenfläche optimal ausgenutzt wird (Prinzip ÄP.5). Dies wird in Abbildung 4.4 veranschaulicht.

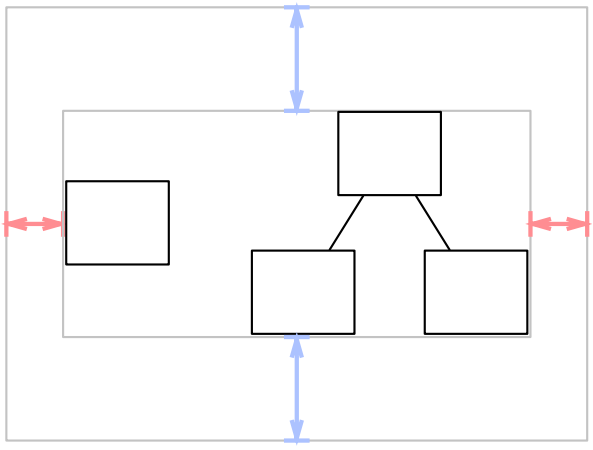


Abbildung 4.4.: Die Zentrierung des Diagramm-Inhalts im Canvas

Durch dieses implizite Layout-Pattern wird neben der Förderung der Ästhetik (Kriterium K.7) vor allem die Einschränkung der freien Positionierung (siehe Abschnitt 4.3.2) möglich gemacht.

[46] Für eine Klasse der Notationssprache UML handelt es sich um ihre Bezeichnung und die Liste der Attribute und Methoden.

4.4.2. EXPLIZITE LAYOUT-PATTERNS

Die expliziten Layout-Patterns referenzieren Knoten im Diagramm und beschreiben deren Struktur anhand von geometrischen Eigenschaften. Während der Erstellung eines Diagramms werden ähnlich wie in [Mai12] **Instanzen der Layout-Patterns** verwaltet. Im Gegensatz zu [Mai12] erfolgt die Verwaltung automatisch und basiert auf dem Inhalt des Diagramms. Der Nutzer hat daher keine Möglichkeit die Layout-Patterns selbst zu erstellen[47].

Um eine Art der Manipulation der Knoten zu ermöglichen, wird das Konzept der **Variation der expliziten Layout-Patterns** eingeführt. Je nach der Art des expliziten Layout-Patterns kann der Nutzer zwischen mehreren Variationen der Anordnung der Knoten (z.B. Reihenfolge) wählen. Das endgültige Layout wird jedoch durch die Geometrie bestimmt. Diese Art der Manipulation fördert die Konzentration auf den Inhalt und ergänzt die Einschränkung der freien Positionierung (Kriterium K.6).

Im Folgenden werden zwei konkrete explizite Layout-Patterns vorgestellt, nämlich das Ausrichtungspattern (siehe Abschnitt 4.4.2.1) und das T-Shape-Pattern (siehe Abschnitt 4.4.2.2). Mit der Implementierung der expliziten Layout-Patterns im Prototyp beschäftigt sich Abschnitt 5.3.1.4.

4.4.2.1. AUSRICHTUNGSPATTERN

Das Ausrichtungspattern beschreibt eine Reihe an Knoten und kombiniert die Ausrichtung und Verteilung der Knoten aus dem manuellen Layout (siehe Abschnitt 3.2.2.2) bzw. die Layout-Patterns zum Ausrichten und Verteilen aus [Mai12, S.143ff und S.136ff]. Die Knoten werden in einer horizontalen oder vertikalen Reihe gleichmäßig verteilt und an den Kanten bzw. Mittelachsen ausgerichtet. In den Abbildungen 4.5 und 4.6 werden zwei Beispiele für das horizontale Verteilen mit der Ausrichtung an den oberen Kanten bzw. Mittelachsen dargestellt. Die gleichen Abstände sind mit roten Pfeilen und die Anordnung durch eine graue Linie gekennzeichnet.

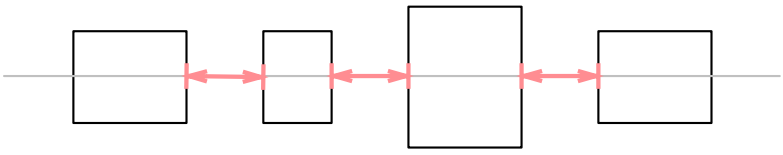


Abbildung 4.5.: Horizontales Ausrichtungspattern mit der Ausrichtung an den Mittelachsen

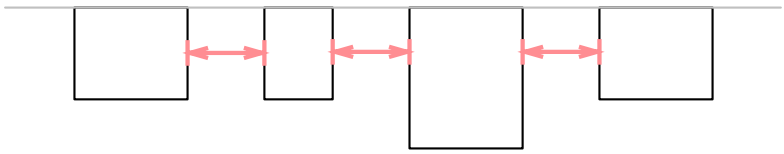


Abbildung 4.6.: Horizontales Ausrichtungspattern mit der Ausrichtung an den oberen Kanten

[47] Eine Möglichkeit der Erstellung von Layout-Patterns durch den Nutzer wird in Abschnitt 7.2.3 diskutiert.

Das Variieren des Ausrichtungspatterns besteht in der Veränderung der Reihenfolge der ausgerichteten Knoten. Dies erfolgt mithilfe der Verschiebung eines Knoten auf die Position eines anderen und basiert auf dem Mechanismus der temporären Schicht (siehe Abschnitt 4.3.2). Ein Ausrichtungspattern mit drei zugeordneten Knoten hat somit sechs möglichen Variationen, die in Abbildung 4.7 veranschaulicht werden.

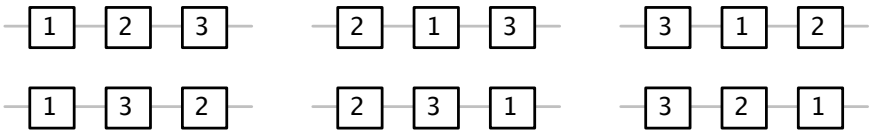


Abbildung 4.7.: Mögliche Variationen des Ausrichtungspatterns mit drei Knoten

4.4.2.2. T-SHAPE-PATTERN

Das T-Shape-Pattern beschreibt die Struktur von Knoten, die in der Form des Buchstabens „T“ angeordnet sind. Dies ist insbesondere für die Darstellung von Hierarchien in Diagrammen von Bedeutung (Prinzip ÄP.8).

Einer der Knoten hat eine spezielle Rolle und ist den anderen Knoten übergeordnet. Die untergeordneten Knoten sind ähnlich wie in dem Ausrichtungspattern gleichmäßig in einer Reihe angeordnet und an den Kanten ausgerichtet, die sich am nächsten zu dem übergeordneten Knoten befinden. Der übergeordnete Knoten befindet sich in einem definierten Abstand zu den untergeordneten Knoten und ist mittig ausgerichtet. In Abbildung 4.8 ist ein Beispiel des T-Shape-Patterns dargestellt, in dem der übergeordnete Knoten oben positioniert ist. Die Abstände sind mit farbigen Pfeilen und die Ausrichtung mit grauen Linien gekennzeichnet. Natürlich gibt es weitere Formen des T-Shape-Patterns, in denen sich der übergeordnete Knoten links, unten oder rechts befindet.

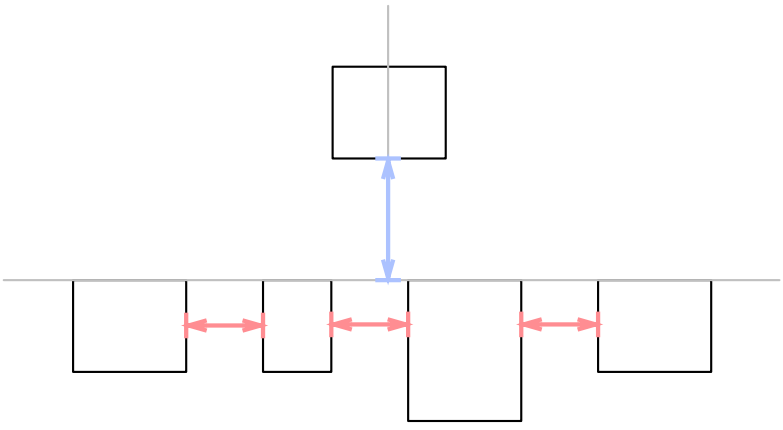


Abbildung 4.8.: Das T-Shape-Pattern mit dem oben positionierten übergeordneten Knoten

Durch das T-Shape-Pattern wird das Ausrichtungspattern erweitert, indem die Reihe der untergeordneten Knoten um einen übergeordneten Knoten ergänzt wird. Genau wie bei dem

Ausrichtungspattern lässt sich bei dem T-Shape-Pattern die Reihenfolge der untergeordneten Knoten variieren.

4.5. BERECHNUNG DES LAYOUTS

Wie bereits in Abschnitt 4.3.1 erläutert wurde, werden während der Bearbeitung eines Diagramms Aktionen durch den Nutzer ausgeführt. Für die ausgeführten Bearbeitungsaktionen werden Layout-Ereignisse generiert, die eine Beschreibung der manipulierten bzw. geänderten Knoten und Kanten sowie die Parameter der Interaktion wie z.B. die Position des Mauszeigers beinhalten. Die Layout-Engine, die für die Berechnung des Layouts zuständig ist, nimmt die erzeugten Layout-Ereignisse entgegen, verarbeitet sie und passt das Layout des Diagramms an. Mit der Layout-Engine beschäftigt sich Abschnitt 4.5.1 näher. Der Prozess der Verarbeitung der Layout-Ereignisse wird in Abschnitt 4.5.2 beschrieben. Schließlich werden die konkreten Layout-Algorithmen in Abschnitt 4.5.3 vorgestellt.

4.5.1. LAYOUT-ENGINE

Die Layout-Engine ist ein zentraler Bestandteil des präsentierten Ansatzes. Sie ist mit dem bearbeiteten Diagramm verknüpft, dessen Layout sie berechnet. Des Weiteren verwaltet sie Instanzen von expliziten Layout-Patterns (siehe Abschnitt 4.4.2), wodurch die Struktur des Diagramms beschrieben wird.

Damit der Prozess der Verarbeitung von Layout-Ereignissen beginnen kann, muss sich die Layout-Engine in einem validen Zustand befinden. Dies wird entweder durch das Speichern des Zustandes der Layout-Engine oder durch die Berechnung eines initialen Layouts gewährleistet. Die Aufgabe der zuletzt genannten Option besteht darin, die expliziten Layout-Patterns ähnlich wie in [MM13] anhand des Inhalts des Diagramms automatisiert zu instanziieren.

4.5.2. VERARBEITUNG DER LAYOUT-EREIGNISSE

Der Lebenszyklus der Layout-Engine besteht aus der sequenziellen Verarbeitung der Layout-Ereignisse, die nach bzw. während der Ausführung von Bearbeitungsaktionen erzeugt werden. Sobald die Layout-Engine ein Layout-Ereignis empfängt, verarbeitet sie dieses sofort. Der Prozess der Verarbeitung der Layout-Ereignisse besteht aus zwei Phasen und ist in Abbildung 4.9 schematisiert.

Je nach der Art des Layout-Ereignisses werden in der ersten Phase die Instanzen der expliziten Layout-Patterns angepasst, indem:

- neue Layout-Patterns erstellt bzw. bestehende Layout-Patterns gelöscht,
- die Referenzen auf Knoten der bestehenden Layout-Patterns modifiziert
- oder die bestehenden Layout-Patterns variiert werden.

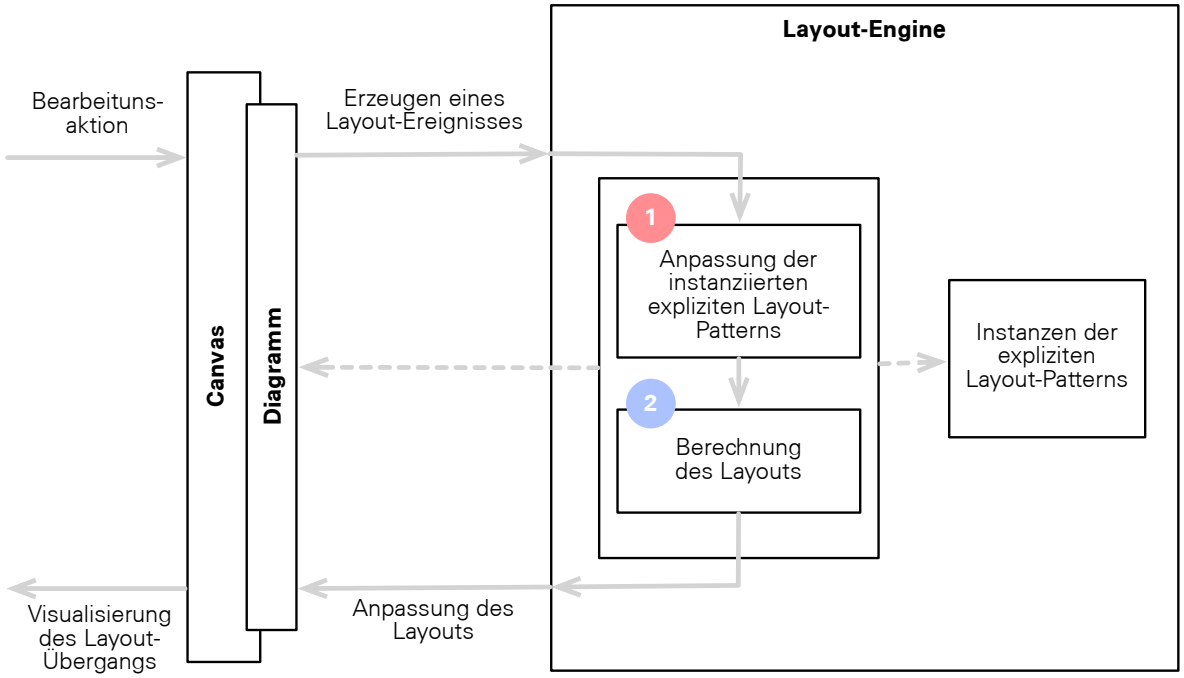


Abbildung 4.9.: Schema der Verarbeitung der Layout-Ereignisse

Insbesondere wird dabei von dem aktuellen Zustand des Diagramms, den bereits instanziierten Layout-Patterns und der Information über die Interaktion Gebrauch gemacht. Das genaue Verhalten der Anpassung der Layout-Patterns ist jedoch von einer konkreten Layout-Engine abhängig, die für den gewählten Diagrammtyp ausgelegt ist. Dieser Sachverhalt wird in Abschnitt 4.5.3 diskutiert und an konkreten Beispielen illustriert.

In der **zweiten Phase** wird das **Layout** berechnet, welches durch die impliziten und instanziierten expliziten Layout-Patterns eindeutig bestimmt ist und die Positionen für alle Knoten im Diagramm enthält[48]. Sofern sich das neu berechnete Layout von dem aktuellen unterscheidet, wird das neue Layout auf das Diagramm angewendet und dadurch ein Layout-Übergang erzeugt (siehe Abschnitt 4.3.3). Damit wird die Verarbeitung des Layout-Ereignisses abgeschlossen und der gesamte Vorgang fängt für das nächste Layout-Ereignis von vorne an.

Im Rahmen der Konzeption und Entwicklung wurde zunächst ein Algorithmus entworfen, der für die Verschiebung eines Knotens alle erreichbaren Layouts berechnet. Des Weiteren wurde der Canvas in Bereiche aufgeteilt. Dabei wurde jeder Bereich auf ein berechnetes Layout abgebildet. Während der Ausführung der Verschiebungsaktion wurde ein Layout-Übergang genau dann ausgelöst, wenn der Mauszeiger in einen anderen Bereich bewegt wurde. Da die Aufteilung der Bereiche nach jedem Layout-Übergang nicht aktualisiert wurde, war sie oft nach dem ersten Layout-Übergang ungültig. Später wurde festgelegt, dass dieser Algorithmus nicht hinreichend ist.

In dem präsentierten Ansatz wird für die Verschiebungsaktion ein Layout-Ereignis für jede Bewegung des Mauszeigers generiert und verarbeitet. Dabei sollen bei dem Entwurf eines kon-

[48] Es wäre möglich auch weitere Layout-Eigenschaften wie Routen für die Kanten oder Parameter der Layout-Übergänge zu berechnen. Dies wurde in dieser Arbeit aus zeitlichen Gründen jedoch außer Acht gelassen.

kreten Algorithmus folgende Eigenschaften beachtet werden:

- **Stabilität** Während der Ausführung der Verschiebungsaktion soll ein Layout-Übergang erst nach einer deutlichen Bewegung des Mauszeigers durch den Nutzer ausgelöst werden.
- **Erreichbarkeit des Ausgangslayouts** Das Layout, das ein Diagramm annimmt, wenn die Verschiebungsaktion beginnt, soll im Verlauf der Ausführung der Verschiebungsaktion stets erreichbar sein.

4.5.3. KONKRETE LAYOUT-ALGORITHMEN

In Abschnitt 4.5.2 wurde der Rumpf des Algorithmus für die Berechnung des Layouts vorgestellt. Dabei wurde nicht auf die interne Funktionsweise der Layout-Engine eingegangen. Das liegt daran, dass die Layout-Engine ein unterschiedliches Verhalten für verschiedene **Diagrammtypen** aufweisen soll, denn jeder Diagrammtyp (z.B. Klassendiagramm, Zustandsdiagramm, Flowchart usw.) hat aufgrund der Syntax- und Semantikregeln spezielle Anforderungen an das ästhetische Layout. Nur dadurch kann das Kriterium K.8 erfüllt werden.

Die in dieser Arbeit präsentierten Algorithmen basieren auf der visuellen Sprache der Klassendiagramme, nämlich der Notationssprache UML. Allerdings handelt es sich um keine vollständigen Algorithmen für das Layout von Klassendiagrammen, sondern um zwei relativ einfache Algorithmen, deren Aufgabe es ist, die eingeführten Mechanismen der Interaktion zu validieren und damit das gesamte Konzept zu prüfen. Daher wird der Einfachheit halber kein UML-Metamodell eingesetzt und die Syntax wird stark eingeschränkt. Es werden nur vereinfachte Klassen (Knoten) und Vererbungsrelationen (Kanten) unterstützt.

4.5.3.1. ALGORITHMUS FÜR DAS HORIZONTALE LAYOUT

Der Algorithmus für das horizontale Layout ordnet alle Klassen eines Klassendiagramms in einer horizontalen Reihe an und lässt dabei die Vererbungsrelationen außer Acht.

Die Layout-Engine, die diesen Algorithmus umsetzt, enthält intern eine Instanz des Ausrichtungspatterns (siehe Abschnitt 4.4.2.1), das alle Klassen des Diagramms referenziert. Dieser Algorithmus hat aus der Sicht des Nutzers keine große Bedeutung und dient ausschließlich als Grundlage für den Algorithmus für das baumbasierte Layout, mit dem sich Abschnitt 4.5.3.2 beschäftigt.

Nichtsdestotrotz macht der Algorithmus die Mechanismen der Interaktion aus Abschnitt 4.3 verständlich. Insbesondere zeigt die Verschiebungsaktion die Möglichkeit der Änderung der Reihenfolge der Klassen im Diagramm. Dies wird durch die Layout-Engine in ein direktes Variieren des Ausrichtungspatterns übersetzt. Die prototypische Implementierung dieses Algorithmus wird in einem Video unter dem Pfad `Prototype/Videos/Horizontal-Layout.mp4` auf der eingereichten CD veranschaulicht.

4.5.3.2. ALGORITHMUS FÜR DAS BAUMBASIERTE LAYOUT

Der Algorithmus für das baumbasierte Layout ermöglicht die Modellierung von Vererbungshierarchien in einem Klassendiagramm. Dabei hält er sich an die in Abschnitt 2.2 beschriebenen ästhetischen Prinzipien. Insbesondere wird das Prinzip der Darstellung von Hierarchien eingehalten (Prinzip ÄP.8).

Ähnlich wie bei der Umsetzung des Algorithmus für das horizontale Layout (siehe Abschnitt 4.5.3.1) enthält die Layout-Engine, die den Algorithmus für das baumbasierte Layout umsetzt, eine Instanz des horizontalen Ausrichtungspatterns, das jedoch nicht alle Klassen im Diagramm sondern nur die Wurzelklassen der Vererbungshierarchien referenziert. Durch dieses globale Ausrichtungspattern wird die Reihenfolge der Vererbungshierarchien bestimmt. Während der Erstellung von Vererbungsrelationen zwischen Klassen werden T-Shape-Patterns (siehe Abschnitt 4.4.2.2) instanziiert, die die Struktur der Vererbungshierarchien beschreiben und durch die Layout-Engine verwaltet werden. In Abbildung 4.10 wird ein Beispiel eines Diagramms mit gekennzeichneten Layout-Patterns gezeigt. Die blaue Linie stellt das globale Ausrichtungspattern dar und mit den grünen Linien sind die instanziierten T-Shape-Patterns markiert. Die Vererbungshierarchien sind mit grauen gestrichelten Linien umrandet.

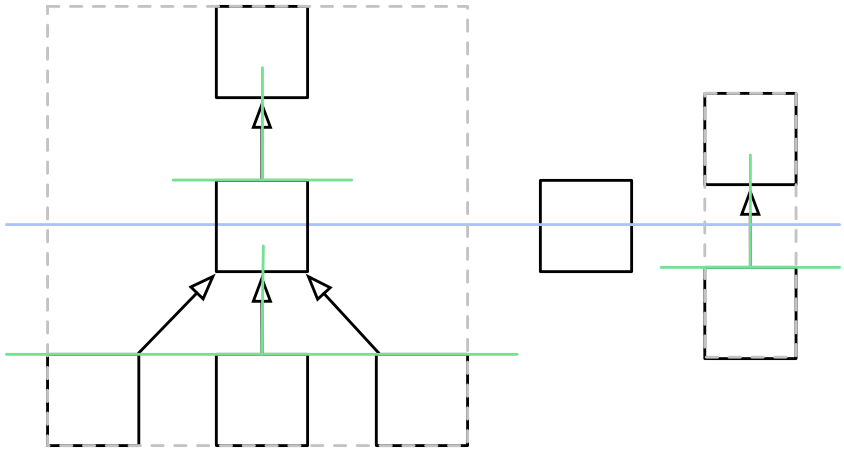


Abbildung 4.10.: Visualisierung von Instanzen der expliziten Layout-Patterns im baumbasierten Layout

Die Berechnung des endgültigen Layouts erfolgt durch die Zusammensetzung der instanziierten expliziten Layout-Patterns, wobei für diesen Algorithmus die oben beschriebene Vereinfachung der Klassendiagramme ausgenutzt wird. Somit ist der Algorithmus ausschließlich für diesen konkreten Fall geeignet. Die Möglichkeit der Entwicklung eines vollständigen Algorithmus für Klassendiagramme wird in Abschnitt 7.2.3 diskutiert.

Je nachdem welche Klasse verschoben wird, verhält sich die Verschiebungsaktion unterschiedlich. Die Verschiebung von Wurzelklassen bewirkt die Änderung der Reihenfolge von Vererbungshierarchien. Im Fall der Verschiebung einer Klasse, die sich in einer Vererbungshierarchie unter der Wurzelklasse befindet, wird die Reihenfolge der Geschwisterklassen angepasst. Dies erfolgt durch das Variieren des zugehörigen T-Shape-Patterns (siehe Abschnitt 4.4.2.2). Zum besseren Verständnis der Funktionsweise dieses Algorithmus wird auf ein Video mit

der Vorführung der prototypischen Implementierung verwiesen. Das Video ist unter dem Pfad `Prototype/Videos/Tree-Layout.mp4` auf der eingereichten CD zu finden.

4.6. ABGRENZUNG ZU BESTEHENDEN ANSÄTZEN

Im Folgenden wird der oben vorgestellte Ansatz mit bestehenden Ansätzen aus Kapitel 3 verglichen und von ihnen abgegrenzt. Dieser Abschnitt wird nach der in Abschnitt 3.1 präsentierten Kategorisierung in Ansätze für das manuelle (siehe Abschnitt 4.6.1), automatische (siehe Abschnitt 4.6.2) und interaktive halbautomatische (siehe Abschnitt 4.6.3) Layout aufgeteilt.

4.6.1. ABGRENZUNG ZU DEN ANSÄTZEN FÜR DAS MANUELLE LAYOUT

Die fundamentale Eigenschaft der Ansätze für das manuelle Layout, die in Abschnitt 3.2 vorgestellt wurden, ist die Unterstützung der Freihand-Bearbeitung. Dadurch kann der Nutzer die Layout-Eigenschaften der Objekte im Diagramm wunschgemäß anpassen. Insbesondere schließt dies die freie Positionierung ein. Die Bearbeitung erfolgt interaktiv und die Editoren stellen in der Regel zusätzliche Hilfsfunktionen für die Layout-Erstellung zur Verfügung (siehe Abschnitt 3.2.2). Obwohl der präsentierte Ansatz keine Freihand-Bearbeitung unterstützt und speziell die freie Positionierung einschränkt, werden neue Bedienungsmechanismen für die interaktive Bearbeitung des Diagramms eingeführt. Im Vergleich zu den Ansätzen für das manuelle Layout ist der präsentierte Ansatz weniger flexibel, dennoch kann das resultierende Layout durch den Nutzer beeinflusst werden, indem es variiert wird. Diese Einschränkungen in Kombination mit der halbautomatischen Berechnung führen dazu, dass der Aufwand an der Layout-Erstellung wesentlich reduziert wird. Des Weiteren ist es möglich, die syntaktischen und semantischen Regeln der konkreten Diagrammtypen durch den Layout-Algorithmus einzuhalten.

4.6.2. ABGRENZUNG ZU DEN ANSÄTZEN FÜR DAS AUTOMATISCHE LAYOUT

Da der präsentierte Ansatz eine direkte Interaktion mit dem Diagramm unterstützt, erfolgt die Eingabe ähnlich wie bei den visuellen Ansätzen für das automatische Layout (siehe Abschnitt 3.3.2) in einer visuellen Sprache. Eine weitere Gemeinsamkeit besteht darin, dass das Layout automatisch berechnet wird. Die Layout-Berechnung wird durch einen Algorithmus durchgeführt, dessen Aufgabe es ist, die Layout-Eigenschaften für die im Diagramm enthaltene Objekte zu bestimmen. Der Layout-Algorithmus in dem präsentierten Ansatz ist zusätzlich für die Verarbeitung der Interaktion ausgelegt (siehe Abschnitt 4.5.2). Dadurch ist der Ansatz im Unterschied zu den Ansätzen für das automatische Layout für interaktive Umgebungen geeignet, da er eine Anpassung des berechneten Layouts nach den Nutzerpräferenzen ermöglicht. Der Einfluss auf das Layout besteht daher in der Variation anstatt der Einstellung der Parameter des Layout-Algorithmus. Weiterhin wird durch die Layout-Übergänge das mentale Modell erhalten (siehe Abschnitt 4.3.3). Obwohl der präsentierte Ansatz eine Berechnung des initialen Layouts

unterstützt (siehe Abschnitt 4.5.1), wird die Möglichkeit der Einbindung in automatisierte Prozesse aufgrund des interaktiven Charakters ausgeschlossen.

Da sich diese Arbeit insbesondere mit den interaktiven Aspekten des Ansatzes beschäftigt, legt sie einen großen Wert auf die Einhaltung der ästhetischen Prinzipien sowohl für Graphen als auch für Klassendiagramme. Im Unterschied zu den speziellen Algorithmen für die Klassendiagramme (siehe Abschnitt 3.3.3) wird deren Unterstützung nur partiell ausgearbeitet und wird der weiteren Forschung überlassen. Diese Problematik wird in Abschnitt 7.2.3 aufgegriffen.

4.6.3. ABGRENZUNG ZU DEN ANSÄTZEN FÜR DAS INTERAKTIVE HALBAUTOMATISCHE LAYOUT

Der präsentierte Ansatz lässt sich in die Kategorie der Ansätze für das interaktive halbautomatische Layout einordnen (siehe Abschnitt 3.4) und besitzt somit viele Gemeinsamkeiten mit Ansätzen aus dieser Kategorie. Die Eigenschaft, die diese Ansätze ausmacht, ist die Unterstützung der interaktiven Bearbeitung des Diagramms. Jedoch kann das Diagramm in dem präsentierten Ansatz durch den Nutzer nicht frei bearbeitet werden. Insbesondere wird die Anpassung der Layout-Eigenschaften nicht gestattet. Dagegen werden neue Mechanismen der Interaktion eingeführt (siehe Abschnitt 4.3), die die Bearbeitungsmöglichkeiten einschränken und die Konzentration auf den Inhalt fördern.

In [Mai12, S.5] wird aufgeführt, dass die Einhaltung der ästhetischen Prinzipien im dynamischen Kontext eine geringere Rolle als die Erhaltung des mentalen Modells spielt. Der präsentierte Ansatz strebt an, diese These durch eine gleichzeitige Unterstützung beider Kriterien zu widerlegen. Die ästhetischen Prinzipien werden durch die automatische Layout-Berechnung berücksichtigt und da die Interaktion auf das Variieren des berechneten Layouts beschränkt ist, können sie durch den Nutzer nicht verletzt werden. Des Weiteren sorgen die Layout-Übergänge (siehe Abschnitt 4.3.3) für die Erhaltung des mentalen Modells.

Ähnlich wie in den strukturbasierten benutzergesteuerten Ansätzen (siehe Abschnitt 3.4.1), wird in dem präsentierten Ansatz das Layout mithilfe von Strukturregeln beschrieben, die einen persistenten Charakter haben. Obwohl der Begriff der Layout-Patterns von [Mai12] übernommen wurde, unterscheiden sich die Konzepte stark voneinander. Die konkreten Unterschiede wurden in Abschnitt 4.4 sowie 4.4.2 diskutiert.

4.7. ZUSAMMENFASSUNG

In diesem Kapitel wurde das Konzept eines neuartigen interaktiven Ansatzes für das Layout in graphbasierten Diagrammen vorgestellt. Dieser Ansatz ist insbesondere für visuelle Editoren geeignet, die eine agile Form der Softwaremodellierung unterstützen. Der Entwurf baut auf Kriterien auf, die am Anfang dieses Kapitels aufgestellt wurden.

Das entworfene Konzept schematisiert den Bearbeitungsprozess und den Ablauf der Layout-Berechnung. Da die Interaktion des Nutzers mit dem Diagramm in dem Ansatz eine wesentliche Rolle spielt und in die Layout-Berechnung einbezogen wird, wurden neue Bedienungsmechanismen entwickelt, die den präsentierten Ansatz ausmachen und ihn von anderen Ansätzen unterscheiden.

Die Layout-Berechnung wird durch einen Layout-Algorithmus ausgeführt, der im Idealfall für den Diagrammtyp angepasst ist und seine Layout-Anforderungen berücksichtigt. In diesem Kapitel wurden zwei vereinfachte Algorithmen für das Layout von eingeschränkten Klassendiagrammen präsentiert, die den Gegenstand für die prototypische Implementierung bilden. Obwohl die Syntax- und Semantik-Beschreibung nicht formalisiert wurde, schafft das allgemeine Konzept eine Grundlage für die Erweiterung um mächtigere Layout-Algorithmen. Diese Problematik wird in Abschnitt 7.2 thematisiert.

Außerdem stellt dieses Kapitel weitere Bestandteile des Konzeptes vor, die in der Layout-Berechnung eingesetzt werden, nämlich die Layout-Ereignisse für die Repräsentation der Interaktion und Layout-Patterns für die Beschreibung der Struktur des Diagramms.

Nachdem der Ansatz in diesem Kapitel entworfen wurde, folgt in Kapitel 5 dessen Umsetzung in Form eines Prototyps. Anschließend wird der Ansatz in Kapitel 6 anhand der prototypischen Implementierung evaluiert.

5. PROTOTYP-IMPLEMENTIERUNG

Um den in Kapitel 4 vorgestellten Ansatz für das interaktive Layout von Diagrammen und insbesondere die neu eingeführten Mechanismen für die Interaktion zu validieren, wurde im Rahmen der Bachelor-Arbeit ein Prototyp entwickelt. Mit der Beschreibung des Prototyps beschäftigt sich dieses Kapitel. In Abschnitt 5.1 wird auf die gewählten Technologien eingegangen. Die Benutzeroberfläche und die unterstützten Funktionen werden in Abschnitt 5.2 vorgestellt. In Abschnitt 5.3 wird die Architektur des Prototyps thematisiert und die einzelnen Komponenten beschrieben. Schließlich folgt in Abschnitt 5.4 eine Zusammenfassung.

5.1. TECHNOLOGIE

Bevor die Entwicklung des Prototyps begonnen hat, wurde nach verfügbaren Komponenten gesucht, die in dem Prototyp verwendet werden könnten. Der umzusetzende Ansatz weist spezielle Mechanismen für die Interaktion auf (siehe Abschnitt 4.3), die weder in kommerziellen Anwendungen noch in wissenschaftlichen Arbeiten gefunden wurden. Daher lässt sich für die Interaktion keine bestehende Software-Bibliothek einsetzen. Die Umsetzung der Mechanismen der Interaktion bildet deswegen einen wesentlichen Teil der Entwicklung des Prototyps.

Des Weiteren wurde der Einsatz einer Bibliothek zum Graphzeichnen bedacht. Da die in diesen Bibliotheken bereitgestellte Algorithmen in der Regel eine automatische Berechnung des Layouts durchführen, ist ein Variieren des Layouts durch den Nutzer nicht möglich (siehe Abschnitt 3.3). Dies ist ebenfalls der Fall beim Einsatz dieser Algorithmen in interaktiven Ansätzen. Insbesondere in dem Pattern-basierten Ansatz aus [Mai12] ist deutlich zu erkennen, dass sich das Layout nach der Anwendung eines Algorithmus zum Graphzeichnen auf einen gewählten Teil des Diagramms nicht modifizieren lässt (siehe Abschnitt 3.4.1.2). Das Problem des statischen Verhaltens wurde ebenfalls in Bibliotheken für das automatische Layout von Klassendiagrammen festgestellt. Da die Möglichkeit der Layout-Variation durch den Nutzer in dem präsentierten Ansatz eine große Bedeutung hat, wurde entschieden, die in Abschnitt 4.5.3 beschriebene Layout-Algorithmen eigenständig zu implementieren. Da es sich um vereinfachte

Algorithmen handelt, wurde der Einfachheit halber zu keinem **Constraintlöser** gegriffen, der z.B. in *Dunnart* eingesetzt wird (siehe Abschnitt 3.4.1.1) und in Form der Software-Bibliothek *Adaptagrams*[49] verfügbar ist. Der Einsatz eines Constraintlösers für einen verallgemeinerten Layout-Algorithmus wird in Abschnitt 7.2.3 diskutiert.

Es wäre denkbar, das **Layout-Framework aus [Mai12]** als Grundlage für die Umsetzung des Prototyps einzusetzen. Dafür wäre es notwendig, die freie Positionierung einzuschränken, z.B. durch das Einführen von impliziten Layout-Patterns. Weiterhin müssten die in [Mai12] vorgestellten allgemeinen Layout-Patterns zusammengefasst und um die Möglichkeit der Variation erweitert werden. Schließlich müsste eine Unterstützung für die Mechanismen der Interaktion geschaffen werden. Da das Layout-Framework zu der Zeit der Anfertigung dieser Bachelor-Arbeit nicht veröffentlicht wurde, war sein Einsatz ausgeschlossen.

Aufgrund der Konzentration auf die Mechanismen der Interaktion und die Bestandteile der Layout-Berechnung wurde das bearbeitete Modell einfach gehalten. Dies wurde bereits in Abschnitt 4.5.3 zur Vorstellung der konkreten Layout-Algorithmen beschrieben. Deshalb war es nicht notwendig, eine verfügbare Implementierung des **UML-Metamodells** (z.B. Ecore[50]) zu verwenden.

Da keine Komponenten wiederverwendet werden konnten, wurde eine komplett neue Desktop-Anwendung entwickelt. Es handelt sich um eine native **Mac OS X Anwendung**, die in der Programmiersprache *Objective-C* entwickelt wurde. Sie verwendet ausschließlich die Standardbibliotheken *Cocoa*[51] und *Core Animation*[52] und besitzt keine weiteren Abhängigkeiten.

5.2. FUNKTIONEN

Der Ansatz für das interaktive und diagrammspezifische Layout von Diagrammen wurde in Form eines prototypischen Werkzeugs für die Erstellung von einfachen Klassendiagrammen umgesetzt. Das Werkzeug unterstützt Klassen und Vererbungsrelationen und bietet somit die Möglichkeit, Vererbungshierarchien von Klassen zu modellieren.

Die Benutzeroberfläche des Werkzeugfensters ist in zwei rechteckige Bereiche aufgeteilt. Links befindet sich die Sidebar mit Bedienungselementen und rechts der Canvas, in dem das modellierte Diagramm abgebildet wird. Die Sidebar besteht aus einer Palette mit Icons für das Hinzufügen von Klassen und Vererbungsrelationen, einem Button für das Zurücksetzen des modellierten Diagramms und einer Auswahl des verwendeten Layout-Algorithmus. Die Benutzeroberfläche der Anwendung wird mithilfe eines Screenshots in Abbildung 5.1 veranschaulicht.

[49] `https://github.com/mjwybrow/adaptagrams`
[50] `http://www.eclipse.org/modeling/emf`
[51] `http://developer.apple.com/technologies/mac/cocoa.html`
[52] `http://developer.apple.com/technologies/mac/graphics-and-animation.html`

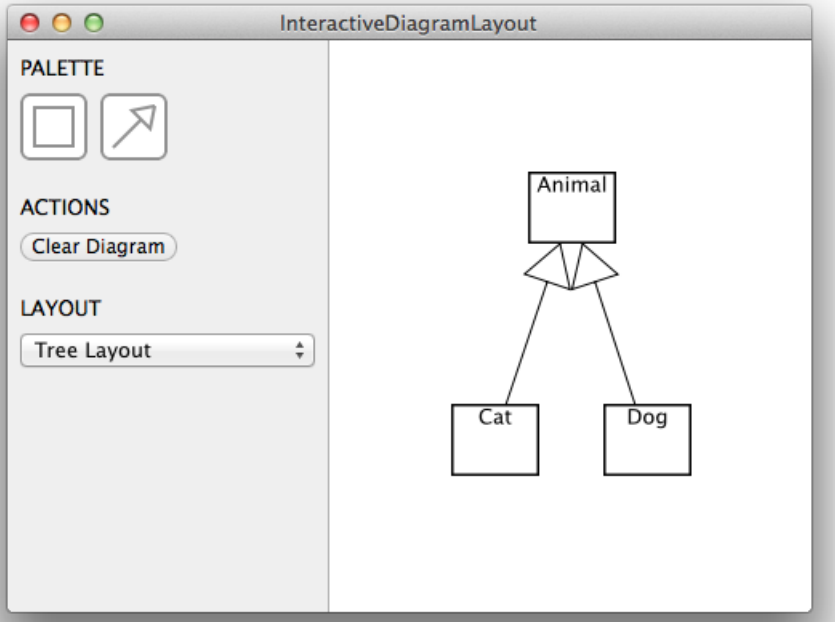


Abbildung 5.1.: Ein Screenshot des prototypischen Werkzeugs

5.2.1. UNTERSTÜTZTE AKTIONEN

Der Prototyp unterstützt die grundlegenden Aktionen, die erforderlich sind, um ein Diagramm modellieren zu können. Insbesondere handelt es sich um die wichtigsten Bearbeitungsaktionen, die in Abschnitt 4.3.1 vorgestellt wurden. Die einzelnen unterstützten Aktionen werden im Folgenden aufgelistet und kurz beschrieben:

- **Hinzufügen einer Klasse** Eine neue Klasse kann mithilfe von „Drag and Drop“ dem Diagramm hinzugefügt werden, indem das linke Icon in der Palette angeklickt und in den Canvas gezogen wird. Nachdem der Mauszeiger den Canvas erreicht, wird die Klasse hinzugefügt, was durch einen Platzhalter gekennzeichnet wird. Danach wandelt sich die Aktion in eine Verschiebungsaktion um (siehe unten), wodurch die Position der hinzugefügten Klasse bestimmt werden kann.
- **Verschiebung einer Klasse** Die Position einer Klasse im Diagramm kann durch den Nutzer geändert werden, indem die Klasse mithilfe des Mechanismus der temporären Schicht (siehe Abschnitt 4.3.2) verschoben wird. Dabei wird das Layout durch die ausgewählte Layout-Engine berechnet (siehe Abschnitt 5.2.2).
- **Umbenennen einer Klasse** Durch einen Doppelklick und das Eintragen eines Namens in das angezeigte Textfeld kann eine Klasse im Diagramm umbenannt werden. Aus zeitlichen Gründen konnte die Anpassung der Größe der Klassen nach der Änderung des Namens nicht implementiert werden.

- **Erzeugen einer Vererbungsrelation zwischen zwei Klassen** Durch das Anklicken des rechten Icons in der Palette und das Ziehen eines Pfeils zwischen zwei Klassen im Diagramm wird eine Vererbungsrelation erstellt. Alternativ kann neben dem Anklicken des Icons auch die Taste `CTRL` gedrückt werden. Wenn der Nutzer versucht, eine semantisch ungültige Vererbungsrelation zu erstellen (z.B. einen Vererbungszyklus oder eine Mehrfachvererbung[53]), wird eine Fehlermeldung angezeigt und der Vorgang bricht ab. Nachdem eine Vererbungsrelation erstellt wird und die ausgewählte Layout-Engine die Vererbungsrelationen unterstützt, wird das neu berechnete Layout auf das Diagramm angewendet.
- **Löschen des Inhalts** Da das prototypische Werkzeug keine Möglichkeit der Auswahl von Objekten im Canvas bietet, wird weder das Löschen von Klassen noch von Relationen unterstützt. Um trotzdem das Zurücksetzen des Diagramms zu ermöglichen, gibt es für diese Funktion ein Button in der Sidebar.

Die Funktionsweise der beschriebenen Aktionen ist in einem Video unter dem Pfad `Prototype/Videos/Actions.mp4` auf der eingereichten CD veranschaulicht.

Die implementierten Aktionen sind für die Validierung des umgesetzten Konzeptes und die Vorbereitung von Aufgaben für eine Nutzerstudie ausreichend, dennoch ist das prototypische Werkzeug nicht produktiv einsetzbar. Das liegt zum einen auf der starken Einschränkung der Notation der Klassendiagramme und zum anderen auf der ausschließlichen Unterstützung von grundlegenden Aktionen. Die möglichen Erweiterungen des Prototyps werden in Abschnitt 7.2.1 diskutiert.

5.2.2. UNTERSTÜTZTE LAYOUT-METHODEN

Der Prototyp unterstützt beide konkreten Layout-Algorithmen aus Abschnitt 4.5.3, wobei der Algorithmus für das horizontale Layout als ein Zwischenschritt für die Implementierung des Algorithmus für das baumbasierte Layout dient und bei der Layout-Berechnung keine Relationen berücksichtigt.

Der aktuelle Algorithmus kann in der Sidebar ausgewählt werden und wird für die Layout-Berechnung im Diagramm verwendet. Außerdem wird bei der Änderung des Algorithmus ein initiales Layout berechnet, um das Layout des Diagramms in einen validen Zustand zu bringen (siehe Abschnitt 4.5.3). Für die Auswahl des Algorithmus können alternativ auch die Tastenkombinationen `CMD+1` und `CMD+2` genutzt werden.

[53] Die Mehrfachvererbung ist ein gültiges Konstrukt in Klassendiagrammen der Notationssprache UML, wird aber in meisten objektorientierten Sprachen nicht unterstützt [AN05]. Für die Zwecke des Prototyps wird sie nicht zugelassen.

5.3. ARCHITEKTUR

Der Prototyp wurde in Form einer objektorientierten Anwendung implementiert und ist in vier logische Komponenten aufgeteilt: *Layout Engine*, *Dragging Manager*, *Canvas View* und *Application*. Die Abhängigkeiten zwischen den Hauptkomponenten sind in Abbildung 5.2 in Form eines Paketdiagramms dargestellt. Zu den Verantwortlichkeiten der Komponente *Layout Engine* gehört u.a. die Beschreibung des Metamodells, die Verarbeitung der Nutzer-Aktionen und vor allem die Durchführung der Layout-Berechnung. Durch die Komponente *Dragging Manager* wird eine abstrahierte Schnittstelle für die „Drag and Drop" Operation bereitgestellt. Die Darstellung und Interaktion mit dem Diagramm im Canvas wird in der Komponente *Canvas View* realisiert. Schließlich werden die genannten Komponenten von der Komponente *Application* zu einer lauffähigen Anwendung zusammengesetzt. Auf die wichtigsten Komponenten wird in den folgenden Abschnitten im Detail eingegangen.

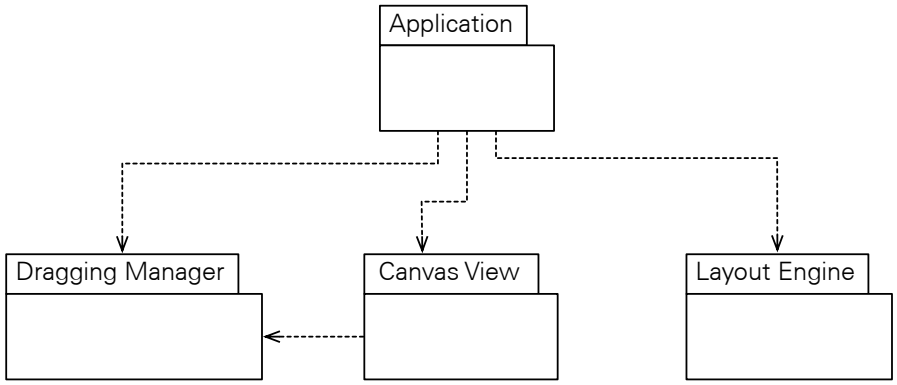


Abbildung 5.2.: Eine Übersicht der grundlegenden Komponenten des Prototyps

5.3.1. LAYOUT ENGINE

Die Komponente *Layout Engine* bildet den wesentlichen Teil der Umsetzung des interaktiven Ansatzes aus Kapitel 4. Neben der Beschreibung der Layout-Eigenschaften und des Metamodells für die Diagramme verfügt diese Komponente über die Verwaltung von Layout-Patterns, Erstellung und Verarbeitung von Layout-Ereignissen und schließlich stellt sie die Funktion der Layout-Berechnung zur Verfügung. Aufgrund der Komplexität wurde *Layout Engine* in folgende Unterkomponenten aufgeteilt: *Geometry*, *Diagram*, *Layout*, *Patterns*, *Events* und *Engines*. Eine Übersicht der Unterkomponenten und deren Beziehungen wird in Form eines Paketdiagramms in Abbildung 5.3 gegeben.

5.3.1.1. GEOMETRY

Die Komponente *Geometry* stellt die grundlegenden geometrischen Datentypen wie Punkt, Größe, Rechteck und Linie und für diese ausgelegte geometrische Funktionen bereit. Die neu eingeführten Datentypen unterscheiden sich von den entsprechenden Datentypen aus der

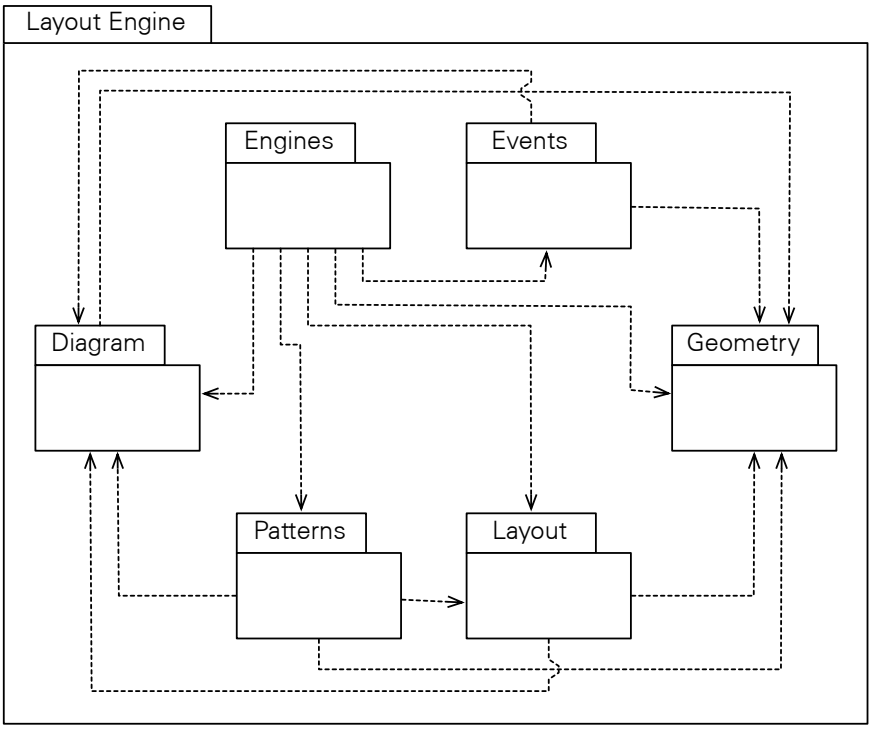


Abbildung 5.3.: Eine Übersicht der Unterkomponenten von *Layout Engine*

Standardbibliothek wie `NSPoint`, `NSSize` und `NSRect` darin, dass die Werte in ganzen Zahlen angegeben werden und dass die Position des Rechtecks seinen Mittelpunkt darstellt. Durch die letztere Eigenschaft wird die Angabe der Positionen mithilfe des Mittelpunkts ermöglicht. Außerdem werden die Positionen in zentrierten Koordinaten angegeben, was die Umsetzung des impliziten Patterns zur Zentrierung des Diagramm-Inhalts fördert (siehe Abschnitt 4.4.1.3) und die relative Layout-Berechnung vereinfacht. Da in *Cocoa* und insbesondere in der für die grafische Repräsentation verantwortlichen Klasse `NSView` der Koordinatenursprung standardmäßig in der linken unteren Ecke liegt, war es notwendig eine Umrechnung der beiden Koordinatensysteme zu implementieren. In Abbildung 5.4 ist der Sachverhalt illustriert, wobei die zentrierten Koordinaten blau und die Standardkoordinaten rot dargestellt sind. Die Zeichnung wird mit einem schwarzen Referenzrechteck eingeschlossen, das für die Umrechnung notwendig ist.

5.3.1.2. DIAGRAM

Das **Metamodell der Diagramme** ist ein Bestandteil der Komponente *Diagram* und wird durch die Klassen `IDLDiagram`[54], `IDLNode` und `IDLEdge` repräsentiert. Durch deren Instanziierung wird ein Modell eines Klassendiagramms mit Klassen und Vererbungsrelationen erstellt. Die Klasse `IDLNode` beschreibt weiterhin ihre Position im Diagramm. Dadurch wird gewährleistet, dass das Modell auch die Information über das aktuelle Layout enthält.

Aus zeitlichen Gründen konnte diese Komponente nicht tiefgründig ausgearbeitet werden. Daher wird die abstrakte und konkrete Syntax der Elemente zusammengefasst und deren Se-

[54] Das Präfix `IDL` steht für den Projektnamen *Interactive Diagram Layout* und wird in *Objective-C* Projekten für Klassen- und Funktionsnamen verwendet, um Namenskonflikte zu vermeiden.

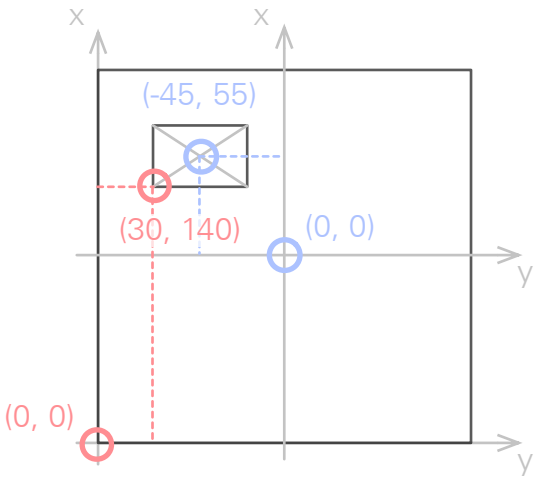


Abbildung 5.4.: Ein Beispiel der Koordinatenumrechnung für ein rechteckiges Objekt mit einer festen Größe

mantik nur unvollständig unterstützt. Die mögliche Behebung dieser Schwachstelle wird in Abschnitt 7.2.2 diskutiert.

5.3.1.3. LAYOUT

Bei der Berechnung des Layouts ist es notwendig, die **Layout-Eigenschaften beschreiben** und auf ein Diagramm anwenden zu können. Diese Aufgaben übernimmt die Komponente *Layout*, die die Klassen `IDLLayout` und `IDLLayoutApplier` zur Verfügung stellt. Die zuerst genannte Klasse beschreibt eine Abbildung von Klassen auf deren Positionen im Diagramm, ohne die eigentlichen Positionen der Klassen zu verändern. Weiterhin bietet sie die Möglichkeit der Layout-Komposition mithilfe einer relativen Zusammensetzung von mehreren Layouts. Um ein gegebenes Layout auf die abgebildeten Klassen anzuwenden, wird die zweitgenannte Klasse eingesetzt.

5.3.1.4. PATTERNS

Die Umsetzung der Layout-Patterns aus Abschnitt 4.4 erfolgt auf zwei Wegen. Die impliziten Layout-Patterns werden nicht instanziiert und müssen von konkreten Layout-Engines eingehalten werden (siehe Abschnitt 5.3.1.6). Dagegen werden von den expliziten Layout-Patterns Instanzen erstellt und in der Layout-Engine verwaltet. Zu diesem Zweck stellt die Komponente *Patterns* zwei Klassen für **konkrete explizite Layout-Patterns**[55] zur Verfügung, nämlich die Klasse `IDLAlignmentPattern` für das Ausrichtungspattern und die Klasse `IDLTShapePattern` für das T-Shape-Pattern. Beide Klassen werden in einem Klassendiagramm[56] in Abbildung 5.5 visualisiert.

[55] Die implementierten expliziten Layout-Patterns unterstützen ausschließlich die Ausrichtungen, die für die Umsetzung der Layout-Algorithmen notwendig sind.

[56] Da die Programmiersprache *Objective-C* spezielle Konstrukte aufweist, ist eine standardkonforme Abbildung von *Objective-C* Klassen in Klassendiagrammen der Notationssprache UML nicht möglich.

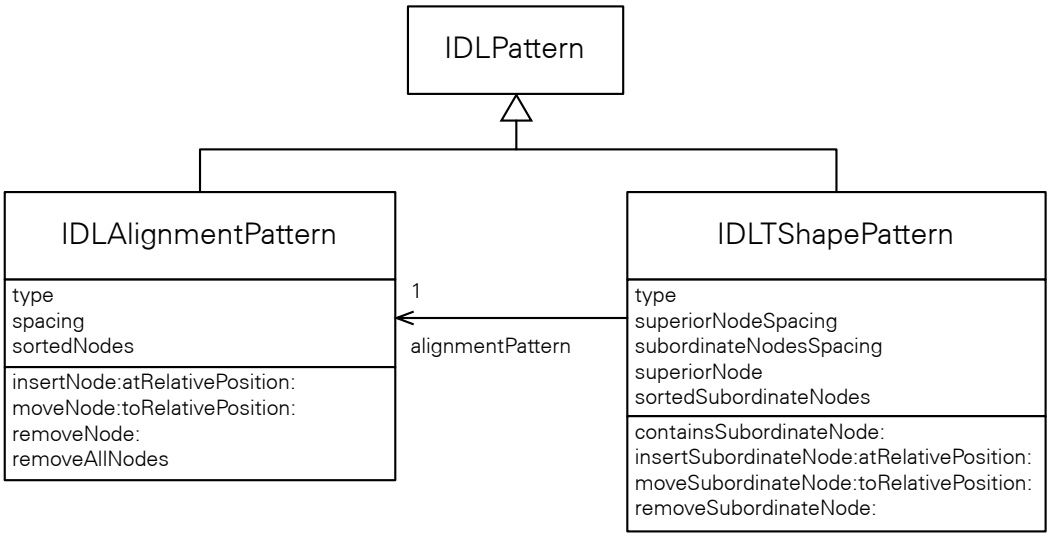


Abbildung 5.5.: Ein Klassendiagramm mit einer Übersicht der expliziten Layout-Patterns

Die Klassen für die konkreten expliziten Layout-Patterns beschreiben ihre geometrischen Eigenschaften, ermöglichen die Verwaltung von verknüpften Knoten[57] und berechnen das relative Layout für diese. Weiterhin besitzen die Klassen Methoden für die Verschiebung von einem verknüpften Knoten an eine relative Position, um die Reihenfolge der Knoten zu verändern. Dadurch wird das Variieren der Layout-Patterns erreicht.

Ein weiterer interessanter Aspekt der implementierten Layout-Patterns ist die Möglichkeit ihrer Wiederverwendung und Komposition. Dem Klassendiagramm in Abbildung 5.5 ist zu entnehmen, dass die Klasse `IDLShapePattern` intern eine Instanz der Klasse `IDLAlignmentPattern` verwendet. Diese ist für die Verwaltung der untergeordneten Knoten zuständig.

5.3.1.5. EVENTS

Die Komponente *Events* stellt Klassen für die **Beschreibung von Layout-Ereignissen** zur Verfügung, deren Konzept in Abschnitt 4.5 erläutert wurde. Im Prototyp wurden die Layout-Ereignisse implementiert, die den unterstützten Bearbeitungsaktionen aus Abschnitt 5.2.1 entsprechen. Alle Klassen für konkrete Layout-Ereignisse erben von der Klasse `IDLLayoutEvent` und sind in einem Klassendiagramm in Abbildung 5.6 veranschaulicht.

Obwohl der Prototyp keine Möglichkeit des Löschens von einzelnen Knoten bietet, wird trotzdem das Layout-Ereignis `IDLNodeDeletionLayoutEvent` unterstützt. Es wird genau dann erzeugt und verarbeitet, wenn während der Hinzufügung eines Knotens der Mauszeiger außerhalb des Canvas bewegt wird. Dadurch wäre mit der Implementierung der Auswahl von einzelnen Knoten die Funktion des Löschens automatisch gewährleistet.

[57] Die Programmierschnittstellen im Prototyp verwenden die Begriffe *Knoten* (engl.: „node“) und *Kante* (engl.: „edge“) anstatt von *Klasse* und *Vererbungsrelation*.

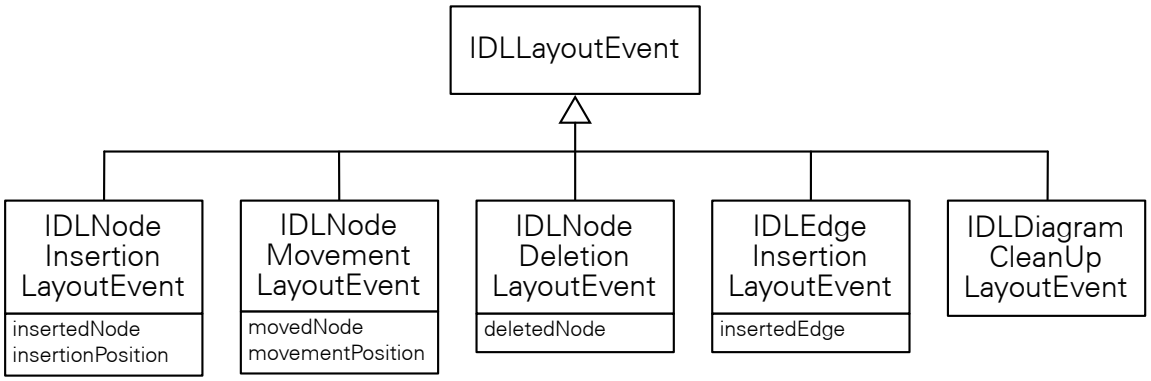


Abbildung 5.6.: Ein Klassendiagramm mit Übersicht der unterstützten Layout-Ereignisse

5.3.1.6. ENGINES

Alle oben aufgeführten Komponenten werden durch die Komponente *Engines* zusammengesetzt, um die eigentliche Layout-Berechnung zu realisieren. Zum einen verfügt die Komponente über eine abstrakte Klasse `IDLLayoutEngine` und zum anderen über zwei konkrete Unterklassen `IDLHorizontalLayoutEngine` und `IDLTreeLayoutEngine`, die die Layout-Algorithmen aus Abschnitt 4.5.3 umsetzen. Wie bereits in Abschnitt 5.2.2 beschrieben wurde, kann die Layout-Engine zur Laufzeit des Prototyps gewählt werden.

Die konkreten Layout-Engines besitzen eine Referenz auf den Inhalt des Diagramms und müssen daher nach jeder Änderungen im Diagramm mithilfe von Layout-Ereignissen synchronisiert werden, um das Layout aktuell zu halten. Neben der Fähigkeit, alle Typen der Layout-Ereignisse aus Abschnitt 5.3.1.5 zu verarbeiten, sind die Layout-Engines in der Lage, das initiale Layout zu berechnen, u.a. durch Instanziierung von expliziten Layout-Patterns, die in den Layout-Engines verwaltet werden. Die Klasse `IDLHorizontalLayoutEngine` besitzt nur eine Instanz des Ausrichtungspatterns. Dagegen werden in der Klasse `IDLTreeLayoutEngine` während der Laufzeit die Layout-Patterns dynamisch erzeugt und gelöscht. Bevor die Layout-Patterns verwendet werden können, müssen ihre geometrischen Parameter konfiguriert werden (siehe Abschnitte 4.4.2 und 5.3.1.4). Die in den Implementierungen der konkreten Algorithmen verwendete Werte wurden beispielhaft gewählt. Idealerweise sollten die passenden Werte für die Algorithmen anhand von geeigneten Methoden ermittelt und festgelegt werden, wobei es nicht ausgeschlossen wird, die Einstellung von einigen Eigenschaften des Algorithmus dem Nutzer zu überlassen, um den Algorithmus nach seinen Wünschen zu konfigurieren.

Das Layout wird in dem Algorithmus durch die Zusammensetzung der instanziierten Layout-Patterns berechnet. Dies ist nur möglich, weil die Algorithmen für die festgelegte Struktur der Diagramme angepasst sind. Eine Verallgemeinerung der Layout-Algorithmen u.a. durch den eventuellen Einsatz eines Constraintlösers wird in Abschnitt 7.2.3 diskutiert.

5.3.2. CANVAS VIEW

Die Komponente *Canvas View* wird in die Präsentationsschicht des Design Patterns *Model View Controller* eingeordnet und ist für die **visuelle Repräsentation des Diagramms** und die **Interaktion im Canvas** zuständig. Den Kern dieser Komponente bildet die Klasse `IDLCanvasView`, die ihre Oberklasse `NSView` um die Möglichkeit des Zeichnens von Diagrammen erweitert. Dies wird mithilfe von der Zusammensetzung von Layer aus dem Framework *Core Animation* realisiert. Die Informationen über die einzelnen Elemente und ihre Layout-Eigenschaften werden mittels des Delegate-Patterns abgefragt, sodass die Komponente vom Rest der Anwendung und insbesondere von der Layout-Berechnung abgekoppelt bleibt.

Obwohl die implementierten Layout-Engines ausschließlich die Positionen der Klassen berechnen, werden von `IDLCanvasView` alle Layout-Eigenschaften benötigt, um die Klassen und Vererbungsrelationen darstellen zu können, d.h. auch die Größe der Klassen und die Start- bzw. Endpunkte der Vererbungsrelationen. Im Prototyp wird einfachheitshalber die Größe der Klassen konstant gehalten und die Start- bzw. Endpunkte ähnlich wie in *OmniGraffle* (siehe Abschnitt 3.2.2.6) anhand der Eigenschaften der verbundenen Klassen außerhalb dieser Komponente berechnet. Da der Koordinatenursprung in `IDLCanvasView` in der linken unteren Ecke liegt, müssen alle Koordinaten zuerst umgerechnet werden (siehe Abschnitt 5.3.1.1). Wenn die Größe des Fensters geändert wird, erfolgt die Umrechnung erneut, sodass der Inhalt des Diagramms stets zentriert bleibt.

Die Interaktion mit dem Canvas besteht in der Ausführung von Bearbeitungsaktionen, die in dieser Komponente aus der Eingabe durch die Maus und Tastatur gebildet werden. Die Anwendung wird über die Ausführung jeder Bearbeitungsaktion informiert und erzeugt für sie ein oder mehrere Layout-Ereignisse. Diese werden anschließend an die Layout-Engine übergegeben. Sobald sich das Layout des Diagramms in der Layout-Engine ändert, wird darüber *Canvas View* benachrichtigt, was zum Auslösen eines Layout-Übergangs führt. Wie bereits in Abschnitt 4.3.3 beschrieben wurde, wird an dieser Stelle eine Animation eingesetzt. Konkret handelt es sich um die implizite Animation der Properties der Layer, die durch das Framework *Core Animation* bereitgestellt wird. Um die Animation flüssiger zu gestalten, wäre es denkbar, fortgeschrittene Mechanismen der Animation wie z.B. das Animations-Framework POP[58] einzusetzen.

5.4. ZUSAMMENFASSUNG

Das in Kapitel 4 entworfene Konzept für das Layout von Diagrammen wurde in Form eines Werkzeugs zur Erstellung von vereinfachten Klassendiagrammen prototypisch umgesetzt.

Es handelt sich um eine objektorientierte Anwendung für das Betriebsystem *Mac OS X*, die in der Programmiersprache *Objective-C* geschrieben ist. Da in der Vorbereitungsphase festgestellt wurde, dass für die Umsetzung des Konzeptes keine bestehende Bibliothek eingesetzt

[58] `https://github.com/facebook/pop`

werden kann, wurde entschieden, eine ganz neue Anwendung zu implementieren. Diese verwendet ausschließlich die Standardbibliotheken der Mac Plattform. Der Quellcode sowie die kompilierte Version der Anwendung sind auf der beigelegten CD zu finden (für die konkreten Pfade siehe Abschnitt B.2).

Das implementierte Werkzeug unterstützt Basisfunktionen, die für die Erstellung und Bearbeitung eines vereinfachten Klassendiagramms notwendig sind. Dies ermöglicht das praktische Testen des vorgestellten Konzeptes in einem wirklichkeitsnahen Anwendungsfall. Dennoch konnten in der initialen Version des Prototyps viele nützliche Funktionen aus zeitlichen Gründen nicht implementiert werden. Des Weiteren wurden in dem Prototyp beide konkreten Algorithmen aus Abschnitt 4.5.3 umgesetzt, zwischen denen zur Laufzeit umgeschaltet werden kann.

Während der Entwicklung der Anwendung wurde auf eine saubere architektonische Trennung geachtet. Die Architektur ist in mehrere Komponenten aufgeteilt, deren Abhängigkeiten untereinander minimiert sind. Insbesondere ist die Layout-Berechnung in der Komponente *Layout Engine* von dem Rest der Anwendung gekapselt und in weitere Unterkomponenten aufgeteilt. Die entworfene Mechanismen der Interaktion wurden in der Komponente *Canvas View* umgesetzt. Eine detaillierte Beschreibung der Architektur kann in diesem Kapitel nachgelesen werden.

Der wesentliche Teil des entworfenen Konzeptes konnte erfolgreich in dem Prototyp umgesetzt werden. Mit einer gründlichen Evaluation beschäftigt sich allerdings Kapitel 6. Unter anderem werden in dem Kapitel Ergebnisse einer Nutzerstudie präsentiert, für die der implementierte Prototyp die Grundlage bildet.

6. EVALUATION

Den Gegenstand dieses Kapitels bildet eine Evaluation des umgesetzten Konzeptes, die aus zwei Teilen besteht. Zum einen basiert sie auf einer Nutzerstudie, deren Beschreibung und Auswertung sich Abschnitt 6.1 widmet. Zum anderen wird in Abschnitt 6.2 gezeigt, inwieweit die für das Konzept aufgestellten Kriterien eingehalten wurden. Anschließend werden die Inhalte dieses Kapitels in Abschnitt 6.3 zusammengefasst.

6.1. NUTZERSTUDIE

Um das entwickelte Konzept zu validieren und seine eventuellen Probleme zu entdecken, wurde im Rahmen der Bachelor-Arbeit eine Nutzerstudie durchgeführt. Dafür wurde nach [Nie09] eine kleine Nutzergruppe gewählt, die Aufgaben in dem umgesetzten Prototyp erfüllt und anschließend die Usability evaluiert hat. Im Detail werden der Aufbau und die Durchführung der Nutzerstudie in Abschnitt 6.1.1 beschrieben. In Abschnitt 6.1.2 folgt eine Auswertung der Ergebnisse.

6.1.1. AUFBAU UND DURCHFÜHRUNG

Die Nutzerstudie erfolgte in individuellen Sitzungen mit einzelnen Teilnehmern. Jede Sitzung hat mit einer kurzen Beschreibung des Themas der Bachelor-Arbeit angefangen. Danach wurde der Ablauf der Sitzung vorgestellt und der Teilnehmer wurde um das Ausfüllen des ersten Teils des vorgefertigten Fragebogens (siehe Abschnitt A.2) gebeten, in dem allgemeine Informationen und Vorkenntnisse abgefragt wurden. Nach Bedarf wurde anschließend eine kurze Einführung über Klassendiagramme gegeben, um die Notation und die wichtigsten Begriffe wie Klasse, Vererbung, Oberklasse und Unterklasse zu erklären.

Die eigentlichen Tests des entwickelten Prototyps (siehe Kapitel 5) wurden an einem *MacBook Pro* durchgeführt. Als Eingabegerät stand den Teilnehmern das in dem Notebook eingebaute

Trackpad, eine *Apple Magic Mouse*[59] und eine klassische optische Maus zur Verfügung. Um potenzielle Probleme mit der Eingabe zu vermeiden, konnten die Teilnehmer das passende Eingabegerät frei wählen.

Als Nächstes wurde eine kurze Einführung zu dem Prototyp gegeben. Die unterstützten Aktionen wie das Erstellen bzw. Umbenennen einer Klasse, das Erstellen einer Vererbungsrelation und Löschen des Diagramms wurden vorgestellt und im Prototyp demonstriert. Insbesondere wurden die zwei Möglichkeiten zur Erstellung der Vererbungsrelation gezeigt, nämlich durch das Anklicken des Icons in der Sidebar und durch das Drücken der der Taste `CTRL` (siehe Abschnitt 5.2.1). Nach dieser Einführung durfte der Teilnehmer den Prototyp für ca. zwei Minuten ausprobieren.

Der eigentliche Test bestand darin, vier verschiedene Aufgaben zur Modellierung von Klassendiagrammen mit Vererbungshierarchien zu lösen. Alle Aufgaben benötigten das baumbasierte Layout, sodass die Layout-Engine nicht geändert werden musste. Dem Teilnehmer wurden nacheinander Blätter mit den einzeln ausformulierten Aufgabenstellungen vorgelegt, die von ihm zu lösen waren. Zum einen handelte es sich um visuelle Aufgaben, in denen der Teilnehmer ein bestehendes Diagramm rekonstruieren sollte. Zum anderen hatten die Aufgaben die Form einer textuellen Beschreibung. Alle Aufgabenblätter sind im Anhang in Abschnitt A.1 abgebildet.

Die Teilnehmer wurden gebeten, während der Erfüllung der Aufgaben ihr Vorgehen laut zu beschreiben. Dabei wurden die wichtigsten Beobachtungen notiert. Weiterhin wurde der Bildschirm aufgenommen, um eventuelle weitere Analysen zu ermöglichen. Die Videodateien sind auf der beigelegten CD zu finden (für das Dateienverzeichnis siehe Abschnitt B.3).

Im Anschluss wurde das Konzept durch den Teilnehmer bewertet, indem der verbleibende Teil des Fragebogens (siehe Abschnitt A.2) ausgefüllt wurde. Obwohl die Teilnehmer in den letzten Fragen des Fragebogens eine Möglichkeit hatten, ein allgemeines Feedback zu geben, wurde in einzelnen Fällen eine Diskussion bevorzugt. Dabei wurden die wichtigsten Punkte aufgeschrieben und ebenfalls für die Auswertung der Nutzerstudie verwendet.

6.1.2. AUSWERTUNG

An der Nutzerstudie haben insgesamt sieben Testpersonen teilgenommen und beide Geschlechter wurden ungefähr gleichmäßig vertreten (vier männliche und drei weibliche Teilnehmer). Großteils handelte es sich um Studenten und wissenschaftliche Arbeiter zwischen 21 und 34 Jahren. Des Weiteren waren sechs von sieben Teilnehmern im Bereich der Informatik tätig und waren daher mit der Notation von Klassendiagrammen vertraut.

Vor dem Test wurden die Teilnehmer in dem Fragebogen (siehe Abschnitt A.2) befragt, welche Werkzeuge sie zur Erstellung von Diagrammen benutzen und in welchem Maß. Die kompletten Ergebnisse sind in Form von Kreisdiagrammen in Abbildung 6.1 veranschaulicht. Es stellt sich heraus, dass die Teilnehmer Stift und Papier zum Zeichnen von Diagrammen am häufigsten

[59] `https://www.apple.com/magicmouse`

verwenden. Das kann dadurch begründet werden, dass dies oft die einfachste Methode ist [Amb02]. Die Softwarewerkzeuge werden in einem kleineren Maß eingesetzt, dennoch ist deren Nutzung von Bedeutung. Überwiegend werden die Diagramme in Präsentationsprogrammen gezeichnet, es werden aber auch Diagramm-Programme und UML-Editoren zum Einsatz gebracht. Dagegen sind die Ergebnisse der Nutzung von Online-Tools eher gering.

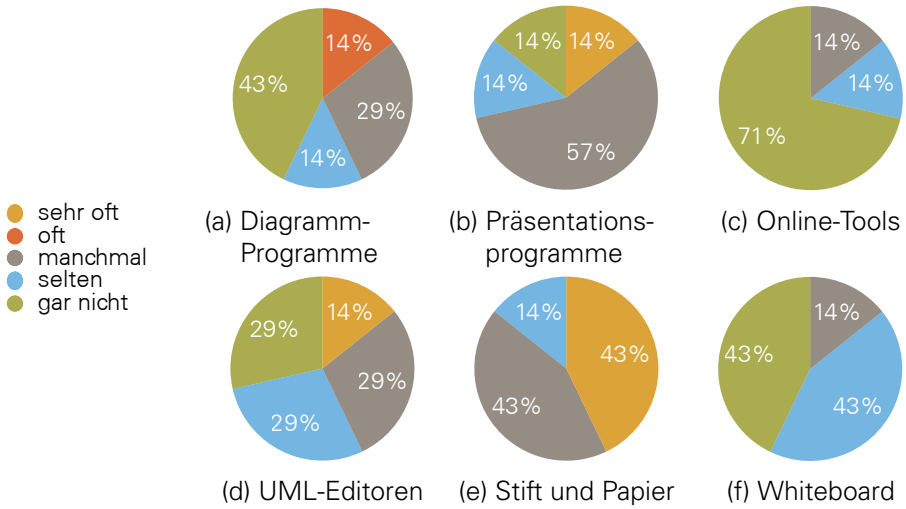


Abbildung 6.1.: Auswertung der Verwendung von Tools zur Erstellung von Diagrammen

Nachdem die Tests durchgeführt wurden, haben die Teilnehmer das Konzept anhand von aufgestellten Aussagen bewertet. Die Ergebnisse sind in Abbildung 6.2 visualisiert.

Laut der Angaben der Teilnehmer konnten die Aufgaben in allen Fällen erfolgreich erfüllt werden (siehe Abbildung 6.2(a)). Weiterhin wurden die Aufgaben als nicht anstrengend eingeschätzt (siehe Abbildung 6.2(b)). Dies deutet darauf hin, dass für die Nutzerstudie angemessene Aufgaben gewählt wurden.

Obwohl es eine Einführung zu dem Prototyp gab (siehe Abschnitt 6.1.1), wurden die Teilnehmer gefragt, ob sie die Aufgaben auch ohne diese Einführung erledigen könnten. Dies wurde zum großen Teil bestätigt, dennoch waren die Ergebnisse nicht ganz eindeutig (siehe Abbildung 6.2(c)). Bei der Vorbereitung des Aufbaus wurde überlegt, die Einführung wegzulassen. Da es sich aber bei den Softwaremodellierungswerkzeugen um ein Szenario handelt, in dem in der Regel das Anlernen der Bedienung und der Funktionen notwendig ist, wurde die Variante mit der Einführung bevorzugt.

Da das umgesetzte Konzept viele neue Interaktionsparadigmen einsetzt, hat die Interaktion einen wesentlichen Teil der Bewertung gebildet. Dadurch wurde ermöglicht, die Erfüllung des Kriteriums der Benutzerfreundlichkeit zu beurteilen (siehe Abschnitt K.9).

Mehr als die Hälfte der Teilnehmer waren der Meinung, dass sie den Prototyp während der Erfüllung der Aufgaben unter Kontrolle hatten (siehe Abbildung 6.2(d)). Weiterhin konnte beobachtet werden, dass die auf „Drag and Drop“ basierende Verschiebungsaktion, die den in Abschnitt 4.3.2 vorgestellte Mechanismus der temporären Schicht einsetzt, als intuitiv angesehen wurde. Dies wurde auch durch eine überwiegende Zustimmung im Fragebogen bestätigt

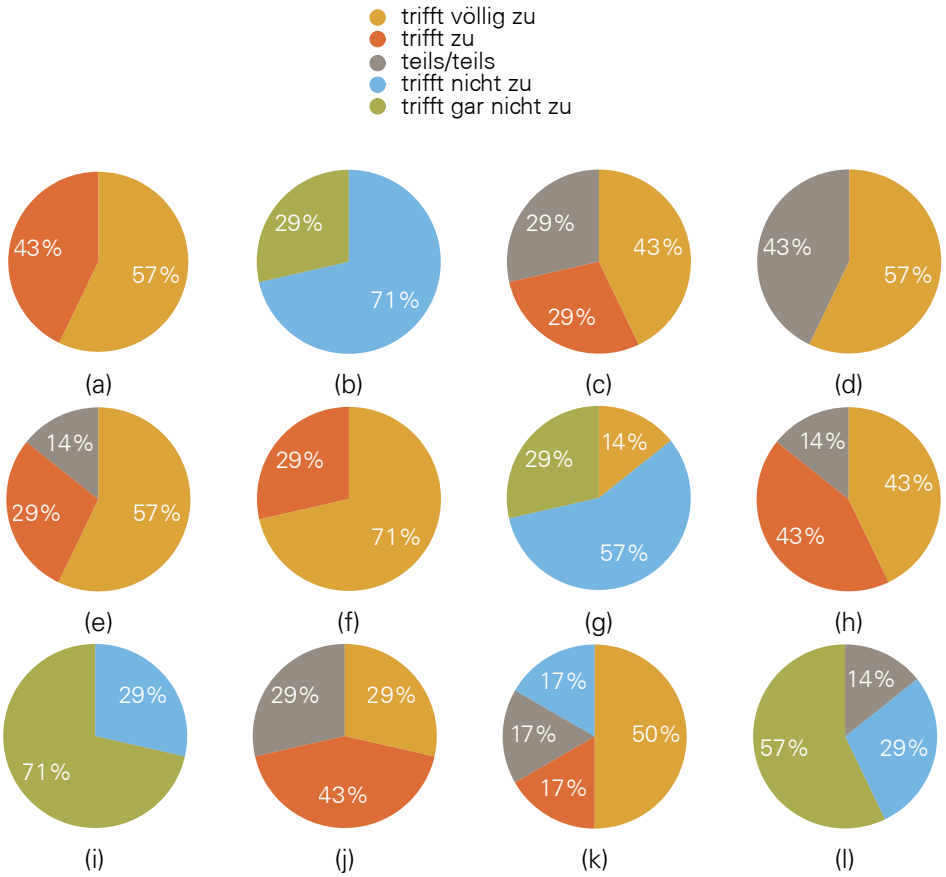


Abbildung 6.2.: Bewertung der Aussagen bezüglich der Evaluation des Konzeptes
(a) Ich habe meiner Meinung nach alle Aufgaben erfolgreich erfüllt.
(b) Die Erfüllung der Aufgaben war anstrengend.
(c) Ich denke, dass ich die Aufgaben auch ohne vorherige Vorstellung der möglichen Aktionen des Prototyps problemlos lösen könnte.
(d) Ich hatte das Programm die ganze Zeit unter Kontrolle.
(e) Die auf „Drag and Drop“ basierende Verschiebungsaktion war verständlich.
(f) Durch die Einschränkung der freien Positionierung wurde der Aufwand an der Erstellung des Layouts reduziert.
(g) Die Einschränkung der freien Positionierung fand ich irritierend.
(h) Für mich war jederzeit erkennbar, an welche Stelle ich die angeklickte Klasse verschieben kann.
(i) Die automatischen Layout-Updates haben mich in der Erstellung des Diagramms gehindert.
(j) Das Layout der modellierten Diagramme fand ich ästhetisch ansprechend.
(k) Ich konnte Fehler, die während der Erfüllung der Aufgaben gemacht wurden, korrigieren.
(l) Ich bin der Meinung, dass ich die Aufgaben in einer Software ohne Layout-Unterstützung schneller erledigen könnte.

(siehe Abbildung 6.2(e)).

Da der umgesetzte Ansatz keine freie Positionierung bietet, die von anderen Werkzeugen bekannt ist, war die Einschränkung der freien Positionierung der Gegenstand der weiteren Bewertung. Die Mehrheit der Teilnehmer hat angegeben, dass die Einschränkung der freien Positio-

nierung den Aufwand an der Erstellung des Layouts reduziert (siehe Abbildung 6.2(f)). Bis auf eine Ausnahme wurden die Teilnehmer durch diesen ungewöhnlichen Aspekt der Interaktion nicht verwirrt (siehe Abbildung 6.2(g)). Diese Resultate beweisen, dass die Einschränkung der freien Positionierung kein Problem dargestellt hat. An dieser Stelle ist allerdings auf die Vereinfachung der gewählten Aufgaben hinzuweisen. Um eine objektive Aussage über die Eignung dieses Verhaltens zu gewinnen, müsste das Konzept für einen allgemeineren Fall erweitert werden, z.B. durch die vollständige Unterstützung der Klassendiagramme.

Infolge des Einsatzes des baumbasierten Layouts hatte die Verschiebungsaktion zwei Funktionen. Einerseits konnte eine gesamte Vererbungshierarchie durch das Ziehen der Wurzelklasse verschoben werden. Anderseits konnte die Reihenfolge der Geschwisterklassen mithilfe der Verschiebungsaktion variiert werden. Obwohl die möglichen Zielpositionen der Verschiebung zum großen Teil für die Teilnehmer erkennbar waren (siehe Abbildung 6.2(h)), sind die Ergebnisse nicht überzeugend. Deshalb sollte in einer eventuellen Weiterentwicklung eine Visualisierung der Verschiebungsoptionen bedacht werden.

Die automatischen Layout-Updates haben kein Hindernis in der Erstellung des Diagramms dargestellt (siehe Abbildung 6.2(i)). Dagegen wurde die Ästhetik des Layouts größtenteils als ansprechend empfunden (siehe Abbildung 6.2(j)). Die möglichen Ursachen für eine nicht überzeugende Zustimmung sind auf die Überlappung der Pfeilspitzen bei der Vererbungsrelationen und die Umbrüche der Klassennamen zurückzuführen.

Die Antworten bezüglich der Möglichkeit der Fehlerkorrektur während der Erfüllung der Aufgaben unterscheiden sich erheblich (siehe Abbildung 6.2(k)). Der Grund dafür war eine unterschiedliche Fehlerquote der einzelnen Teilnehmer. Wenn ein Fehler gemacht wurde, der mithilfe der im Prototyp verfügbaren Aktionen nicht korrigiert werden konnte, musste die komplette Aufgabe von Anfang an wiederholt werden. Die fehlende Unterstützung von Aktionen wird im Folgenden diskutiert. Trotz der Unvollständigkeit des Prototyps waren die Teilnehmer vorwiegend der Meinung, dass die Layout-Unterstützung ein Zeitersparnis im Vergleich zu anderen Werkzeugen bringt (siehe Abbildung 6.2(l)). Dadurch wurde der geschaffene Mehrwert des Ansatzes bestätigt.

Wie bereits in Kapitel 5 erwähnt wurde, ist der umgesetzte Prototyp nicht ausgereift und es gibt eine Reihe an Funktionen, die nicht unterstützt werden. Die fehlenden Funktionen bilden den Gegenstand für einen weiteren Teil der Nutzerstudie. Einerseits wurden die Versuche der Ausführung von nicht implementierten Funktionen durch die Teilnehmer beobachtet. Anderseits wurden die Teilnehmer explizit nach den vermissten Funktionen gefragt. Die Ergebnisse sind in Abbildung 6.3 dargestellt.

Als eine der meist vermissten Funktionen gilt die Möglichkeit des Hinzufügens einer neuen Klasse mit einem Klick bzw. Doppelklick auf das Icon in der Sidebar. Die Einschränkung auf das „Drag and Drop“ wurde als ungenügend eingeschätzt, insbesondere wenn mehrere Klassen auf einmal hinzugefügt werden sollten. Die Behebung dieser Schwachstelle besteht darin, beide Techniken zu unterstützen und bei dem Klick, die neu hinzugefügte Klasse an eine vorgegebene Position im Diagramm zu platzieren.

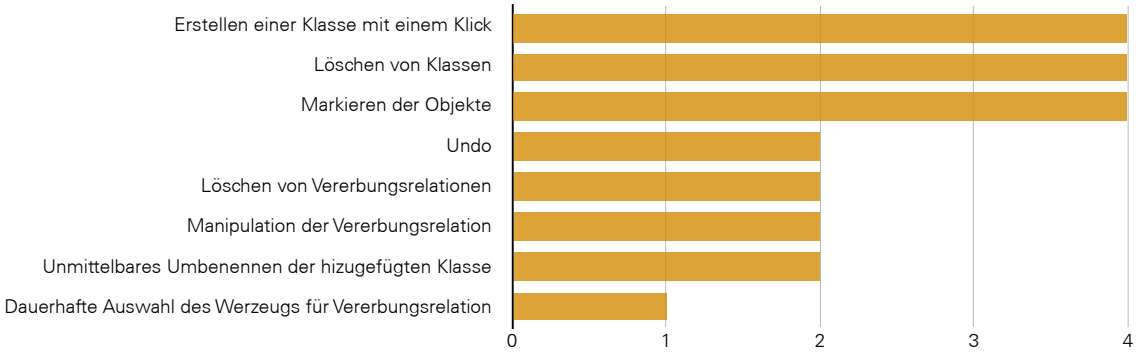


Abbildung 6.3.: Übersicht der vermissten Funktionen im Prototyp

Um die Korrektur von Fehlern zu ermöglichen, müssten Funktionen wie Löschen von Objekten, Manipulation der Vererbungsrelationen und „Undo“ zur Verfügung stehen. Für die objektbezogenen Funktionen ist weiterhin die Funktion deren Markierung im Canvas notwendig, die durch einige Teilnehmer für eine gleichzeitige Verschiebung von mehreren Klassen eingesetzt wurde. Die genannten Funktionen bilden eine Gruppe von Standardfunktionen, die in der Regel in einer kompletten Anwendung unterstützt werden sollten und durch die Teilnehmer der Nutzerstudie vermisst wurden. Deren Implementierung im Prototyp für Zwecke dieser Arbeit wurde aus zeitlichen Gründen weggelassen. Dennoch werden alle genannten Funktionen im Konzept berücksichtigt, denn sie können mithilfe von Layout-Ereignissen beschrieben werden (siehe Abschnitte 4.5 und 5.3.1.5).

Die Abbildung 6.3 zeigt weiterhin zwei kleinere Probleme der Benutzeroberfläche im Prototyp. In zwei Fällen fänden die Teilnehmer intuitiver, wenn neu hinzugefügten Klassen den Fokus behalten würden, um den Klassennamen auch ohne einen zusätzlichen Doppelklick eingeben zu können. Des Weiteren wurde in einem Fall angemerkt, dass das Werkzeug für die Vererbungsrelation dauerhaft ausgewählt werden könnte. Aufgrund der Kompliziertheit mit einer ständigen Auswahl des Icons in der Sidebar hat die Mehrheit der Teilnehmer zu der Alternative mit der Taste `CTRL` (siehe Abschnitt 5.2.1) gewechselt. Beide Probleme könnten durch einfache Anpassungen im Prototyp behoben werden und haben keinen Einfluss auf das gesamte Konzept. Die Erstellung von Relationen könnte zusätzlich noch mit an die Klassen gebundenen Buttons wie in *Visual Paradigm* durchgeführt werden [Vis14]. Insbesondere würde dies ein angemerktes Problem mit der nicht intuitiven Positionierung von neu hinzugefügten Klassen lösen.

Der Ablauf der Modellierung hat sich je nach dem Teilnehmer und der konkreten Aufgabe unterschieden. Im Wesentlichen wurden die Klassen nacheinander hinzugefügt und jede Klasse direkt umbenannt und in eine Vererbungshierarchie eingeordnet. In einigen Fällen wurde versucht, die Klassen frei zu positionieren. Die Funktionsweise wurde den Teilnehmern vor allem anhand des Platzhalter-Objektes verständlich gemacht und sie haben sich an die Einschränkung der freien Positionierung und das halbautomatische Layout gewöhnt. Ferner wurde die Zentrierung des Inhalts bei der Vergrößerung des Fensters als natürlich empfunden.

Die Nutzerstudie hat bestätigt, dass der entwickelte Ansatz den Prozess der Layout-Erstellung deutlich vereinfacht. Obwohl der umgesetzte Prototyp sehr eingeschränkt war und nur die Mo-

dellierung von einfachen Aufgaben ermöglicht hat, haben sich die Möglichkeiten der Interaktion dank der eingesetzten Bedienungskonzepten als intuitiv aufgewiesen. Weiterhin wurden durch die Nutzerstudie Stellen entdeckt, die verbessert bzw. erweitert werden könnten. Die gewonnenen Informationen sind für eine weitere Entwicklung des Prototyps und vor allem des zu Grunde liegenden Ansatzes sehr hilfreich.

6.2. ERFÜLLUNG DER KRITERIEN

Im Folgenden werden die Kriterien aus Abschnitt 4.1 einzeln aufgegriffen und deren Erfüllung in dem entwickelten Ansatz kritisch anhand von Fakten und Ergebnissen der Nutzerstudie bewertet:

K.1 **GUI** Die Bearbeitung des Diagramms basiert auf Mechanismen, die für klassische grafische Benutzeroberflächen ausgelegt sind. Insbesondere ist der Einsatz des Mauszeigers von Bedeutung, wodurch eine sequenzielle Ausführung von Bearbeitungsaktionen ermöglicht wird. Somit hält sich der Ansatz an die Rahmenbedingungen aus Abschnitt 1.3 und erfüllt das Kriterium K.1.

K.2 **Interaktivität** Das modellierte Diagramm wird in einem Canvas dargestellt, worin es unmittelbar bearbeitet werden kann. Die Grundlage dafür bilden die Bearbeitungsaktionen (siehe Abschnitt 4.3.1), die durch den Nutzer ausgeführt werden. Während bzw. nach der Ausführung einer Bearbeitungsaktion reagiert das System, indem das Layout des Diagramms mithilfe von Layout-Übergängen (siehe Abschnitt 4.3.3) angepasst wird. Dadurch dass der Prozess der Diagramm-Bearbeitung durch eine Sequenz von Bearbeitungsaktionen gebildet wird, ist die Interaktivität in dem Ansatz fest verankert. Das Kriterium K.2 wird daher ebenfalls erfüllt.

K.3 **Unmittelbares Feedback** Wie in Abschnitt 4.3.2 erläutert wurde, macht sich die Verschiebungsaktion den neu eingeführten Mechanismus der temporären Schicht zunutze. Während der Verschiebung wird das Layout des Diagramms kontinuierlich angepasst, wobei die Änderung anhand eines Platzhalters für den verschobenen Knoten und einer Animation des Layout-Übergangs visualisiert wird. Dem Nutzer wird im Verlauf der Aktion ständig das Ergebnis präsentiert, wodurch eine unmittelbare Feedback-Schleife erreicht wird (Kriterium K.3). Da alle anderen unterstützten Bearbeitungsaktionen diskret sind, setzen sie das unmittelbare Feedback nicht ein. Dagegen wird erst nach ihrer Ausführung der Layout-Übergang initiiert. Die Funktionsweise von beiden Arten der Bearbeitungsaktionen hat sich in der Nutzerstudie als verständlich erwiesen (siehe Abschnitt 6.1.2).

K.4 **Förderung des Prozesses der Diagramm-Erstellung** Die inkrementelle Bearbeitung der Diagramme bildet eine Grundlage des präsentierten Ansatzes. Sie erfolgt durch die Ausführung von Bearbeitungsaktionen, die entweder den Inhalt oder das Layout des Diagramms modifizieren (siehe Abschnitt 4.3.1). Die Änderung des Inhalts bewirkt eine automatische Anpassung des Layouts, welches wiederum durch den Nutzer mithilfe der Verschiebungsaktion in einem beschränkten Maß nach seinen Präferenzen verän-

dert werden kann. Durch die fehlende Möglichkeit der expliziten Anpassung von Layout-Eigenschaften[60] kann sich der Nutzer in der Bearbeitung des Diagramms gehindert fühlen. Dies wurde in der Nutzerstudie untersucht und für das gewählte Beispiel hat dies kein Problem dargestellt, wodurch das Kriterium K.4 erfüllt wird. Eine derartige Eignung für allgemeinere Fälle wurde in Abschnitt 6.1.2 diskutiert.

K.5 **Erhaltung des mentalen Modells** Wie in Kapitel 4 erläutert wurde, werden während der Bearbeitung des Diagramms Layout-Änderungen berechnet und auf das Diagramm angewendet. Dabei wird es angestrebt, sie minimal zu halten. Des Weiteren werden die Layout-Änderungen in sequenzielle Layout-Übergänge gekapselt (siehe Abschnitt 4.3.3), wodurch eine getrennte Wahrnehmung durch den Nutzer erreicht wird. Zudem werden die Layout-Übergänge animiert, sodass die neuen Positionen der Objekte ermittelt werden können. Diese Maßnahmen sorgen für die Erhaltung des mentalen Modells (Kriterium K.5).

K.6 **Förderung der Konzentration auf den Inhalt** Durch die automatische Layout-Berechnung und die Einschränkung der Möglichkeiten an der manuellen Layout-Anpassung wird der Aufwand an der Erzeugung des Layouts verringert, was durch die Nutzerstudie bestätigt wurde (siehe Abschnitt 6.1.2). Dies führt dazu, dass sich der Nutzer in erster Linie mit der Erstellung des Inhalts auseinandersetzt und das berechnete Layout nur ein wenig variieren kann. Damit wird das Kriterium K.6 für das gewählte Beispiel erfüllt. Allerdings ist zu erwähnen, dass eine Verallgemeinerung des Konzeptes für eine vollständige visuelle Sprache (z.B. für Klassendiagramme) die Notwendigkeit der Unterstützung von weiteren Möglichkeiten der manuellen Layout-Anpassungen mit sich bringen kann.

K.7 **Berücksichtigung der ästhetischen Prinzipien** Die ästhetischen Prinzipien aus Abschnitt 2.2 werden in dem entwickelten Ansatz durch den Algorithmus der Layout-Engine (siehe Abschnitt 4.5) und den Einsatz der Layout-Patterns (siehe Abschnitt 4.4) umgesetzt. Dadurch dass die unterstützte Notation der Klassendiagramme eingeschränkt wurde, konnte die Berücksichtigung der ästhetischen Prinzipien in dem Algorithmus für das baumbasierte Layout 4.5.3.2 deutlich vereinfacht werden. Im Folgenden wird gezeigt, inwiefern das Kriterium K.7 erfüllt wurde, indem auf die einzelnen ästhetischen Prinzipien eingegangen wird.

Durch die baumbasierte Struktur und die Umsetzung des impliziten Layout-Patterns für die Verhinderung der Knoten-Überlappung (siehe Abschnitt 4.4.1.2) wird sichergestellt, dass sich die Klassen nicht überlappen und entsprechend voneinander entfernt sind (Prinzip ÄP.1). Die Verbindung der Klassen erfolgt ausschließlich durch die Erstellung von azyklischen Vererbungsrelationen, wodurch die Kantenkreuzungen komplett ausgeschlossen werden (Prinzip ÄP.2). Obwohl der Ansatz für die Unterstützung der „Gabelform“ für Vererbungsrelationen ausgelegt ist, wurde diese aus zeitlichen Gründen im Prototyp nicht umgesetzt. Dahingegen wurde eine direkte Verbindung gewählt (Prinzip ÄP.9). Da die Vererbung der einzige unterstützte Relationstyp ist und die Relationen durch direkte Verbindungen dargestellt werden, sind die Prinzipien ÄP.3 und ÄP.4 für das Beispiel über-

[60] Insbesondere handelt es sich an dieser Stelle um die Einschränkung der freien Positionierung.

flüssig und wurden nicht betrachtet. Die hierarchischen Kanten der Vererbung werden aufgrund der Baumstruktur des Diagramm in einem vertikalen Verlauf dargestellt (Prinzip ÄP.7). Somit können Vererbungshierarchien mit mehreren Ebenen gebildet werden, die sich nach dem Prinzip der Darstellung von Hierarchien richten (Prinzip ÄP.8) und für die Positionierung der Klassen Symmetrie einsetzen (Prinzip ÄP.6).

Die einzelnen Vererbungshierarchien werden kompakt in einer horizontalen Linie im Diagramm angeordnet (Prinzip ÄP.10). Auf diese Weise werden ebenso die nicht verbundenen Klassen dargestellt. Durch den Nutzer kann ihre Reihenfolge geändert werden, sodass sie auch am Rand des Diagramms positioniert werden können (Prinzip ÄP.11). Um den Canvas möglichst optimal auszunutzen und das Prinzip ÄP.5 einzuhalten, wird ein großer Wert auf die Verteilung der Objekte gelegt, u.a. durch die automatische Zentrierung des Diagramms (siehe Abschnitt 4.4.1.3). Die horizontale Anordnung in dem Algorithmus für das baumbasierte Layout hat allerdings mit dem Problem der Skalierbarkeit zu kämpfen. Die Größe des Canvas im Prototyp ist durch die Größe des Fensters bestimmt und wird nicht erweitert, wenn der Inhalt des Diagramms nicht hineinpasst. Dies könnte durch den Einsatz eines „Scroll Views" gelöst werden. Durch mehrere vereinzelte Klassen oder Klassenhierarchien könnte trotzdem die Breite des Diagramms unangemessen zu der Höhe sein und dadurch das Prinzip ÄP.5 verletzen. In der Regel kommen in den Klassendiagrammen Klassen vor, die miteinander verbunden sind. Des Weiteren wird in [Amb02] empfohlen, Klassendiagramme klein zu halten, sodass sie nur wenige Klassen beinhalten. Dadurch könnte das genannte Problem umgegangen werden. Für eine objektive Aussage wäre allerdings eine weitere Untersuchung notwendig.

K.8 **Berücksichtigung der Syntax und Semantik** Der präsentierte Ansatz sieht es vor, die Layout-Berechnung anhand der Semantik der Objekte durchzuführen. Dadurch, dass sich das gewählte Beispiel auf stark vereinfachte Klassendiagramme beschränkt, wurde der Einfachheit halber kein Metamodell eingesetzt. Obwohl die Beschreibung der Syntax und Semantik in dem Ansatz nicht formalisiert ist, werden die Syntax- und Semantikregeln für Klassendiagramme in dem Algorithmus für das baumbasierte Layout eingehalten. Leider konnte der diagrammspezifische Aspekt des Ansatzes aus zeitlichen Gründen nicht tiefgründig ausgearbeitet werden und das Kriterium K.8 wurde nur teilweise erfüllt. Dahingegen wurde der Schwerpunkt der Arbeit auf die Interaktivität und die Gestaltung der Bedienungsmechanismen gelegt. An dieser Stelle bildet sich ein Potenzial für eine weitere Entwicklung, die in Abschnitt 7.2.2 thematisiert wird.

K.9 **Benutzerfreundlichkeit** Da die Benutzerfreundlichkeit ein subjektives Kriterium ist und faktisch nicht beurteilt werden kann, wurde sie im Rahmen der Nutzerstudie untersucht. Es hat sich herausgestellt, dass die Mechanismen grundsätzlich verständlich und leicht zu bedienen waren. Durch manche Teilnehmer wurde allerdings angemerkt, dass einige Funktionen des Prototyps (wie z.B. das Verhalten der Buttons in der Sidebar) intuitiver gestaltet werden könnten. Diese waren nicht eng mit dem Konzept verbunden und deren Verbesserung würde die Funktionsweise des umgesetzten Konzeptes nicht beeinflussen. Eine genauere Auswertung der Ergebnisse wurde in Abschnitt 6.1.2 aufgeführt.

6.3. ZUSAMMENFASSUNG

Die in diesem Kapitel behandelte Evaluation hat gezeigt, dass sich das entworfene Konzept und dessen Umsetzung an den aufgestellten Kriterien grundsätzlich halten und vor allem dass sie einen Mehrwert für die Modellierung von Diagrammen schaffen.

Den ersten Teil der Evaluation hat eine Nutzerstudie gebildet. Durch sieben Teilnehmer wurden einfache Diagramme in dem implementierten Prototyp modelliert und anschließend das Konzept bewertet. Es wurde bestätigt, dass sich die eingesetzten Bedienungsmechanismen als intuitiv aufweisen. Des Weiteren wurde gezeigt, dass die interaktive halbautomatische Layout-Berechnung einen positiven Einfluss auf den Bearbeitungsprozess hat, indem der Aufwand an der Layout-Erstellung reduziert wird. Obwohl die wichtigsten Funktionen für die Überprüfung des Konzeptes in der initialen Version des Prototyps umgesetzt wurden, haben die Teilnehmer weitere Funktionen vermisst. Insbesondere handelte es sich um Funktionen, die Korrekturen während der Modellierung ermöglichen würden. Diese Funktionen wurden im Rahmen der Nutzerstudie dokumentiert.

Es ist anzumerken, dass die Ergebnisse der Nutzerstudie durch die starke Vereinfachung des Prototyps beeinflusst werden könnten. Um eine objektive Aussage über die Eignung des Ansatzes zu bekommen, müsste das Konzept für eine vollständige visuelle Sprache erweitert werden, wodurch der Aspekt des diagrammspezifischen Layouts überprüft werden könnte. Des Weiteren würde die Implementierung von fehlenden Basisfunktionen in dem Prototyp zu einer genaueren Betrachtung des eigentlichen Ansatzes führen. Eventuell könnte eine quantitative Studie mit mehreren Nutzern durchgeführt werden.

In dem zweiten Teil der Evaluation wurde die Erfüllung der aufgestellten Kriterien bewertet. Da die Kriterien in den Entwurf des Konzeptes direkt eingeflossen sind, wurden sie im Großen und Ganzen erfüllt. Aus zeitlichen Gründen konnte die Berücksichtigung der Syntax- und Semantik allerdings nicht formalisiert werden und das Kriterium wurde dadurch nur teilweise erfüllt. Dennoch sieht dies das Konzept vor und kann nachträglich um die Formalisierung erweitert werden. Weiterhin ist zu erwähnen, dass die Unterstützung der ästhetischen Prinzipien die Vereinfachung der Klassendiagramme in dem gewählten Beispiel ausnutzt. Um einen verallgemeinerten Layout-Algorithmus für vollständige Klassendiagramme realisieren zu können, müssten unter Umständen weitere Maßnahmen eingeführt werden.

Schließlich konnten durch beide Teile der Evaluation Verbesserungs- und Erweiterungsmöglichkeiten entdeckt werden. Diese werden in Abschnitt 7.2 diskutiert.

7. ZUSAMMENFASSUNG

7.1. FAZIT

Diese Arbeit setzt sich mit der Problematik der **Layout-Berechnung in Diagrammen** auseinander und betrachtet diese aus der Sicht der Interaktion und der Art der Layout-Unterstützung. Zu Beginn wurden bestehende Ansätze für das Layout von Diagrammen ausführlich untersucht und bezüglich der genannten Aspekte kategorisiert. Die durchgeführte Analyse und die Aufstellung der Anforderung zur Eignung für **agile Modellierung** haben eine Grundlage für einen neuartigen Ansatz geschaffen, dessen Entwicklung den Schwerpunkt dieser Arbeit bildet.

Es handelt sich um einen Ansatz für Editoren zur Modellierung von graphbasierten Diagrammen, der die **interaktive Bearbeitung** des Diagramms mit der **automatischen Layout-Berechnung** kombiniert. Dies konnte dadurch erreicht werden, dass die Möglichkeit der expliziten Einstellung von Layout-Eigenschaften durch den Nutzer eingeschränkt wurde, ohne auf die Interaktivität komplett verzichten zu müssen. Dafür war allerdings der Einsatz von speziellen Bedienungsmechanismen notwendig, die das besondere Merkmal des präsentierten Ansatzes bilden.

Das in dieser Arbeit entwickelte Konzept beschreibt die allgemeine Funktionsweise des Ansatzes und dessen Bestandteile, u.a. die Interaktionsschleife, die Bedienungsmechanismen und die eigentliche Layout-Berechnung. Allerdings ist dieser Ansatz nicht universell einsetzbar und erfordert eine Anpassung für konkrete Diagrammtypen. Im Rahmen dieser Arbeit wurden zwei Layout-Algorithmen für **vereinfachte Klassendiagramme** entworfen, an den der Ansatz getestet wurde. Die Vertiefung bzw. der Entwurf von neuen Layout-Algorithmen stellen Gegenstände für weitere Forschung dar und werden in Abschnitt 7.2 diskutiert.

Schließlich wurde das entworfene Konzept in Form eines Werkzeugs für die Erstellung von Klassendiagrammen prototypisch umgesetzt. Des Weiteren wurde der Prototyp in einer Nutzerstudie an realen Nutzern evaluiert. Diese hat im Großen und Ganzen positive Ergebnisse geliefert. Insbesondere wurde gezeigt, dass die halbautomatische Layout-Berechnung einen Mehrwert schafft und dass die eingesetzten Mechanismen der Interaktion verständlich sind.

Dadurch stellt sich heraus, dass der Ansatz ein Potenzial besitzt und prinzipiell in Werkzeugen für agile Modellierung eingesetzt werden könnte. Dafür wäre allerdings eine eventuelle Weiterentwicklung und eine Anpassung für konkrete Anwendungsfälle notwendig.

7.2. AUSBLICK

In dieser Arbeit wurde ein initiales Konzept vorgestellt, welches in vielen Richtungen verbessert und erweitert werden kann. Insbesondere durch die Evaluation wurden Stellen entdeckt, die in der weiteren Entwicklung aufgearbeitet werden können. Sie werden in diesem Abschnitt im Detail dokumentiert.

7.2.1. IMPLEMENTIERUNG VON ERWEITERTEN FUNKTIONEN IM PROTOTYP

In dem Prototyp wurden ausschließlich die Funktionen implementiert, die das Validieren des Konzeptes ermöglicht haben. Das Ziel war es nicht, ein vollständiges Modellierungswerkzeug zu entwickeln. Natürlich bedeutet dies, dass der Prototyp viele Funktionen vermissen lässt, die aus anderen Programmen bekannt sind. Wie bereits in Abschnitt 6.1.2 aufgeführt wurde, hat die Nutzerstudie gezeigt, welche Funktionen den Teilnehmern am meisten gefehlt haben. Im Einzelnen handelte es sich um weitere Möglichkeiten der Manipulation von Elementen im Diagramm und um Funktionen zur Verbesserung der Benutzerfreundlichkeit. Eine Übersicht der Ergebnisse wurde in Abbildung 6.3 illustriert. Des Weiteren wäre die Möglichkeit des Speicherns von Dokumenten sehr nützlich. Obwohl die genannten Funktionen nicht direkt mit dem eigentlichen Konzept verbunden sind, wäre deren Implementierung in dem Prototyp wünschenswert.

7.2.2. VERBESSERUNG DER UNTERSTÜTZUNG DER SYNTAX UND SEMANTIK

Durch das Aufstellen des Kriteriums K.8 wurde gefordert, dass der entworfene Ansatz die Syntax und Semantik der Diagramme berücksichtigt. Diese Anforderung ist insbesondere wichtig, um die Anpassung der Layout-Berechnung für einzelne Diagrammtypen zu ermöglichen. Obwohl dieses Kriterium in den präsentierten konkreten Layout-Algorithmen eingehalten wurde, ist für die Weiterentwicklung der Algorithmen bzw. den Entwurf von neuen Algorithmen für weitere Diagrammtypen notwendig, eine Formalisierung der Syntax und Semantik einzuführen. Dafür müsste ein Metamodell für die Beschreibung der abstrakten und konkreten Syntax der visuellen Sprachen eingesetzt werden. Des Weiteren könnten die Layout-Patterns ähnlich wie in [Mai12] anhand von Pattern-spezifischen Metamodellen definiert werden, um deren Wiederverwendung in Layout-Algorithmen für unterschiedliche Diagrammtypen zu gewährleisten.

7.2.3. VERALLGEMEINERUNG DES LAYOUT-ALGORITHMUS FÜR KLASSENDIAGRAMME

Der in Abschnitt 4.5.3.2 vorgestellte baumbasierte Layout-Algorithmus ist für vereinfachte Klassendiagramme ausgelegt und unterstützt ausschließlich die Modellierung von Vererbungshierarchien. Die Beschränkung der Struktur auf Bäume konnte für die Implementierung des Algorithmus ausgenutzt werden. Damit ist eine Erweiterung für vollständige Klassendiagramme mit weiteren Maßnahmen verbunden. Insbesondere müsste die Zusammensetzung der Layout-Patterns überarbeitet werden, möglicherweise durch den Einsatz eines Constraintlösers. Dafür wäre es notwendig, sowohl die expliziten als auch die impliziten Layout-Patterns in Form von Constraints ausdrücken zu können. Wie zu erwarten müsste in diesem Fall auf den Aspekt der Performance geachtet werden.

Da es sich bei den vollständigen Klassendiagrammen aufgrund der Erweiterung der unterstützten Notation um komplexere graphbasierte Strukturen handelt, wäre eventuell eine Anpassung der Behandlung von Layout-Patterns notwendig. Es wäre denkbar, die Layout-Patterns während der Bearbeitung zu visualisieren[61]. Des Weiteren könnte eine direkte Interaktion mit den Layout-Patterns unterstützt werden, z.B. durch die Verschiebung einer Klasse auf die visuelle Repräsentation des Layout-Patterns für das Hinzufügen der Klasse zu dem Layout-Pattern. Obwohl der in dieser Arbeit vorgestellte Ansatz eine vollautomatische Verwaltung von Layout-Patterns durch die Layout-Algorithmen vorsieht, könnte aufgrund der komplexeren Struktur der Diagramme sogar die Möglichkeit der Erstellung von Layout-Patterns durch den Nutzer[62] ähnlich wie in [Mai12] benötigt werden. Weiterhin führt die Unterstützung von mehreren Relationstypen zu der Notwendigkeit der Einführung von Layout-Patterns für Kanten, wodurch deren Routen bestimmt würden.

Außerdem ist es denkbar, den Ansatz um weitere Eingabequellen für die Layout-Algorithmen zu erweitern. Dazu zählen Eigenschaften der Mausbewegung wie z.B. die Richtung oder Geschwindigkeit sowie eine Verwaltung der Historie von angewendeten Layouts.

[61] Um das Diagramm übersichtlich zu halten, könnten die Layout-Patterns nur während der Verschiebungsaktion eingeblendet werden.

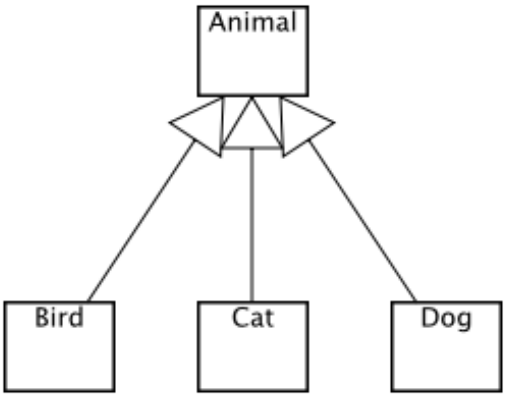
[62] Je nach Semantik könnte dies mit der automatischen Erstellung von Layout-Patterns kombiniert werden.

A. MATERIAL ZUR NUTZERSTUDIE

A.1. AUFGABENSTELLUNG

Aufgabe 1

1. Modellieren Sie folgendes Diagramm:

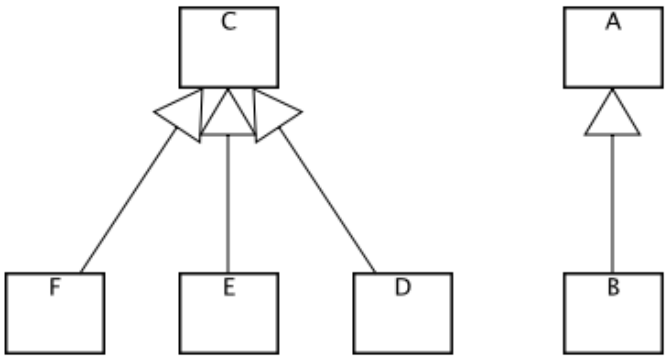


Aufgabe 2

1. Erstellen Sie eine Klasse `Bicycle`.
2. Fügen Sie der Klasse `Bicycle` eine neue Oberklasse `Vehicle` hinzu.
3. Erstellen Sie eine neue Klasse `Driver`.
4. Fügen Sie der Klasse `Vehicle` eine neue Unterklasse `Car` hinzu.
5. Fügen Sie der Klasse `Vehicle` eine neue Unterklasse `Train` hinzu.
6. Nennen Sie die Klasse `Bicycle` zu `Bike` um.
7. Fügen Sie der Klasse `Car` eine neue Unterklasse `Van` hinzu.

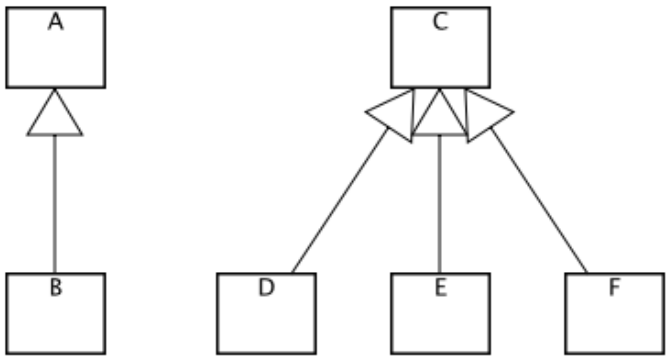
Aufgabe 3A

1. Modellieren Sie folgendes Diagramm:



Aufgabe 3B

2. Verändern Sie nun das modellierte Diagramm, so dass es wie folgt aussieht:



Aufgabe 4

In dieser Aufgabe soll eine Vererbungshierarchie für geometrische Objekte erstellt werden. Die Oberklasse der Hierarchie soll **`Shape`** genannt werden. Die Objekte werden in zwei- und dreidimensionale Objekte aufgeteilt, die durch die Klassen **`Shape2D`** und **`Shape3D`** repräsentiert werden. Beide Klassen sind Unterklassen der Klasse **`Shape`**.

Zu den zweidimensionalen Objekten gehören Ellipse (Klasse **`Ellipse`**) und Polygon (Klasse **`Polygon`**). Eine spezielle Form der Ellipse, der Kreis, soll durch die Klasse **`Circle`** repräsentiert werden. Zu den Polygonen gehören Rechteck (Klasse **`Rectangle`**) und Dreieck (Klasse **`Triangle`**). Eine spezielle Form des Rechtecks, das Quadrat, soll durch die Klasse **`Square`** repräsentiert werden.

Zu den dreidimensionalen Objekten gehören Kugel (Klasse **`Sphere`**), Pyramide (Klasse **`Pyramid`**), Quader (Klasse **`Cuboid`**), Zylinder (Klasse **`Cylinder`**) und Kegel (Klasse **`Cone`**). Eine spezielle Form des Quaders, der Würfel, soll mit der Klasse **`Cube`** repräsentiert werden.

Sortieren Sie anschließend die erstellten Klassen im Diagramm alphabetisch.

A.2. FRAGEBOGEN

Nutzerstudie zur Bachelor-Arbeit „Entwicklung eines Konzeptes für das interaktive und diagrammspezifische Layout von graphbasierten Softwarediagrammen“

1. Geben Sie bitte Ihr Alter an.

☐☐

2. Geben Sie bitte Ihr Geschlecht an.

- ☐ Männlich
- ☐ Weiblich

3. Geben Sie bitte Ihren derzeitigen Berufsstatus an.

- ☐ Berufstätig, Beruf: _____
- ☐ Student, Studiengang: _____
- ☐ Sonstiges

4. Haben Sie Erfahrung mit dem Betriebssystem Mac OS X?

- ☐ Ja, ich nutze es täglich
- ☐ Ja, ich habe es bereits benutzt
- ☐ Nein

5. Verfügen Sie über Kenntnisse der objektorientierten Programmierung? Verstehen Sie Begriffe wie Klasse, Vererbung, Unterklasse und Oberklasse?

- ☐ Ja
- ☐ Nein

6. Haben Sie Erfahrung mit Klassendiagrammen der Notationssprache UML? Kennen Sie die Notation für Klassen und Vererbung in UML?

- ☐ Ja
- ☐ Nein

7. Welche Tools verwenden Sie zum Erstellen von (Software-)Diagrammen und in welchem Maß?

	sehr oft	oft	manchmal	selten	gar nicht
Diagramm-Programme (Microsoft Visio, OmniGraffle, ConceptDraw)	☐	☐	☐	☐	☐
Präsentationsprogramme (Apple Keynote, Microsoft PowerPoint)	☐	☐	☐	☐	☐
Online-Tools (draw.io, yUML.me, Lucidchart)	☐	☐	☐	☐	☐
UML-Editoren (Visual Paradigm, MagicDraw, ArgoUML)	☐	☐	☐	☐	☐
Andere Programme:					
_____	☐	☐	☐	☐	☐
Stift und Papier	☐	☐	☐	☐	☐
Whiteboard	☐	☐	☐	☐	☐
Sonstiges:					
_____	☐	☐	☐	☐	☐

8. Bitte bewerten Sie folgende Aussagen:

	trifft völlig zu	trifft zu	teils/teils	trifft nicht zu	trifft gar nicht zu
Ich habe meiner Meinung nach alle Aufgaben erfolgreich erfüllt.	☐	☐	☐	☐	☐
Die Erfüllung der Aufgaben war anstrengend.	☐	☐	☐	☐	☐
Ich denke, dass ich die Aufgaben auch ohne vorherige Vorstellung der möglichen Aktionen des Prototyps problemlos lösen könnte.	☐	☐	☐	☐	☐
Ich hatte das Programm die ganze Zeit unter Kontrolle.	☐	☐	☐	☐	☐
Die auf „Drag and Drop“ basierende Verschiebungsaktion war verständlich.	☐	☐	☐	☐	☐
Durch die Einschränkung der freien Positionierung wurde der Aufwand an der Erstellung des Layouts reduziert.	☐	☐	☐	☐	☐
Die Einschränkung der freien Positionierung fand ich irritierend.	☐	☐	☐	☐	☐
Für mich war jederzeit erkennbar, an welche Stelle ich die angeklickte Klasse verschieben kann.	☐	☐	☐	☐	☐
Die automatischen Layout-Updates haben mich in der Erstellung des Diagramms gehindert.	☐	☐	☐	☐	☐
Das Layout der modellierten Diagramme fand ich ästhetisch ansprechend.	☐	☐	☐	☐	☐
Ich konnte Fehler, die während der Erfüllung der Aufgaben gemacht wurden, korrigieren.	☐	☐	☐	☐	☐
Ich bin der Meinung, dass ich die Aufgaben in einer Software ohne Layout-Unterstützung schneller erledigen könnte.	☐	☐	☐	☐	☐

9. Welche nicht unterstützte Aktion haben Sie in dem Prototyp vermisst?

_____

_____

10. Geben Sie das größte Problem an, das Sie während der Erfüllung der Aufgaben beobachten konnten:

_____

_____

_____

11. Haben Sie weitere Anmerkungen/Kritikpunkte/Verbesserungsvorschläge bzgl. der Bedienung, der Mechanismen der Interaktion, der prototypischen Implementierung oder der Durchführung der Nutzerstudie?

_____

_____

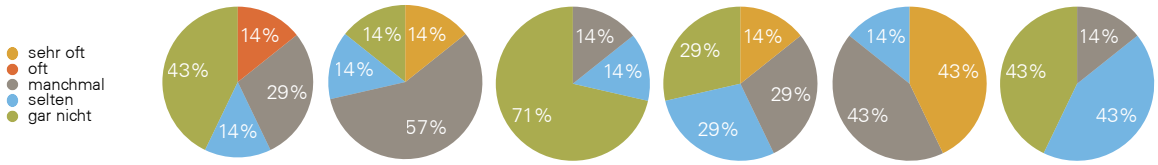
_____

_____

A.3. AUSWERTUNG

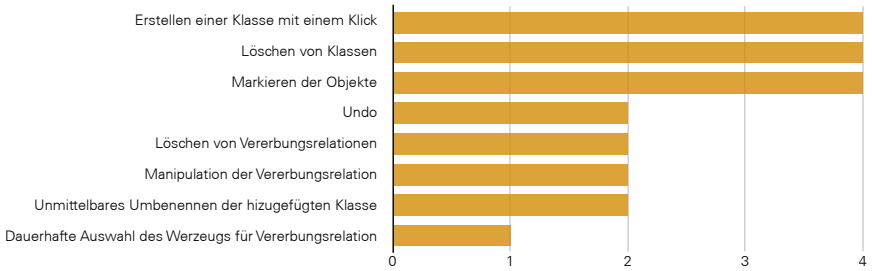
Auswertung der Frage 7: Verwendete Tools

	Diagramm-Programme	Präsentationsprogramme	Online-Tools	UML-Editoren	Stift und Papier	Whiteboard
1	gar nicht	manchmal	gar nicht	gar nicht	manchmal	selten
2	selten	manchmal	gar nicht	selten	selten	selten
3	gar nicht	manchmal	gar nicht	manchmal	manchmal	gar nicht
4	manchmal	gar nicht	gar nicht	sehr oft	sehr oft	gar nicht
5	oft	sehr oft	selten	manchmal	sehr oft	manchmal
6	gar nicht	selten	gar nicht	gar nicht	manchmal	selten
7	manchmal	manchmal	manchmal	selten	sehr oft	gar nicht
sehr oft	**0**	**1**	**0**	**1**	**3**	**0**
oft	**1**	**0**	**0**	**0**	**0**	**0**
manchmal	**2**	**4**	**1**	**2**	**3**	**1**
selten	**1**	**1**	**1**	**2**	**1**	**3**
gar nicht	**3**	**1**	**5**	**2**	**0**	**3**



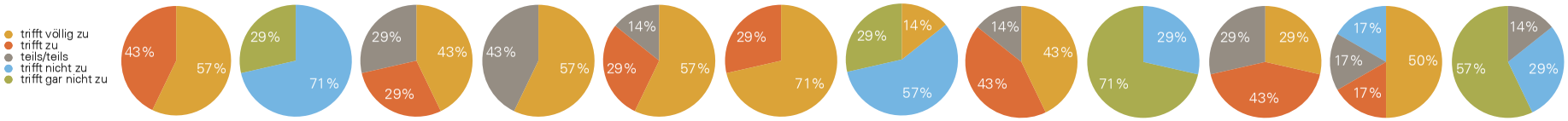
Auswertung der Frage 9: Vermisste Aktionen im Prototyp

	Erstellen einer Klasse mit einem Klick	Löschen von Klassen	Markieren der Objekte	Undo	Löschen von Vererbungsrelationen	Manipulation der Vererbungsrelation	Unmittelbares Umbenennen der hizugefügten Klasse	Dauerhafte Auswahl des Werzeugs für Vererbungsrelation
1	☑	☐	☑	☑	☐	☐	☐	☐
2	☐	☑	☑	☐	☑	☐	☐	☐
3	☑	☑	☐	☐	☐	☐	☐	☐
4	☐	☐	☑	☑	☐	☐	☑	☐
5	☐	☐	☐	☐	☐	☑	☐	☑
6	☑	☑	☑	☐	☑	☑	☑	☐
7	☑	☑	☐	☐	☐	☐	☐	☐
	4	**4**	**4**	**2**	**2**	**2**	**2**	**1**



Auswertung der Frage 8: Evaluation

	Ich habe meiner Meinung nach alle Aufgaben erfolgreich erfüllt.	Die Erfüllung der Aufgaben war anstrengend.	Ich denke, dass ich die Aufgaben auch ohne vorherige Vorstellung der möglichen Aktionen des Prototyps problemlos lösen könnte.	Ich hatte das Programm die ganze Zeit unter Kontrolle.	Die auf "Drag and Drop" basierende Verschiebungsaktion war verständlich.	Durch die Einschränkung der freien Positionierung wurde der Aufwand an der Erstellung des Layouts reduziert.	Die Einschränkung der freien Positionierung fand ich irritierend.	Für mich war jederzeit erkennbar, an welche Stelle ich die angeklickte Klasse verschieben kann.	Die automatischen Layout-Updates haben mich in der Erstellung des Diagramms gehindert.	Das Layout der modellierten Diagramme fand ich ästhetisch ansprechend.	Ich konnte Fehler, die während der Erfüllung der Aufgaben gemacht wurden, korrigieren.	Ich bin der Meinung, dass ich die Aufgaben in einer Software ohne Layout-Unterstützung schneller erledigen könnte.
1	trifft zu	trifft nicht zu	trifft zu	teils/teils	teils/teils	trifft völlig zu	trifft nicht zu	trifft zu	trifft gar nicht zu	trifft völlig zu	trifft nicht zu	trifft gar nicht zu
2	trifft völlig zu	trifft nicht zu	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft nicht zu	trifft zu	trifft völlig zu	trifft gar nicht zu
3	trifft völlig zu	trifft nicht zu	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft zu	trifft nicht zu	trifft zu	trifft gar nicht zu	trifft zu	trifft völlig zu	trifft nicht zu
4	trifft völlig zu	trifft gar nicht zu	teils/teils	trifft völlig zu	trifft völlig zu	trifft völlig zu	trifft gar nicht zu	trifft völlig zu	trifft gar nicht zu	trifft völlig zu		trifft gar nicht zu
5	trifft zu	trifft nicht zu	trifft zu	teils/teils	trifft völlig zu	trifft zu	trifft nicht zu	teils/teils	trifft nicht zu	teils/teils	trifft völlig zu	teils/teils
6	trifft zu	trifft nicht zu	teils/teils	teils/teils	trifft zu	trifft völlig zu	trifft nicht zu	trifft zu	trifft gar nicht zu	teils/teils	teils/teils	trifft gar nicht zu
7	trifft völlig zu	trifft gar nicht zu	trifft völlig zu	trifft völlig zu	trifft zu	trifft völlig zu	trifft gar nicht zu	trifft völlig zu	trifft gar nicht zu	trifft zu	trifft zu	trifft nicht zu
trifft völlig zu	4	0	3	4	4	5	1	3	0	2	3	0
trifft zu	3	0	2	0	2	2	0	3	0	3	1	0
teils/teils	0	0	2	3	1	0	0	1	0	2	1	1
trifft nicht zu	0	5	0	0	0	0	4	0	2	0	1	2
trifft gar nicht zu	0	2	0	0	0	0	2	0	5	0	0	4



B. INHALT DER CD

Die auf der beigelegten CD enthaltenen Dateien werden im Folgenden mit ihren Pfaden und kurzen Beschreibungen aufgelistet.

B.1. SCHRIFTLICHE ARBEIT

`Bachelor-Thesis/Bachelor-Thesis.pdf`
Originalversion der Bachelor-Arbeit

`Bachelor-Thesis/Bachelor-Thesis-Print.pdf`
Druckversion der Bachelor-Arbeit

B.2. PROTOTYP

`Prototype/InteractiveDiagramLayout.app`
Kompiliertes Prototyp in Version 0.1.2

`Prototype/Builds/`
Alle verfügbaren kompilierten Versionen des Prototyps

`Prototype/Code/`
Xcode-Projekt und Quellcode des Prototyps in Version 0.1.2

`Prototype/Videos/`
Videos zur Vorführung des implementierten Prototyps

B.3. NUTZERSTUDIE

`User-Study/Exercises.pdf`
Aufgabenstellung für die Nutzerstudie

`User-Study/Questionnaire.pdf`
Fragebogen für die Nutzerstudie

`User-Study/Screen-Recordings/`
Bildschirmaufnahmen der einzelnen Sitzungen

`User-Study/Completed-Questionnaires/`
Eingescannte ausgefüllte Fragebögen

`User-Study/Evaluation.{pdf,numbers}`
Auswertung der Ergebnisse der Nutzerstudie

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

[Amb02] AMBLER, Scott W.: *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York, USA : John Wiley & Sons, Inc., 2002. – ISBN 978–0471202820

[Amb04] AMBLER, Scott W.: *UML 2 Class Diagrams: An Agile Introduction*. `http://www.agilemodeling.com/artifacts/classDiagram.htm`. Version: 2004, Abruf: 15.09.2014

[Amb05] AMBLER, Scott W.: *The Elements of UML 2.0 Style*. Cambridge, UK : Cambridge University Press, 2005. – ISBN 978–0521616782

[AN05] ARLOW, Jim ; NEUSTADT, Ila: *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. 2. Auflage. Pearson Education, 2005. – ISBN 978–0321321275

[App11] APPLE INC. (Hrsg.): *Keynote '09 Benutzerhandbuch*. Apple Inc., 2011. `http://manuals.info.apple.com/de_DE/Keynote09_Benutzerhandbuch.pdf`

[Arv02] ARVO, James: Techniques for Interactive Graph Drawing. In: KOBOUROV, Stephen G. (Hrsg.) ; GOODRICH, Michael T. (Hrsg.): *Graph Drawing* Bd. 2528. Springer, 2002. – ISBN 978–3540001584, S. 380

[BBB+01] BECK, Kent ; BEEDLE, Mike ; BENNEKUM, Arie van ; COCKBURN, Alistair ; CUNNINGHAM, Ward ; FOWLER, Martin ; GRENNING, James ; HIGHSMITH, Jim ; HUNT, Andrew ; JEFFRIES, Ron ; KERN, Jon ; MARICK, Brian ; MARTIN, Robert C. ; MELLOR, Steve ; SCHWABER, Ken ; SUTHERLAND, Jeff ; THOMAS, Dave: *Manifest für Agile Softwareentwicklung*. `http://agilemanifesto.org/iso/de/`. Version: 2001, Abruf: 29.07.2014

[Bra01] BRANKE, Jürgen: Dynamic Graph Drawing. In: KAUFMANN, Michael (Hrsg.) ; WAGNER, Dorothea (Hrsg.): *Drawing Graphs: Methods and Models* Bd. 2025. Springer, 2001. – ISBN 978–3540420620, S. 228–246

[DMW08] DWYER, Tim ; MARRIOTT, Kim ; WYBROW, Michael: Interactive, Constraint-based Layout of Engineering Diagrams. In: *ECEASST* 13 (2008)

[Eic05] EICHELBERGER, Holger: *Aesthetics and Automatic Layout of UML Class Diagrams*, Julius Maximilian University of Würzburg, Diss., 2005. http://opus.bibliothek.uni-wuerzburg.de/files/1260/diss_print.pdf

[Eig04] EIGLSPERGER, Markus: *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach*, Eberhard-Karls-Universität Tübingen, Diss., 2004

[ES09] EICHELBERGER, Holger ; SCHMID, Klaus: Guidelines on the Aesthetic Quality of UML Class Diagrams. In: *Information and Software Technology* 51 (2009), Nr. 12, S. 1686–1698

[FHD10] FRISCH, Mathias ; HEYDEKORN, Jens ; DACHSELT, Raimund: Diagram Editing on Interactive Displays Using Multi-touch and Pen Gestures. In: GOEL, Ashok K. (Hrsg.) ; JAMNIK, Mateja (Hrsg.) ; NARAYANAN, Hari N. (Hrsg.): *Proceedings of the 6th International Conference on Diagrammatic Representation and Inference* Bd. 6170, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3642145995, S. 182–196

[Fow03] FOWLER, Martin: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. Auflage. Addison-Wesley Professional, 2003. – ISBN 978–0321193681

[Fuh11] FUHRMANN, Hauke A. L.: *On the Pragmatics of Graphical Modeling*, Christian-Albrechts-Universität zu Kiel, Diss., 2011. http://books.google.de/books?id=eGANfl9jnaAC

[Gan14] GANSNER, Emden R. ; AT&T RESEARCH (Hrsg.): *Using Graphviz as a Library*. AT&T Research, 2014. http://www.graphviz.org/doc/libguide/libguide.pdf

[GSE+14] GLADISCH, Stefan ; SCHUMANN, Heidrun ; ERNST, Mathias ; FÜLLEN, Georg ; TOMINSKI, Christian: Semi-Automatic Editing of Graphs with Customized Layouts. In: *Computer Graphics Forum* 33 (2014), Nr. 3, S. 381–390

[Ide14] IDEASONCANVAS (Hrsg.): *MindNode Pro 1.10 User Guide*. IdeasOnCanvas, 2014. https://mindnode.com/support/MindNodeProUserGuide.pdf

[Mai12] MAIER, Sonja: *A Pattern-based Approach for the Combination of Different Layout Algorithms in Diagram Editors*, Universitätsbibliothek der Universität der Bundeswehr, Diss., 2012. http://athene.bibl.unibw-muenchen.de:8081/doc/90423/90423.pdf

[MM10] MAIER, Sonja ; MINAS, Mark: Combination of Different Layout Approaches. In: BOTTONI, Paolo (Hrsg.) ; GUERRA, Esther (Hrsg.) ; JUAN, Lara de (Hrsg.): *Proceedings of the 2nd International Workshop on Visual Formalisms for Patterns* Bd. 31, 2010 (Electronic Communications of the EASST)

[MM13] MAIER, Sonja ; MINAS, Mark: A Pattern-based Approach for Initial Diagram Layout. In: TICHY, Matthias (Hrsg.) ; RIBEIRO, Leila (Hrsg.): *Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques* Bd. 58, 2013 (Electronic Communications of the EASST)

[NGEH14] NORTH, Stephen C. ; GANSNER, Emden R. ; ELLSON, John C. ; HU, Yifan: *Dot Manual Page*, 2014

[Nie09] NIELSEN, Jakob: *Discount Usability: 20 Years*. `http://www.nngroup.com/articles/discount-usability-20-years`. Version: 2009, Abruf: 12.10.2014

[Ols10] OLSEN, Ruben: *OmniGraffle 5 Diagramming Essentials: Create Better Diagrams with Less Effort Using OmniGraffle*. Packt Publishing, 2010. – ISBN 978–1849690768

[Omn08] THE OMNI GROUP (Hrsg.): *OmniGraffle 5 Professional Manual*. The Omni Group, 2008. `http://downloads2.omnigroup.com/software/MacOSX/Manuals/OmniGraffle-5-Manual.pdf`

[Roq10] ROQUES, Arnaud: *Drawing UML with PlantUML*, 2010

[SGMMS03] SEYBOLD, Christian ; GLINZ, Martin ; MEIER, Silvio ; MERLO-SCHETT, Nancy: An Effective Layout Adaptation Technique for a Graphical Modeling Tool. In: *Proceedings of the 25th International Conference on Software Engineering*. Washington, USA : IEEE Computer Society, 2003 (ICSE '03), S. 826–827

[Sie03] SIEBENHALLER, Martin: *Automatisches Layout von UML-Klassendiagrammen*, University of Tübingen, Diplomarbeit, 2003. `http://www-pr.informatik.uni-tuebingen.de/mitarbeiter/martinsiebenhaller/downloads/diplomarbeit.pdf`

[SKM93] SHIEBER, Stuart M. ; KOSAK, Corey ; MARKS, Joe: Automating the Layout of Network Diagrams with Specified Visual Organization. In: *IEEE Transactions on Systems, Man, and Cybernetics* 24 (1993), S. 440–454

[Ver14] VERSIONONE (Hrsg.): *8th Annual State of Agile Development Survey*. VersionOne, 2014. `http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf`

[Vis14] VISUAL PARADIGM (Hrsg.): *Visual Paradigm User's Guide*. Visual Paradigm, 2014. `http://www.visual-paradigm.com/support/documents/`

[Wyb08] WYBROW, Michael: *Using Semi-automatic Layout to Improve the Usability of Diagramming Software*, Clayton School of Information Technology, Monash University, Diss., 2008. `http://www.csse.monash.edu.au/~mwybrow/papers/wybrow-thesis-2008.pdf`