Label Management in Graph Layout Algorithmen

Yella Lasch

Bachelorarbeit
2015

Prof. Dr. Reinhard von Hanxleden
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Betreut durch
Dipl.-Inf. Christoph Daniel Schulze

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Label sind Bezeichner von Diagrammelementen und tragen im Diagramm so zum Verständnis bei. Je nach visueller Sprache können diese dabei sehr groß ausfallen. Damit große Label trotzdem sinnvoll dargestellt werden, werden Knoten und Kanten entsprechend vergrößert. Dies führt zu einer allgemeinen Vergrößerung des Diagramms und macht es damit oft unübersichtlich. Um diesem Problem entgegenzuwirken, können Label gekürzt werden.

In dieser Arbeit werden Möglichkeiten zur Kürzung von Labeln vorgestellt, implementiert und evaluiert. Dazu gibt es Basisstrategien, wie das Abschneiden ab einer bestimmten Breite, die es ermöglichen, Label zu kürzen, ohne dass über diese etwas bekannt ist. Andere Kürzungsstrategien sind sehr problemspezifisch.

Weiterhin wird sich im *Label Management* damit befasst, welche Label gekürzt werden. So können aktuell für den Benutzer interessante Elemente ungekürzt bleiben, während die umliegenden gekürzt werden. Es wird gezeigt, dass ein Graph, der in horizontalen Ebenen organisiert ist, ohne Kantenlabel um bis zu 75% in der Breite verkleinert werden kann. Die vorgestellten Kürzungsstrategien können ähnlich gute Werte erreichen. Die platzeffektivsten Kürzungen von Labeln gehen allerdings immer mit Informationsverlusten einher.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Visualisierung ist ein großer Bestandteil der Informatik. Ob beim Taschenrechner oder in einer Entwicklungsumgebung, die Frage, wie die Informationen dem Benutzer verständlich gemacht werden und ob die Bedienung des Programms dadurch intuitiv ist, stellt sich überall.

Ein wichtiges Werkzeug der Visualisierung von Informationen sind *Graphen*. Graphen bestehen aus *Knoten* und *Kanten*. Ein Beispiel für einen einfachen Graphen ist ein U-Bahn-Plan, wobei die Haltestellen durch Knoten und die Verbindungen zwischen den Stationen durch Kanten repräsentiert werden.

Graphen werden auch zur Visualisierung großer Datenmengen verwendet. Ein Beispiel dafür sind soziale Netzwerke. Für solche können die Beziehungen zwischen Benutzern als Graph dargestellt werden. Dabei werden die Benutzer durch Knoten und die Beziehungen zwischen ihnen durch Kanten visualisiert.

Ein weiterer Anwendungsfall von Graphen ist die Modellierung. Nehmen wir an, wir wollen das Verhalten einer Lampe modellieren. Wir haben dabei zwei Zustände: „An“ und „Aus“. Jeder dieser Zustände kann bei der Visualisierung mit einem Knoten dargestellt werden. Die Zustandswechsel von An zu Aus werden dann durch Kanten repräsentiert. Das Modell ist in Abbildung 1.1 zu sehen.

Beide Beispiele wären aber nicht verständlich ohne *Label*. Label sind Zeichenketten, also zum Beispiel Namen. Sie können aber auch deutlich komplexer sein und einer bestimmten Form folgen. Label in einem U-Bahn-Plan sind die Namen der Stationen und die Namen der U-Bahn-Linie pro Kante. Ohne sie wäre der Graph als Plan nicht zu verwenden, da es nicht möglich wäre, sich darin zurechtzufinden. In dem kleinen Lampenmodell hingegen verdeutlichen die Label an den Kanten, unter welchen Bedingungen der Zustand gewechselt wird. Label sind also wichtig für das Verständnis von Graphen; sie transportieren Informationen.

Neben Knoten und Kanten können Label auch an *Ports* auftreten. Ports können in komplexeren Modellen vorkommen, in denen Kanten Signalleitungen und Knoten sogenannte *Aktoren* repräsentieren. Aktoren verarbeiten Signale und können welche ausgeben. Ports können an diesen Aktoren angebracht sein und repräsentieren dann die Ein- oder Ausgänge der Aktoren, ähnlich den Kontakten von Bauteilen in elektrischen Schaltplänen. Um zu verdeutlichen welches Signal an den Port angelegt wird oder welchen Zweck es erfüllt, wird zum Beispiel dessen Name an den Port geschrieben. Abbildung 1.2 zeigt ein

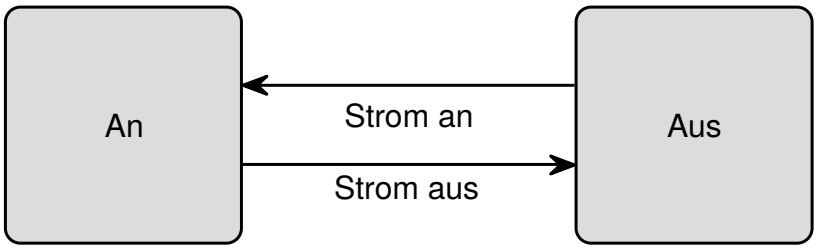

Abbildung 1.1. Modellierung einer Lampe mit Hilfe eines Graphen.

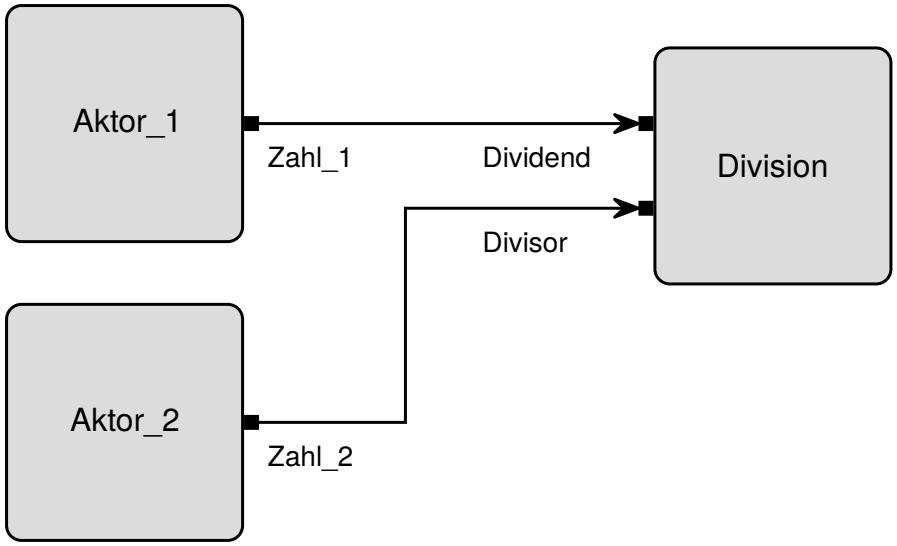

Abbildung 1.2. Graph mit Ports. Zwei Aktoren emittieren jeweils eine Zahl, während der Dritte diese dividiert. Die Ports verdeutlichen welche Zahl Dividend und welche Divisor ist.

Beispielmodell. Dort emittieren zwei Aktoren jeweils eine Zahl, während der Dritte diese dividiert. Die Portlabel verdeutlichen, was die Aktoren ausgeben, was für Signale erwartet werden und wozu sie weiterverwendet werden.

In dem Kontext, in dem wir die Graphen betrachten wollen, werden diese bereits durch automatisches Layout organisiert. Dabei werden die Knoten in *Layern* angeordnet, das heißt, dass sie in verschiedenen Schichten platziert werden, sodass die gerichteten Kanten möglichst nur in eine Richtung zeigen. Weiterhin wird das Platzieren der Label bereits übernommen. Dafür werden die Kanten, die mit Labeln versehen sind, so weit gestreckt, dass das Label keine Überschneidungen mit Knoten, Ports oder anderen Labeln hat.

Tabelle 1.1. Kürzungstrategien nach Fuhrmann [Fuh11] für das Label „(not SignalA) and (not SignalB) / SignalC(counter)“.

Type		Ergebnis
Wrapped	syntactical hard	`(not SignalA) and (n ot SignalB) / Signal C(counter)`
	syntactical soft	`(not SignalA) and ( not SignalB) / SignalC( counter)`
	semantical	`(not SignalA) and (not SignalB) / SignalC(counter)`
Abbreviated	syntactical	`(not SignalA) and (not Si...`
	semantical	`SignalA, SignalB / SignalC`

1.1. Problemstellung

In den mit Labeln versehenen Graphen, die durch automatisches Layout organisiert werden, kann es vorkommen, dass besonders lange Label den Graphen auseinander ziehen. Der Graph kann dadurch unübersichtlich werden oder Informationen, die für den Betrachter nicht von Interesse sind, geraten in den Fokus. Daher wird eine Möglichkeit gesucht, um den Platz, den die Label einnehmen, zu verkleinern. Am besten erfolgt dies, ohne dass die Informationen, die sie liefern, verloren gehen. Dazu muss man sich mit zwei konkreten Fragestellungen auseinandersetzten: *wie* und *was* wird gekürzt. Für das „Wie“ werden Strategien zum Kürzen, zum Umstrukturieren oder zum Filtern der Label gesucht. Bezüglich des „Was“ wird hingegen der Frage nachgegangen, ob immer alle Label auf einmal geändert werden sollen oder man sich auf bestimmte Bereiche beschränken kann.

1.2. Verwandte Arbeiten

1.2.1. Kürzen von Labeln

Zu der Frage, *wie* man am besten Label im Graphen kürzt, hat sich bereits Fuhrmann [Fuh11] Gedanken gemacht. Dort wird das Problem zu langer Label an Kanten bereits angesprochen und es werden verschiedene Lösungsansätze vorgestellt. Ein Beispiel zu den jeweiligen Lösungsansätzen findet sich in Tabelle 1.1.

Um Label kompakter zu machen, unterscheidet Fuhrmann zwischen dem Umbrechen

von Text und dem tatsächlichem Kürzen. Diese beiden Bereiche werden jeweils noch in semantisches und syntaktisches Umbrechen beziehungsweise Kürzen unterteilt. Das syntaktische Umbrechen unterscheidet zusätzlich noch zwischen hartem und weichem Umbrechen.

Bei syntaktisch hartem Umbrechen kann mitten im Wort oder Ausdruck umgebrochen werden, während beim syntaktisch weichen Umbrechen nur zwischen Wörtern umgebrochen wird. Das semantische Umbrechen hingegen achtet darauf, dass zusammenhängende Ausdrücke nicht oder nur an passenden Stellen getrennt werden, in dem Beispiel hinter dem Slash, so das dieser das Label in zwei Teile teilt. Aufgeführt wird außerdem die Verkürzung durch Abschneiden der Label mit Dahinterfügen von drei Punkten.

Eine weitere Art der Verkürzung ist die der semantischen Abkürzung. Dort werden Label so gekürzt, dass nur noch die wichtigsten Informationen erhalten bleiben. Fuhrmann schlägt vor, dass bei dem Beispiel nur die beteiligten Signale angezeigt bleiben und nicht die logischen Verknüpfungen, die damit einhergehen. Diese Ideen werden im Folgenden wieder aufgegriffen und weiterentwickelt.

1.2.2. Fokus und Kontext

Im Bezug auf die Frage, *was* am besten gekürzt wird, soll neben der Option, entweder alles oder nichts zu kürzen, ein Fokus-und-Kontext-Ansatz betrachtet werden. Dabei bezeichnet der Fokus diejenigen Elemente eines Graphen, die für den Benutzer zu einem bestimmten Zeitpunkt am interessantesten sind. Der Kontext hingegen besteht aus den umliegenden Objekten, diese können für das Verständnis des Fokus entscheidend sein. Fokus-und-Kontext-Ansätze, die sich im Besonderen auf Label beziehen, sind mir nicht bekannt. Trotzdem soll hier ein kleiner Einblick in den Themenbereich des Fokus und Kontext gegeben werden, um die zu Grunde liegende Technik vorzustellen.

Furnas [Fur86] präsentierte 1986 das Konzept der *generalisierten Fischaugen Sicht*. Diese stellt nur Objekte dar, die für wichtig erachtet werden, die anderen werden nicht angezeigt. Dafür wird jedem Objekt eine *a priori Wichtigkeit* zugewiesen, eine allgemeine Wichtigkeit des Objekts für den Betrachter. Ein weiterer zentraler Begriff ist der des *Grads des Interesses* (engl. *Degree of Interest*). Dieser setzt sich zusammen aus der Differenz der a priori Wichtigkeit und der aktuellen Distanz zum Fokuspunkt. Abhängig von einem bestimmten Schwellwert wird dann entschieden, ob ein Objekt angezeigt wird oder nicht. Die Knoten der in dieser Arbeit betrachteten Graphen besitzen alle die gleiche Wichtigkeit, da hier aufgrund des Layouts nicht bestimmt werden kann, welcher Knoten wie wichtig ist. Daher ergibt sich ein Grad des Interesses der mit abnehmender Distanz zum Fokus immer kleiner wird, was später wieder aufgriffen wird.

Sakar und Brown stellten *Graphical Fisheye* [SB92] vor. Das Prinzip geht zurück auf Furnas, bezieht sich aber ausschließlich auf Graphen. Beim graphischen Fischauge werden die Knoten im Fokus vergrößert, während die umliegenden Knoten mit wachsender

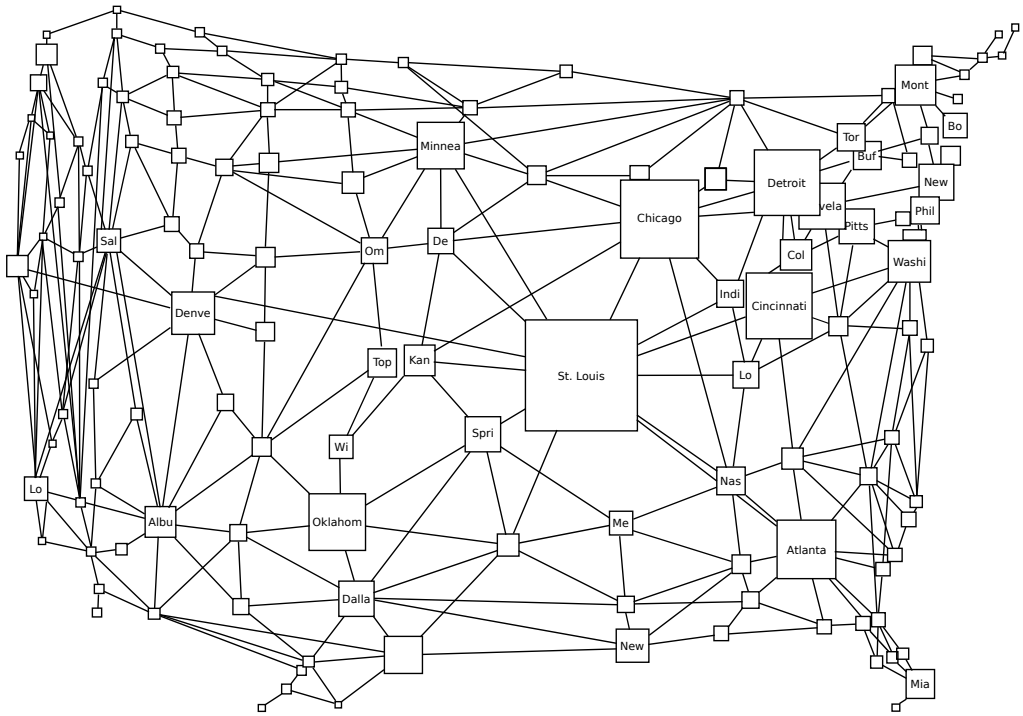

Abbildung 1.3. Graphical Fisheye von Sakar und Brown [SB92] angewandt auf einen Graphen, der die größten Städte der USA miteinander verbindet, der Fokus liegt auf St.Louis.

Entfernung zum Fokus verkleinert werden. Um das zu ermöglichen, werden die Positionen der Knoten neu berechnet und die Kanten neu gezeichnet. Der Grad der Details soll proportional zu der Größe des Knotens sein; unter Details fallen dabei auch die Label. Abbildung 1.3 von Sakar und Brown ist ein Beispiel für Graphen mit Labeln, auf den Graphical Fisheye angewandt wird. Dort wird im Fokus das gesamte Label angezeigt, während mit abnehmender Knotengröße auch die Label kürzer werden. Dabei werden nicht mehr passende Zeichen von hinten weggeschnitten, bis am Ende keine Zeichen mehr übrig sind. Die Knoten neu zu verteilen ist nicht Zweck dieser Arbeit, allerdings wird das stärkere Kürzen von Labeln mit abnehmenden Detailgrad aufgegriffen.

Musial und Jacobs [MJ03] beschäftigten sich mit Fokus und Kontext in UML-Diagrammen. Vorgestellt werden verschiedene Detailgrade, die durch unterschiedliche Abstraktionen erreicht werden. Mit abnehmender Nähe zum Fokus wird das Abstraktionslevel höher. Der dort verwendete Grad des Interesses bezieht zusätzlich noch die Frequenz mit ein, mit der ein UML-Knoten vom Benutzer angewählt wird. So wird ein oft besuchter Knoten weniger abstrahiert als ein selten besuchter Knoten mit derselben Distanz zum Fokus. In dieser Arbeit werden durch die Kürzungsstrategien ebenfalls mehrere Detailgrade

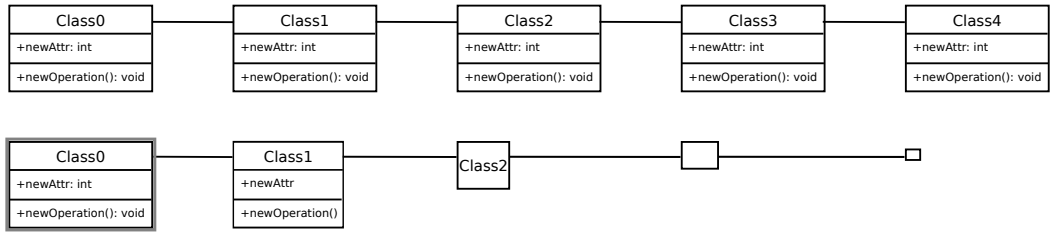

Abbildung 1.4. Ein Beispiel von Musial und Jacobs [MJ03] wie Fokus und Kontext auf ein UML-Diagramm angewandt wird. Der Fokus liegt auf dem linken Knoten.

ermöglicht. Der Gedanke, die Frequenz mit zu berücksichtigen, erscheint lohnenswert und wird hier wieder aufgegriffen.

Fuhrmann [Fuh11] geht ebenfalls auf Fokus und Kontext ein und schlägt verschiedene Lösungsideen vor. Er regt an, dass der Fokus nicht nur den jeweils angewählten Zustand beinhaltet, sondern auch den vorherigen, sodass es zwei Fokusse gibt. Da die Benutzerbewegungen nicht immer vorhersehbar sind, kann es durchaus passieren, dass die Fokusse an zwei unterschiedlichen Enden des Graphen liegen. Ein weiterer Vorschlag ist der eines *Meta Fokus*. Dabei wird der Fokus auf ein bestimmtes Signal gelegt. Das bedeutet, dass alle Label, die dieses Signal beinhalten, im Fokus liegen. Dies wird später in den Kürzungsstrategien wieder aufgegriffen. Weiterhin betont er, dass eine statische Verwendung einer Fokus-und-Kontext-Strategie nicht ausreichend ist und empfiehlt, die Auswahl einer Strategie dem Benutzer zu überlassen. Auch dies soll bei der Implementierung für diese Arbeit berücksichtigt werden.

1.3. Leitfaden

Zum Schluss dieses Kapitels noch eine kleine Übersicht über den Aufbau dieser Arbeit: Kapitel 2 stellt die relevanten Technologien vor. Darunter sind zum Beispiel die Entwicklungsumgebung und verschiedene Arten von Graphen, anhand derer die Ansätze getestet werden. In Kapitel 3 werden die Kürzungsstrategien vorgestellt, die bestimmen, wie gekürzt werden soll. Kapitel 4 widmet sich der Frage, was am besten gekürzt wird, indem verschiedene Fokus-und-Kontext-Strategien präsentiert werden. Die Implementierung ausgewählter Strategien im Kontext des automatischen Layouts werden in Kapitel 5 dargestellt. Danach werden diese Strategien in Kapitel 6 mit Metriken wie „Fläche“ und „Durchschnittliche Kantenlänge“ anhand von Beispielgraphen evaluiert. Kapitel 7 liefert eine kurze Zusammenfassung der Ergebnisse und gibt einen Ausblick auf noch offene Themen.

Kapitel 2

Verwendete Technologien

Dieses Kapitel widmet sich den verwendeten Technologien. So soll zum einen vorgestellt werden, womit implementiert wurde. Zum anderen sollen die Technologien eingeführt werden, die das automatische Layout zur Verfügung stellen und somit den Grundbaustein für die Implementierung des LabelManagements bilden.

Außerdem werden im Bereich der Anwendungsfälle zwei visuelle Programmiersprachen vorgestellt. Diese basieren auf verschiedenen Graphen, die spezielle Label haben. Der Aufbau dieser Label wird im Folgenden ein spezifischeres Label Management erlauben.

2.1. Basistechnologien

2.1.1. Eclipse IDE

Eclipse[1] ist eine quelloffene Entwicklungsumgebung. Sie ist im Besonderen für Entwicklungen mit der Programmiersprache Java ausgelegt. Weiterhin unterstützt sie die Entwicklung eigener Plugins, sodass sie beliebig verändert und für eigene Anwendungen angepasst werden kann.

2.1.2. KIELER

Kiel Integrated Environment for Layout Eclipse RichClient (KIELER)[2] ist ein Forschungsprojekt des Lehrstuhls für Echtzeitsysteme und Eingebettete Systeme der Christian-Albrechts-Universität zu Kiel. KIELER basiert auf mehreren Eclipse Plugin-Projekten.

Ziel ist es, das graphische, modellbasierte Design komplexer Systeme zu verbessern [vHF10]. Dies wird unter anderem durch automatisches Layout erreicht. Eine Zielsetzung dabei ist es, die Verständlichkeit von Graphen zu erhöhen. In KIELER wird bereits die Modellierung mit verschiedenen visuellen Sprachen unterstützt.

[1] https://eclipse.org/
[2] https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Home

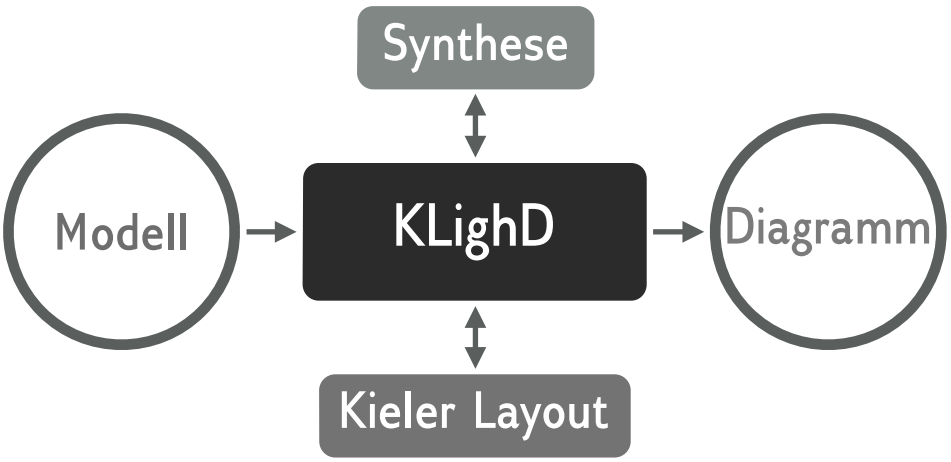

Abbildung 2.1. Der Ablauf von KLighD.

2.1.3. KLighD

KIELER Lightweight Diagrams (KLighD)[3] erstellt nicht persistente Diagramme, die mit automatischem Layout organisiert wurden. KLighD wird daher von KIELER zur Diagrammerstellung verwendet. Dazu bekommt es ein Modell eines Graphen übergeben, dieses kann zum Beispiel mit Hilfe einer textuellen Sprache definiert werden.

Durch die vom Entwickler zu entwickelnde Synthese ermittelt KLighD dann das *Rendering*, das heißt, wie Knoten und Kanten dargestellt werden sollen. KLighD ruft danach einen Layoutalgorithmus auf, der ihm die Information liefert, wie die Knoten angeordnet werden sollen. Aus diesen Informationen wird dann ein Diagramm erstellt. Dieser Ablauf ist in Abbildung 2.1 zu sehen.

2.1.4. KLay Layered

Kieler Layout Layered (KLay Layered)[4] ist ein im Rahmen des KIELER-Projekts entwickelter Layoutalgorithmus. Er ist einer der Layoutalgorithmen, die von KLighD aufgerufen werden können. KLay Layered durchläuft mehrere Phasen und organisiert dabei den Graphen in *Layern*. Dies bedeutet, die Knoten werden derart in Schichten angeordnet, dass die gerichteten Kanten nur in eine Richtung verlaufen. Die Phasen von KLay Layered sind in Abbildung 2.2 zu sehen.

Um das Anordnen in Layern zu ermöglichen, werden zuerst Zyklen, also Kreise im Graphen, entfernt, dann erfolgt die Zuweisung der Knoten in die einzelnen Ebenen. Anschließend werden die Knoten innerhalb der Ebenen derart sortiert, dass die Kreuzungen

[3] https://rtsys.informatik.uni-kiel.de/confluence/pages/viewpage.action?pageId=328115
[4] https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/KLay+Layered

Abbildung 2.2. Die Phasen von KLay Layered.

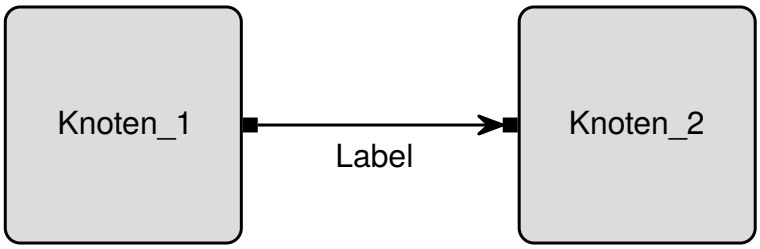

Abbildung 2.3. Beispiel eines KGraphen.

zwischen den Kanten minimiert werden. Danach erst werden die Knoten platziert. Als Letztes wird der Weg der Kanten zwischen den Knoten berechnet.

Zusätzlich zu den Phasen gibt es noch verschiedene Prozessoren, die mit Hilfe der Informationen aus den Phasen das Graphenlayout weiter verändern. Diese werden dementsprechend zwischen den Phasen aufgerufen.

2.1.5. KGraph

KGraphen sind eine Form der Modellierung von Graphen. Ein KGraph besteht aus Knoten und Kanten. Zusätzlich können hier Ports, also Endpunkte für Kanten, auftreten.

Eine Kante kann beliebig viele Label haben. Dabei wird zwischen zentrierten, links- oder rechtsorientierten Labeln unterschieden. Das bedeutet, dass die Label auf der Kante unterschiedlich angeordnet sein könnten. Knoten können auch beliebig viele Label haben. Ebenfalls können an Knoten beliebig viele Ports auftreten. Außerdem können Knoten auch weitere Graphen mit allen bisher genannten Bestandteilen beinhalten. Weiterhin können an einen KGraphen über sogenannte *Properties* Layoutinformationen angeheftet werden.

Um zu verdeutlichen, dass die Elemente Teil eines KGraphen sind, wird ihren englischen Bezeichnern jeweils ein „K“ vorangestellt. So besteht ein KGraph aus KNodes, KEdges, KPorts und KLabeln.

In KIELER dienen KGraphen als Basis für die Umsetzung der visuellen Sprachen, sowie für den Datenaustausch mit den Layoutalgorithmen. Ein kleines Beispiel für einen einfachen KGraphen ist in Abbildung 2.3 zu sehen. Dort sind zwei Knoten, eine Kante, sowie zwei Knotenlabel und ein Kantenlabel abgebildet.

2.2. Anwendungsfälle

Im Folgenden soll genauer auf zwei spezielle Anwendungsfälle der KGraphen eingegangen werden: die SCCharts und Ptolemy.

2.2.1. SCCharts

Sequentially Constructive Charts (SCCharts)[5] sind eine *visuelle Sprache*, die zur Modellierung von *sicherheitskritischen reaktiven Systemen* entwickelt wurde [vHDM+14].

Dabei sind SCCharts insbesondere *Zustandsdiagramme*: sie bestehen aus Zuständen und Zustandsübergängen, den sogenannten *Transitionen*. Da sie für reaktive Systeme entwickelt wurden, gibt es weiterhin Signale, die von außen ins Modell eingegeben und nach außen ausgegeben werden können. Eine Besonderheit der SCCharts ist der *Hostcode*, also Funktionsaufrufe, die SCCharts fremd sind, allerdings durch Bibliotheken eingebunden werden können.

Die Transitionen gehen einher mit einem *Trigger* und einem *Effekt*. Der Trigger gibt an, unter welcher Bedingung der Zustand gewechselt wird. Der Effekt gibt an, was dabei passieren soll. Zum Beispiel können sich als Effekt die Signale, die nach Außen gegeben werden, ändern oder aber es wird ein Hostcode-Ausdruck aufgerufen.

Jeder Zustand kann innere Zustände haben, was bedeutet, dass in diesem Graphen erneut Zustände mit Transitionen angegeben werden können. Zur Darstellung der SCCharts werden KGraphen verwendet. Dabei entspricht jeder Zustand einem Knoten und jede Transition einer Kante. Trigger und Effekt werden als Label der Kante durch einen Schrägstrich getrennt dargestellt.

Ein einfaches Beispiel für SCCharts ist in Abbildung 2.4 zu sehen. Hier gibt es ein eingegebenes Signal `S` und ein auszugebenes Signal `O`. Außerdem gibt es die drei Zustände `A`, `B`, `C`. Die Transition von `A` zu `B` wird genommen, sobald das boolesche Signal `S` wahr wird. Als Effekt wird das boolesche Signal `O` auf wahr gesetzt. Die Transition zum Zustand `C` stellt das genau entgegengesetzte Verhalten dar.

[5] https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/SCCharts

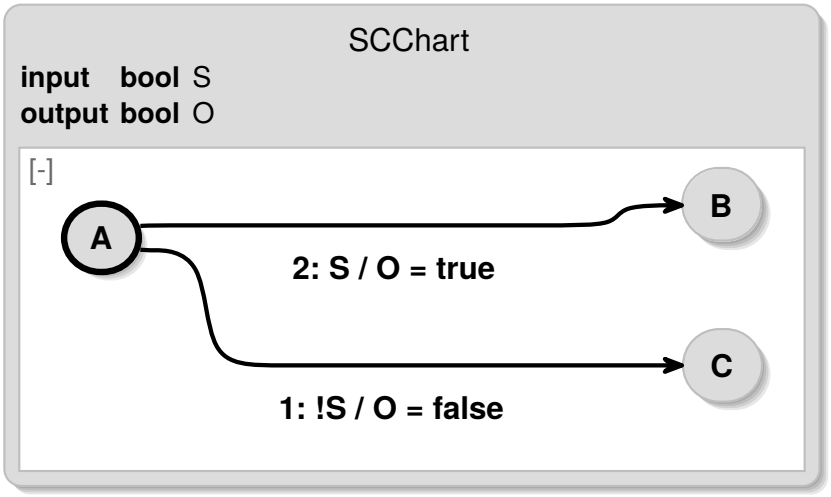

Abbildung 2.4. Beispiel eines einfachen SCChart.

2.2.2. Ptolemy

Ptolemy[6] dient der Modellierung von heterogenen Systemen [EJL+03]. Das bedeutet, dass es möglich ist, viele verschiedene Arten von Systemen zu modellieren, zum Beispiel zustandsbasierte, aber auch ereignisgesteuerte Systeme. Außerdem sind diese Systeme in Ptolemy kombinierbar. Dies geschieht durch die Verwendung verschiedener Hierarchieebenen. Hierbei repräsentiert jede Hierarchieebene nur eine Art von System, jede Ebene ist damit homogen. Dies ermöglicht eine klare Abgrenzung.

Jede Ebene besitzt einen sogenannten *Direktor*, der die Art des Systems und den Takt bestimmt, in dem die Signale oder Ereignisse auftreten. Der wichtigste Bestandteil von Ptolemy sind die *Aktoren*, die sehr unterschiedlich ausfallen können. Beispiele sind Aktoren, die Werte speichern und ausgeben oder aber Aktoren, die Werte berechnen.

Ein wichtiger Aktor ist das *Modal Model*, welches die Möglichkeit bietet, zustandsbasierte Systeme zu modellieren. Das Modal Model nimmt Signale, kann welche ausgeben und verfügt über Zustände und Transitionen. Außerdem treten Kantenlabel in den Modal Models auf; dort sind sie Teil der Transitionen und ähnlich aufgebaut wie die Label der SCCharts.

Eine besondere Eigenschaft ist, dass in Ptolemy Ports an den Aktoren auftreten, diese können ebenfalls mit Labeln versehen sein.

Abbildung 2.5 zeigt ein Beispiel eines in KIELER angezeigten Ptolemy-Modells. Es ist ein Modell eines Autogetriebes. Als Eingabe erhält es den aktuellen Gang der Schaltung und die Position des Gaspedals. Im car model wird dann berechnet, wann hoch geschaltet wird und wie die aktuelle Beschleunigung zu diesem Zeitpunkt ist.

[6] http://ptolemy.eecs.berkeley.edu/

Das Modell ist ein Beispiel für ein Hybridsystem, da die Eingaben von Schaltung und Pedal kontinuierlich erfolgen, während die Entscheidung der Beschleunigung und des Hochschaltens zustandsbasiert getroffen wird.

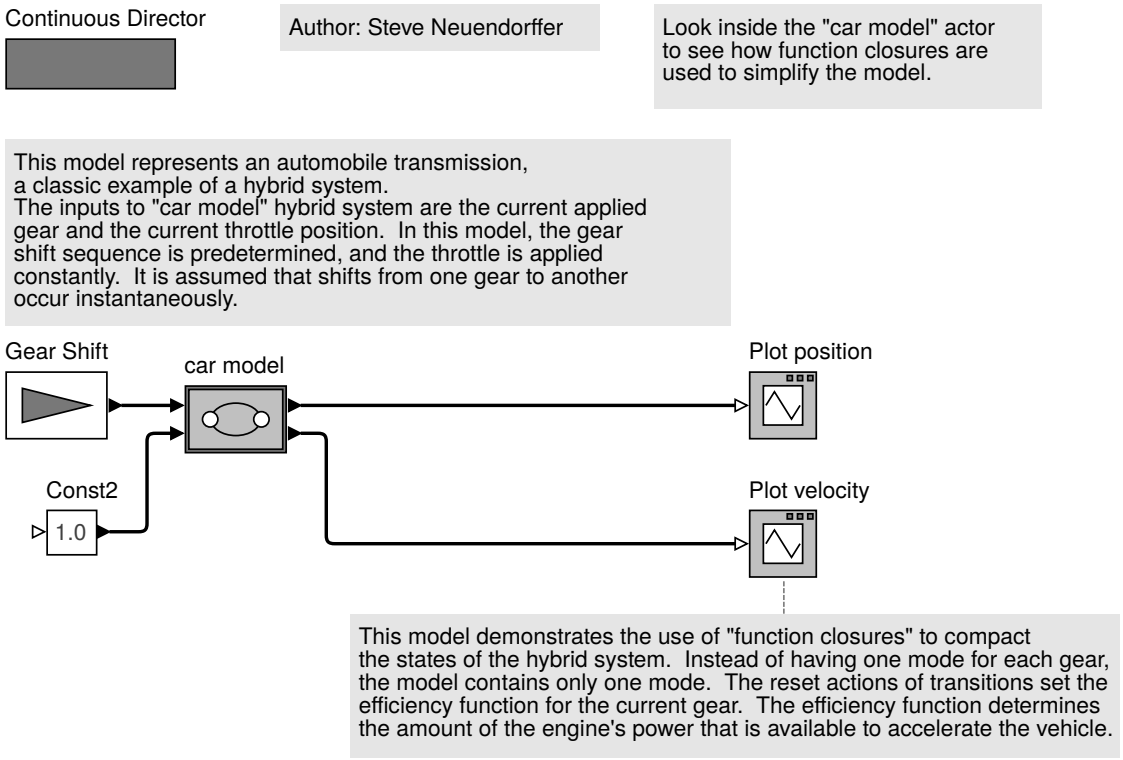

Abbildung 2.5. Beispiel eines Ptolemy-Modells.

Kapitel 3

Kürzungsstrategien

Dieses Kapitel widmet sich der Frage, *wie* lange Label am besten gekürzt werden. Dazu werden zunächst Kürzungsstrategien erörtert, die für alle Zeichenketten und insbesondere auch für Label gültig sind. Danach werden Strategien betrachtet, die sich speziell für SCCharts und für Ptolemy-Diagramme eignen.

Das Ziel aller angegebenen Strategien ist das Kürzen in der Breite. Die einzelnen Strategien unterscheiden sich in ihrer Zielsetzung insofern, dass einige eine bestimmte Zielbreite zu erreichen versuchen und andere, zum Teil zielbreitenunabhängig, nach bestimmten Aspekten kürzen. Die Zielbreite kann entweder vom Benutzer frei gewählt werden oder sie richtet sich nach dem Layoutalgorithmus. Im letzteren Fall wird die Zielbreite durch die Breite des größten Knotens im Layer bestimmt; sind im Layer keine weiteren Knoten enthalten, nimmt der Layoutalgorithmus eine Mindestgröße an. Auch die Auswahl durch den Benutzer sollte durch sinnvolle Größen eingeschränkt werden.

Die Strategien sind für Kantenlabel und, im Kontext von Ptolemy, auch für Portlabel gedacht. Andere Label können zwar mit den Basistechnologien gekürzt werden, dies erscheint aber für die hier betrachteten Graphen nicht sinnvoll, da die anderen Label zumeist nur Bezeichner sind und eine Kürzung keine Bereicherung wäre. Außerdem sind es in den Anwendungsfällen zumeist die Kantenlabel, die die Tendenz dazu haben, zu groß auszufallen.

3.1. Einfache Kürzungsstrategien

Die einfachen Kürzungsstrategien zeichnen sich dadurch aus, dass sie angewendet werden können, ohne dass weitere Informationen über das Label bekannt sind. Das heißt, es gibt keine bestimmte Zeichenreihenfolge und die möglichen Zeichen sind nicht beschränkt. Erlaubt sind damit explizit auch Leerzeichen und Zeilenumbrüche.

Bei diesen Kürzungsstrategien wird hier mit Umbrüchen so verfahren, dass sie zuerst entfernt werden; danach erst wird die Strategie angewandt. Sind Umbrüche in den Labeln zulässig, muss man bedenken, dass die Label bereits umgebrochen worden sein könnten. Dabei gibt es drei Varianten, nach denen dieses erfolgt sein könnte: es wurde mitten im Wort umgebrochen, es wurde nur hinter Worten umgebrochen oder es wurde sowohl hinter Worten umgebrochen als auch in Worten selbst, letzteres aber mit Bindestrichen

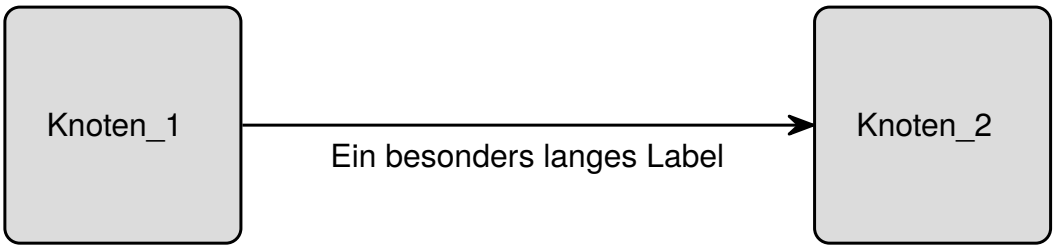

Abbildung 3.1. Beispiel eines einfachen KGraphen mit einem langen Label.

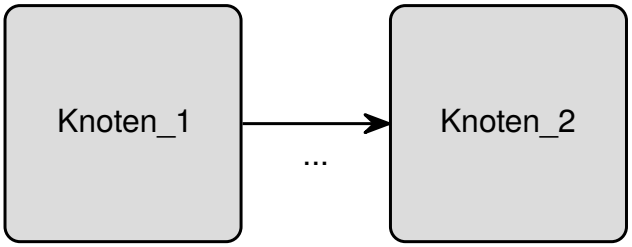

Abbildung 3.2. Beispiel für einen Graphen mit entfernten Labeln.

gekennzeichnet. Je nachdem, mit welcher Art von Umbruch man es zu tun hat, muss dieser unterschiedlich behandelt werden. Stand der Umbruch zwischen zwei Worten, muss ein Leerzeichen eingefügt werden. Wurde im Wort oder hinter einem Bindestrich umgebrochen, wird kein Leerzeichen eingefügt und der Bindestrich entfernt. Welche Strategie ein Benutzer angewandt hat, ist anhand des Layouts ohne Verwendung eines Wörterbuchs nicht ersichtlich. Daher wird für diese Arbeit angenommen, dass hinter Worten oder mit Bindestrich umgebrochen wurde.

Im Folgenden werden die Kürzungsstrategien am Beispiel eines KGraphen erläutert, da dort die Label ebenfalls keiner bestimmten Form folgen. Das Beispiel, zu sehen in Abbildung 3.1, besteht aus zwei Knoten und einer Transition mit dem Label „Ein besonders langes Label“ dazwischen. Anhand dieses Labels sollen die Effekte der einzelnen Strategien verdeutlicht und ein optischer Eindruck über die Verkleinerung des Graphen geliefert werden.

3.1.1. Keine Label

Die einfachste aller Kürzungsstrategien ist das Entfernen sämtlicher Zeichen. Damit erreichen wir die maximale Verkleinerung und die beste Platzoptimierung im Graphen, die unter der Bedingung, dass einzig die Label veränderbar sind, erreicht werden kann.

Der Nachteil dabei ist, dass auch die Informationen, die von den Labeln transportiert werden, verloren gehen. Dabei bleibt jedoch die Aufgabe, dem Nutzer deutlich zu machen, dass es Label gibt, die gekürzt wurden.

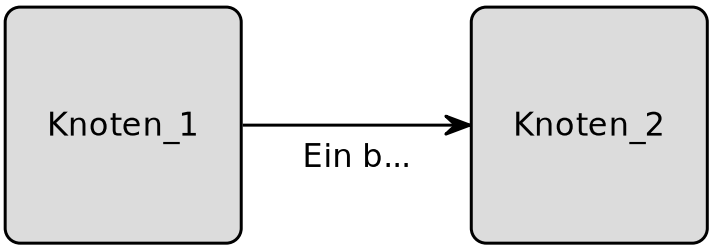

Abbildung 3.3. Graph mit einem syntaktisch gekürzten Label.

Eine Möglichkeit um das Kürzen deutlich zu machen, ist zum Beispiel, drei Punkte an Stelle des gekürzten Labels zu setzen. Dies ist auch eine im Alltag gängige Methode. Dargestellt wird dies im Beispiel, das in Abbildung 3.2 zu sehen ist.

3.1.2. Syntaktisches Kürzen

Syntaktisches Kürzen steht für das Kürzen eines Textes auf der Basis von Zeichen, ohne Rücksicht auf den möglichen Bedeutungsverlust. Im Allgemeinen bedeutet das, dass der Text gegen die Leserichtung abgeschnitten wird, also von rechts Zeichen entfernt werden. Eine syntaktische Abkürzung für das Beispiel wäre dann „Ein besonders l“, eine weitere Möglichkeit ist in Abbildung 3.3 zu sehen. Zur Verdeutlichung, dass ein Wort oder Satz abgekürzt wurde, werden gängigerweise drei Punkte, auch Auslassungspunkte genannt, hinten angefügt.

Die Zielsetzung beim syntaktisches Kürzen ist es, möglichst nah an eine Zielbreite heranzukommen. Leerzeichen und Umbrüche müssen hier besonders betrachtet werden. So sollten keine Leerzeichen zwischen dem letzten Zeichen und den Auslassungspunkten sein. Das Entfernen der Umbrüche sorgt bei dieser Strategie dafür, dass nicht mehrere Zeilen auf einmal syntaktisch gekürzt werden müssen, dies wäre dem Verständnis noch abträglicher. Außerdem können so keine Umbrüche vor den Auslassungspunkten auftreten.

3.1.3. Semantisches Kürzen

Semantisches Kürzen, im Gegensatz zu syntaktischem Kürzen, versucht, eine Zeichenkette so zu kürzen, dass der Sinn nicht verloren geht. Eine semantische Kürzung von „Abkürzung” ist zum Beispiel „Abk”. Da die Bedeutung vom Kontext, in dem sie steht, abhängt, gibt es keine einheitliche Methode des semantischen Kürzens, sondern viele verschiedene Varianten.

Im Alltag am gängigsten ist das semantische Kürzen von Wörtern. Hierbei ist zu beachten, dass für alle Beteiligten die Bedeutung verständlich sein muss. Welche Bedeutung man versteht, hängt auch hier vom Kontext ab. So können dieselben Abkürzungen in unterschiedlichen Kontexten unterschiedliche Bedeutungen haben. Ein einfaches Beispiel

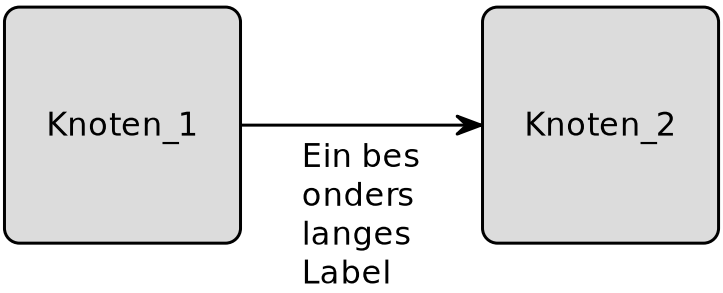

Abbildung 3.4. KGraph mit syntaktisch umgebrochenen Label.

ist „HA“. Während die meisten an die Hausaufgaben von früher zurückdenken, denken Mediziner an den Hausarzt. Es ist also wichtig, dass alle den Kontext kennen.

Semantisches Kürzen ist zumeist nicht an einer bestimmten Zielbreite orientiert, sondern daran, die stärkste Kürzung unter Erhaltung der Bedeutung zu erreichen. Weitere Strategien für semantisches Kürzen werden genauer am Beispiel der Anwendungsfälle betrachtet.

3.1.4. Syntaktisches Umbrechen

Syntaktisches Umbrechen, ähnlich zu syntaktischem Kürzen, verfolgt das Ziel, eine gewisse Zielbreite zu erreichen. Hierbei wird, sobald die Zielbreite erreicht ist, ein Umbruch gesetzt und das Label wird mehr in die Höhe anstatt in die Breite verteilt.

Es erfolgt zwar kein Bedeutungsverlust, da die Zeichen erhalten bleiben, allerdings kann die Lesbarkeit in Mitleidenschaft gezogen werden, da Umbrüche häufig mitten im Wort auftreten. Ein weiteres Risiko ist, dass der Graph zwar in der Breite verkleinert wird, aber stattdessen die Höhe stark zunimmt. Hier muss ein sinnvolles Mittelmaß gefunden werden.

Beim syntaktischen Umbrechen ist das vorherige Entfernen der Umbrüche vorteilhaft, da das Label so nicht noch zusätzlich in die Vertikale gestreckt wird. Ein Beispiel ist in Abbildung 3.5 zu sehen.

3.1.5. Semantisches Umbrechen

Semantisches Umbrechen bedeutet, dass unter Beachtung der Bedeutung umgebrochen wird, also so, dass es an dieser Stelle sinnvoll ist, umzubrechen. Auch hier ist es im Allgemeinen nötig, den jeweiligen Kontext zu kennen, um sinnvoll umzubrechen.

Ähnlich zum semantischen Kürzen gibt es hier viele verschiedene Ansätze, die problemspezifisch sind. Das Erreichen der Zielbreite ist hier schwieriger, da im Vordergrund die Erhaltung der Semantik und nicht das Erreichen der Breite steht. Allerdings kann die Zielbreite als Richtwert für das Kürzen verwendet werden.

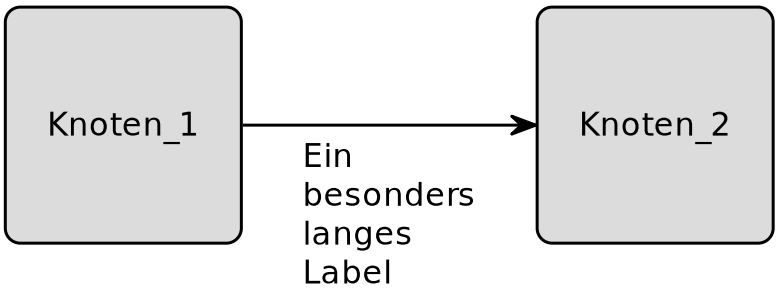

Abbildung 3.5. KGraph mit semantisch umgebrochenen Label.

Eine einfache Umsetzung von semantischem Umbrechen ist die des Umbrechens mit der Orientierung am längsten Wort. Das längste Wort wird gesucht und stellt danach die minimale Zielbreite dar, anhand derer der übrige Text umgebrochen wird. Ist die gegebene Zielbreite größer als das längste Wort, ist diese wieder die Orientierung für die Breite. Die Anwendung dieser Strategie auf das Beispiel ist in Abbildung 3.5 dargestellt. Hier ist das längste Wort „besonders“, dass nun die Zielbreite bestimmt. Die anderen Worte werden jeweils in eine neue Zeile gerückt, da keine zwei mehr gemeinsam in eine Zeile passen.

3.1.6. Kombination von Strategien

Eine weitere Möglichkeit, Label zu kürzen, ist das Kombinieren von verschiedenen Strategien. Man kann dabei mehrere Strategien hintereinander auf das selbe Label anwenden. Hierbei ist zu beachten, dass die Reihenfolge, in der sie angewandt werden, zu unterschiedlichen Ergebnissen führen kann. Beispielsweise wäre semantisches nach syntaktischem Kürzen nicht mehr möglich, umgekehrt allerdings schon.

Ein Beispiel wäre das Label „Hausaufgaben über Abkürzungen“. Hierfür soll angenommen werden, dass beim syntaktischen Kürzen das Label immer nach dem fünften Zeichen, inklusive Leerzeichen, abgeschnitten wird. Beim semantischen Kürzen, soll die umgangssprachlich verwendeten Abkürzungen verwendet werden, also „HA“ für Hausaufgaben und „Abk.“ für Abkürzungen. Wird erst semantisch und dann syntaktisch gekürzt, entsteht das Label “HA üb“. Bei umgekehrter Reihenfolge wiederum entsteht durch syntaktisches Kürzen "Hausa“. Semantisches Kürzen ist darauf nicht mehr möglich.

Wird eine Reihe von Strategien nacheinander auf ein Label angewandt, dann ist es eine Möglichkeit, den Prozess abzubrechen, sobald die erste Strategie das Label erfolgreich gekürzt hat. So kann erreicht werden, das alle Label auf die für sie sinnvollste Weise gekürzt werden. In diesem Zusammenhang ist es sinnvoll, Bedingungen zu formulieren, die eine Strategie nur anwenden, wenn sie erfüllt sind . Ein Beispiel für solche Bedingungen könnte sein, dass nur Portlabel gekürzt werden und alle anderen Labelarten unangetastet bleiben.

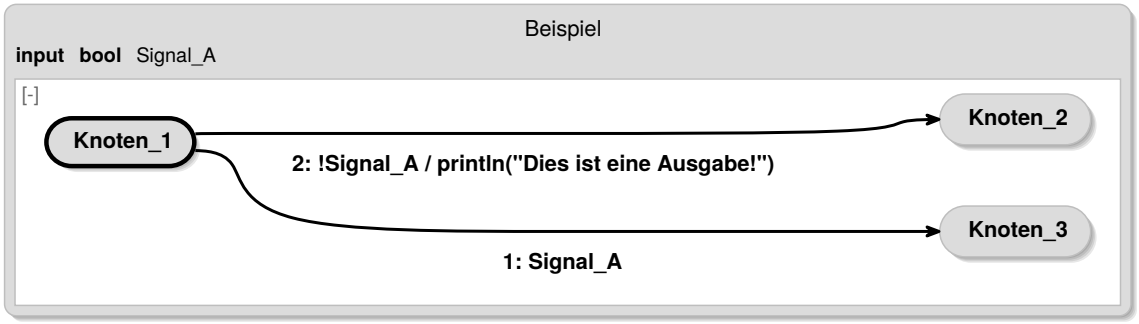

Abbildung 3.6. Beispiel eines SCChart mit langen Labeln.

3.2. Kürzungsstrategien für SCCharts

Die Label der SCCharts folgen einer ganz bestimmten Form, was einen sehr viel spezifischeren Umgang mit ihnen ermöglicht. Sie sind nach folgender Struktur aufgebaut: [<Priorität>:][<Trigger>][/<Effekt>]. Hierbei sind die Ausdrücke in den eckigen Klammern optional, die spitzen Klammern wiederum weisen Platzhalter aus. Die Priorität steht für einen Zahlenwert, der angibt, welche Transitionsbedingung zuerst geprüft wird. Trigger und Effekt sind jeweils Zeichenketten. Sie bestehen aus Signalen, die ebenfalls Zeichenketten sind, logischen und booleschen Operatoren, sowie Zahlenwerten. Im Effekt sind zudem Zuweisungen möglich. Eine weitere Besonderheit, die auftreten kann, ist Hostcode. Dieser besteht aus einem Funktionsaufruf, einer öffnenden Klammer, einer Liste von durch Kommata getrennten Argumenten und einer schließenden Klammer. Der Funktionsaufruf ist dabei wieder eine Zeichenkette, die Argumente können ebenfalls Zeichenketten, aber auch Zahlenwerte sein.

Dies sind viele Informationen, die für die Kürzungsstrategien berücksichtigt werden können. Es ist durchaus möglich, die bisher vorgestellten Strategien zu verwenden, doch ist es auch möglich angepasstere Lösungswege zu suchen. Die Vorschläge für Kürzungsstrategien auf Basis der Semantik der Kantenlabel der SCCharts werden im Folgenden vorgestellt.

Das Beispiel, anhand dessen die Strategien demonstriert werden, ist im Original in Abbildung 3.6 zu sehen. Es besteht aus drei Zuständen und zwei Transitionen. Die Transition von Knoten_1 zu Knoten_3 wird nur genommen, wenn das Signal_A gesetzt ist und falls es nicht gesetzt ist, wird die Transition zu Knoten_2 genommen. Wird die Transition von Knoten_1 zu Knoten_2 ausgelöst, wird zusätzlich mit Hilfe eines Hostcode Aufrufs „Dies ist eine Ausgabe!“ auf die Konsole geschrieben.

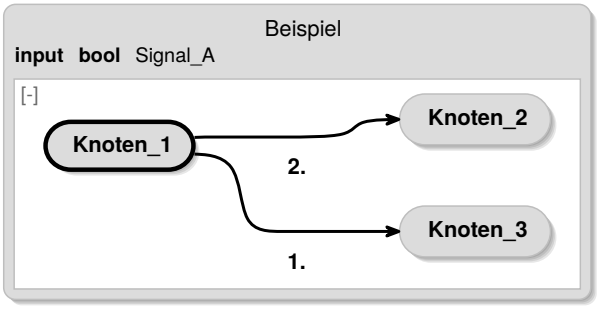

Abbildung 3.7. Beispiel eines SCCharts mit Kürzung bis auf die Transitionspriorität.

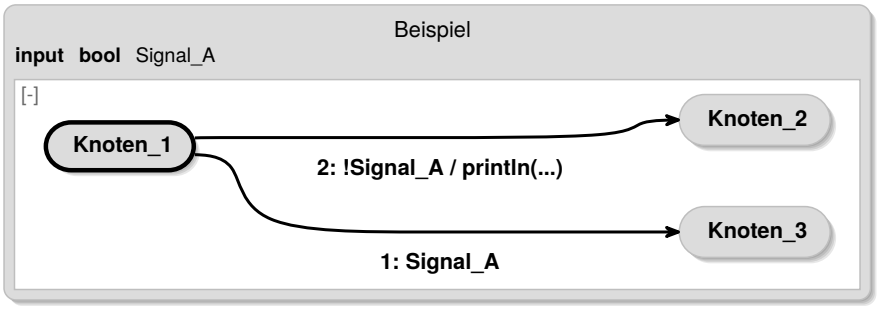

Abbildung 3.8. Beispiel eines SCCharts mit Labeln ohne Hostcode-Argumente.

3.2.1. Nur Transitionsnummern

Eine sehr simple Abwandlung der „Keine Label"- Kürzungsstrategie ist die, dass sämtliche Label auf die Transitionspriorität reduziert werden. So ist klar, welche Prioritäten die Transitionen haben und der Graph wird stark verkleinert. Visualisiert ist dies in Abbildung 3.7.

3.2.2. Keine Hostcode-Argumente

Weiterhin ist es möglich, eine Kürzung herbeizuführen, indem die Argumente vom Hostcode ausgeblendet werden. So ist deutlich, was aufgerufen wird, nur Details des Aufrufs gehen verloren. Besonders in großen Modellen mit viel Hostcode, der sich laufend wiederholt, kann dies hilfreich sein. Für unser Beispiel bedeutet dies, dass die Information, dass „Dies ist eine Ausgabe!" ausgeben wird, verborgen wird. Dies ist in Abbildung 3.8 zu sehen.

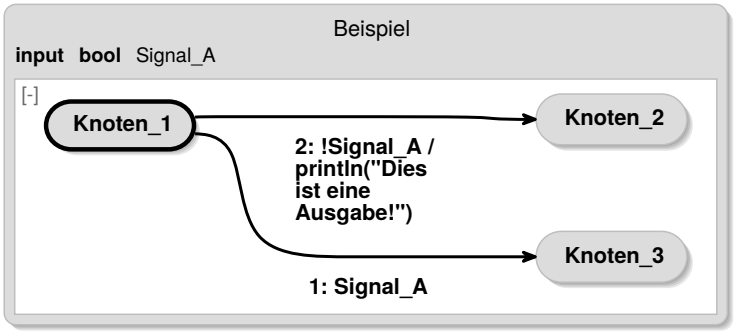

Abbildung 3.9. Beispiel eines SCCharts mit semantisch umgebrochenen Labeln.

3.2.3. Semantisches Umbrechen

Für SCCharts ist es möglich, das semantische Umbrechen konkreter zu gestalten, als nur nach dem größten Wort zu suchen. So kann der Text hinter jedem Schrägstrich umgebrochen werden. Weiterhin kann in Trigger und Effekt gemeinsam nach dem größten Wort gesucht werden, sodass beide Hälften auf dieselbe Größe umgebrochen werden. Angewandt wurde das auf die Beispielgraphik in Abbildung 3.9. Das größte Wort ist hier `println("Dies`, da dort kein Leerzeichen zwischen Klammer und dem Beginn des Ausgabetextes ist.

Ein weiterer Ansatz ist der, die unterschiedlich starke Bindung der Operatoren zu verwenden. Dann würde verhindert werden, dass Umbrüche Zuweisungen trennen und ähnliches. Dies könnte die Lesbarkeit beim Umbrechen sogar erhöhen. Ein Beispiel dafür ist ein Ausdruck wie `(SignalA == SignalB || SignalC != SignalD)`. Zeilenumbrüche wären vor beziehungsweise hinter dem logischen Oder denkbar. Auf diese Weise bleiben die Vergleiche, die den Kern des Ausdrucks bilden, zusammen.

3.2.4. Semantisches Kürzen

Ein Ansatz des semantischen Kürzens, der bereits von Fuhrmann [Fuh11] stammt, ist der, dass nur noch die Signale angezeigt und sämtliche Operatoren weggekürzt werden. Doppelte Signale werden nur einfach dargestellt und die verbliebenen werden mit Komma getrennt aufgelistet. Die Kürzung des Beispiels ist in Abbildung 3.10 dargestellt. Hier ändert sich in der Transition zu `Knoten_2`, dass `Signal_A` und der Hostcode nun durch Komma getrennt sind. Auch der Operator wird nicht mehr angezeigt.

3.2.5. Kürzen von Signalen

Eine andere Variante des Kürzens ist es, alle Signale und Hostcode einzeln zu kürzen. Dies kann durch syntaktisches, wenn möglich auch durch semantisches, Kürzen geschehen. Die logischen Operatoren und der Schrägstrich des Triggers bleiben dabei bestehen. Auf diese

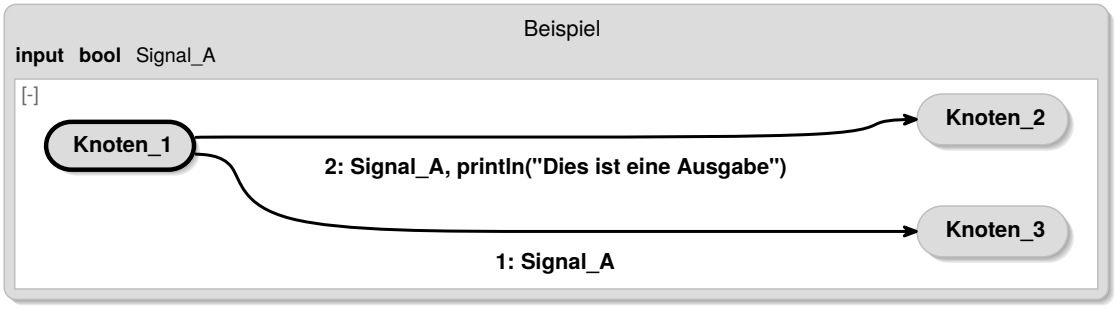

Abbildung 3.10. Beispiel eines SCCharts gekürzt auf Signale und Funktionsaufrufe.

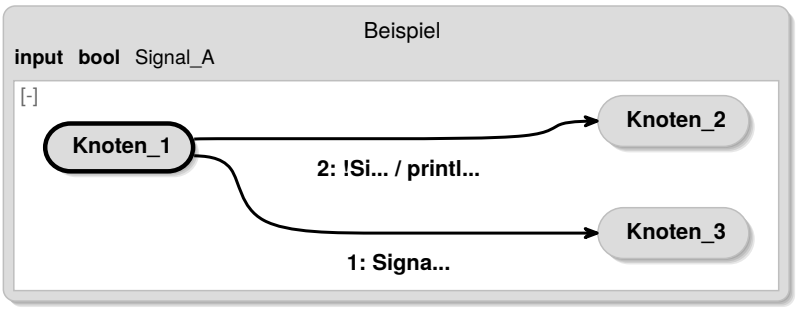

Abbildung 3.11. Beispiel eines SCCharts mit syntaktisch gekürzten Signalen.

Weise lassen sich die logischen und booleschen Zusammenhänge erhalten. Die Zielbreite wird dann auf die Bereiche zwischen den Operatoren aufgeteilt. Eine Visualisierung unter Verwendung von syntaktischem Kürzen ist in Abbildung 3.11 zu betrachten.

3.2.6. Umbennen von Labeln

Ein komplett anderer Ansatz ist der des Kürzens durch Umbenennen. Statt Informationen zu verbergen, werden sie hier nur neu verpackt. Da die Hauptursache zu langer Label häufig die langen Namen sind, wird das Problem so direkt angegangen. Namen, die überlang sind, werden in eine gekürzte Variante oder dem Alphabet folgend umbenannt. Nachteil einer solchen Methode ist allerdings, dass der Benutzer sich danach neu orientieren muss oder schlimmstenfalls den Überblick, den zu gewinnen angestrebt wurde, ganz verliert. Ein Beispiel wäre `Signal_A`, das zu `SigA` gekürzt wird.

3.2.7. Kombinationen

Auch bei den SCCharts ist das Kombinieren von Strategien möglich. Eine Strategie, die das semantische Kürzen variiert, ist folgende: Das semantische Umbrechen der Label zusammen mit dem Entfernen der Hostcode-Argumente unter Aussparung der Operatoren.

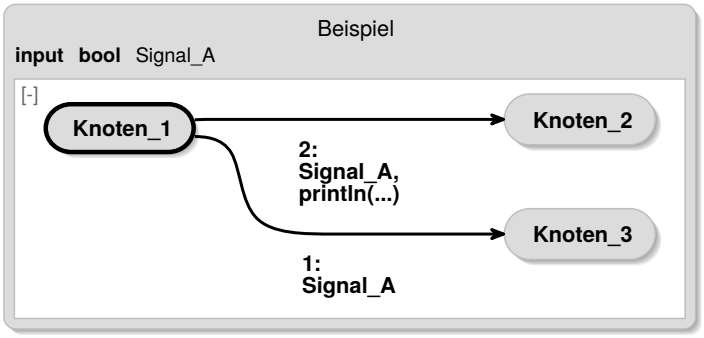

Abbildung 3.12. Beispiel eines SCCharts mit einer Kombination aus HodeCode, semantischem Umbrechen und semantischem Kürzen.

So werden die Signalnamen und der Rumpf des Hostcodes abgebildet. Das Ganze wird dann zusätzlich umgebrochen, sodass noch mehr Breite weggekürzt werden kann. Zu sehen ist dies in Abbildung 3.12.

Eine Variante ist es, verschiedene Kürzungsstrategien auf Trigger und Effekt anzuwenden. So kann im Trigger der Hostcode entfernt werden, während der Effekt syntaktisch gekürzt wird.

3.3. Kürzungsstrategien für Ptolemy

In Ptolemy treten Label als Knotenbezeichner, als Portlabel und in den Modal Models auf. Das Auftreten der Label in Modal Models entspricht denen der SCCharts in Transitionen. Daher können diese mit den bisher vorgestellten Strategien gekürzt werden, vor allem mit denen für die SCCharts.

Zu betrachten bleiben nur noch die Portlabel, welche bisher grundsätzlich ausgeblendet werden. Nur über einen Tooltip konnte bislang die Bezeichnung eines Ports in Erfahrung gebracht werden. Es soll dem Benutzer zusätzlich die Möglichkeit gegeben werden, zu wählen, ob die Label angezeigt, nicht angezeigt oder nur für einen selektierten Knoten angezeigt werden.

Kürzen ist hier nicht weiter sinnvoll, da die Label zumeist aus nur einem Wort bestehen. Daher ist semantisches Umbrechen nicht möglich und alle weiteren Kürzungsstrategien würden nicht zu besserem Verständnis führen.

Kapitel 4

Fokus und Kontext

Bisher wurde davon ausgegangen, dass alle Label im Graphen auf einmal verändert werden. Stattdessen können auch nur bestimmte Label im Graphen gekürzt werden. Dafür soll hier ein Ansatz von Fokus und Kontext für Label vorgestellt werden.

Im Allgemeinen besagt die Fokus-und-Kontext-Strategie, dass die für den Benutzer aktuell interessanten Elemente im Fokus liegen, die anderen entsprechend im Kontext. Im Graphen bezieht sich Fokus und Kontext in den verwandten Arbeiten hauptsächlich auf Knoten und eher passiv auch auf die Label. Hier hingegen sollen Fokus-und-Kontext-Strategien vorgestellt werden, die sich im Besonderen auf Label beziehen.

Dabei ist vorstellbar, dass für die im Fokus liegenden Label schwächere und für die im Kontext liegenden stärkere Kürzungsstrategien angewandt werden. Eine Variante davon ist, dass der Fokus unverändert bleibt und der Kontext beliebig gekürzt wird. Um die folgende Beschreibung zu vereinfachen, wird angenommen, dass der Fokus unverändert bleibt. Vor dem Setzen des Fokus seien weiterhin sämtliche Label gekürzt.

Nun stellt sich die Frage, wie der Fokus im Graphen gewählt wird. Es soll davon ausgegangen werden, dass der Benutzer durch Selektion im Graphen navigiert und damit die Elemente auswählt, die für ihn interessant sind. Die verschiedenen Arten, mit denen der Fokus durch Benutzerselektion gesetzt werden kann, sollen nun vorgestellt werden.

Das Beispiel, das zur Verdeutlichung dient, ist in Abbildung 4.1 aufgeführt. Der Graph besteht aus drei einfachen und einem hierarchischen Knoten. Weiterhin gibt es `Signal_A`, `Signal_B` und `Signal_C`, sowie drei Transitionen mit Labeln.

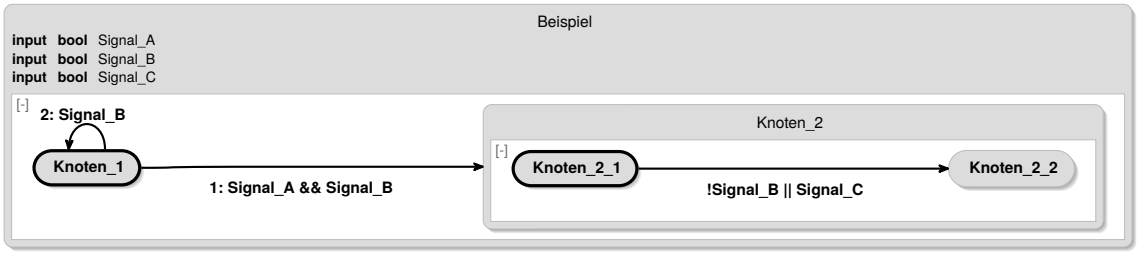

Abbildung 4.1. Beispiel eines hierarchischen SCCharts.

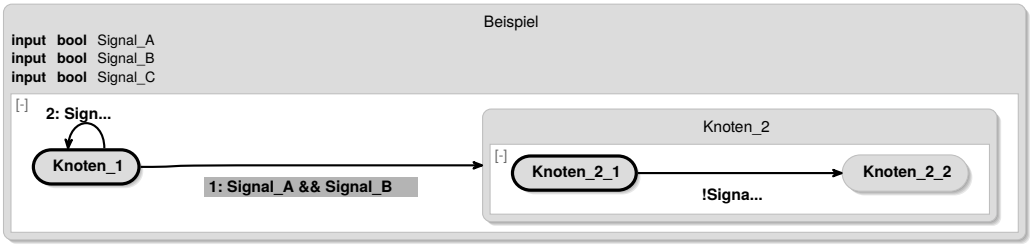

Abbildung 4.2. Beispiel eines ausgewählten Labels im Fokus.

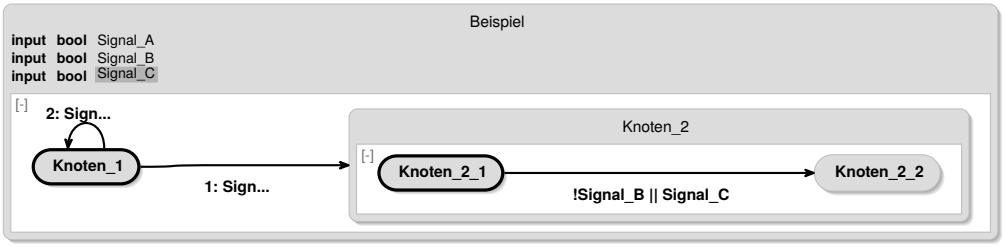

Abbildung 4.3. Beispiel für einen Meta Fokus, der Fokus liegt auf `Signal_C`.

4.1. Fokus auf Selektion

Die einfachste Variante des Setzens von Fokus und Kontext ist es, den Fokus auf das selektierte Objekt zu legen.

Da das Hauptziel dieser Arbeit die Veränderung der Kantenlabel ist, bedeutet diese Variante, dass ein selektiertes Label wieder ungekürzt dargestellt wird. Dies ist in Abbildung 4.2 zu sehen. Hier liegt der Fokus auf dem Label „`1: !Signal_A && Signal_B`“. Alle anderen Label sind gekürzt.

Wird ein Knoten ausgewählt, der selbst weitere Knoten und Kanten enthält, kann der Fokus so weit gefasst werden, dass deren Label ebenfalls im Fokus sind. Alternativ dazu kann beispielsweise festgelegt werden, dass höchstens zwei Hierachieebenen tiefer zum Fokus gezählt werden.

4.2. Metafokus

Die Idee des Meta Fokus stammt von Fuhrmann [Fuh11]. Hierbei wird ein Signal ausgewählt und Label, die dieses Signal beinhalten, bilden den Fokus. Dies bietet die Möglichkeit, dem Signalfluss eines einzelnen Signals zu folgen. In der Abbildung 4.3 wurde der Fokus auf das `Signal_C` gelegt. Da dieses in diesem Beispiel nur in `Knoten_2` auftritt, bleibt dort das Label unverändert, während die anderen Label gekürzt werden.

4.3. Detaillevel

Eine weitere Anpassungsmöglichkeit ist es, mit verschiedenen Detailleveln zu arbeiten, wie bereits Musial und Jacobs [MJ03]. Das bedeutet, dass mit zunehmender Distanz zum Fokus die Informationen, die durch die Label dargestellt werden, abnehmen. Für das Label Management bedeutet dies, dass der Kürzungsgrad steigt.

Ein Beispiel dafür wäre, dass der Fokus in die Mitte des Graphen gesetzt wird. Dann könnten Label in direkter Umgebung zum Fokus semantisch umgebrochen werden, während Label am Rand des Graphen ausgeblendet werden. Die Verwendung anderer Kürzungsstrategien ist auch hier denkbar. Auch ist es möglich, hier mit verschiedenen Schriftgrößen zu arbeiten, indem mit zunehmender Distanz die Schriftgröße verringert wird.

4.4. Frequenzfokus

Eine weitere Idee von Musial und Jacobs [MJ03] war es, den Fokus durch die Frequenz, mit der ein Objekt selektiert wurde, zu bestimmen. So geraten Objekte dadurch in den Fokus, dass sie häufig selektiert wurden.

Hierbei entstehen Multifokusse. Ein Beispiel wäre, dass ein Label bereits fünfzig Mal angewählt wurde und nun ein bisher unselektiertes Label angeschaut werden soll. Dieses muss für den Moment auch in den Fokus. Außerdem stellt sich die Frage einer zeitlichen Komponente, denn wenn zu Beginn des Arbeitens ein bestimmtes Label interessant war und später nicht wieder selektiert wurde, sollte es nicht mehr im Fokus sein.

Verwendet man zusätzlich zum Frequenzfokus verschiedene Detaillevel, kann es mit Multifokusse zu Überschneidungen der Level kommen. Angenommen das Detaillevel bestimmt sich mit der Distanz zum Fokus. Ein Knoten hat nun eine Distanz von 3 zu Fokuspunkt A und eine Distanz von 5 zu Fokuspunkt B. Dann muss sich eine Strategie überlegt werden, wie mit diesem Problem umgegangen wird. Eine Möglichkeit dafür wäre eine Maximumsfunktion, die immer das höchstmögliche Detaillevel liefert.

Kapitel 5

Implementierung

Dieses Kapitel befasst sich mit der Umsetzung der bisher genannten Strategien im Kontext von KLighD und KLay Layered. Dafür soll ein allgemeiner Überblick über den Aufbau des Label Managements und wie es aufgerufen wird, gegeben werden. Weiterhin wird die Implementierung der wichtigsten Kürzungsstrategien vorgestellt.

5.1. Framework

Um den Kontext besser verstehen zu können, soll hier die Umgebung und ihr Ablauf genauer erläutert werden. KLay Layered durchläuft verschiedene Phasen um das Layout des Graphen zu berechnen. Zwischen den Phasen werden dabei zusätzlich verschiedene Prozessoren aufgerufen, die ebenfalls das Layout verändern.

Für das Label Management wurde ein solcher Prozessor hinzugefügt, was in Abbildung 5.1 zu sehen ist. Dieser wird nach dem Zuordnen der Knoten und Label zu Ebenen aufgerufen. Sobald bekannt ist, wie und in welcher Ebene die Knoten und Kantenlabel angeordnet sind, kann das Label Management aufgerufen werden. Da die Zielbreite der jeweiligen Label auch durch den breitesten Knoten der Ebene bestimmt werden kann, ist es notwendig, dass die Objekte bereits Ebenen zugeordnet sind.

Die jeweilige Synthese, die von KLighD angestoßen wird, ist dafür zuständig, das Label Management überhaupt zu aktivieren. Hierdurch wird angegeben, auf welche Art und was gekürzt werden soll. Es werden dort also Kürzungsstrategien als Eigenschaft des Graphen registriert.

Die in KIELER implementierte Synthese hat zusätzlich Zugriff auf die Angaben, die der Benutzer in der Benutzeroberfläche zu den Kürzungsstrategien eingibt. Um das Kürzen auszuführen, werden diese ausgewertet und an die entsprechende Kürzungsstrategie weitergegeben.

In KIELER gibt es jeweils eine Synthese für die KGraphen, die SCCharts und Ptolemy-Modelle. Die individuellen Synthesen ermöglichen es, verschiedene Kürzungsstrategien für die verschiedenen Graphenarten zu registrieren.

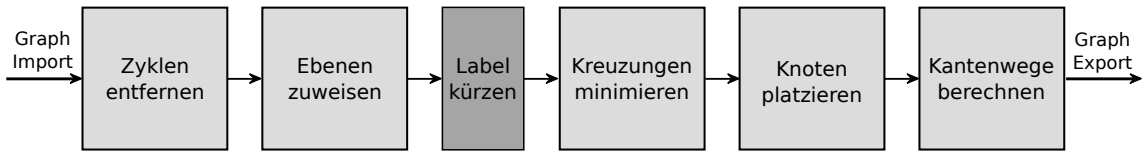

Abbildung 5.1. Die Phasen von KLay Layered erweitert um das Label Management.

5.2. Aufbau des Label Managements

Der Aufbau des Label Managements ist Abbildung 5.2 dargestellt. Dieses soll im Folgenden genauer erläutert werden.

Das Label Management wird in KLay Layered durch einen Prozessor aufgerufen, den LabelManagementProcessor. Der LabelManagementProcessor sucht pro Ebene nach Labeln und stößt für diese das Kürzen an. Das Kürzen wird nur angestoßen, wenn in den Eigenschaften des Graphen durch die Synthese festgehalten wurde, dass überhaupt gekürzt werden soll. Außerdem berechnet er die Zielbreite, die ein Label annehmen kann. Dazu sucht er nach dem breitesten Knoten der Ebene, in dem sich das entsprechende Label befindet.

Aufgerufen wird das Kürzen über das Interface ILabelManager. Dieses stellt eine Methode manageLabels zur Verfügung, die ein Label und die gewünschte oder berechnete Zielbreite erwartet.

Implementiert wird das Interface bislang ausschließlich vom AbstractKlighdLabelManager. Dieser stellt eine abstrakte Oberklasse zur Implementierung von Kürzungsstrategien dar. Sämtliche Kürzungsstrategien werden als einzelne LabelManager implementiert, die von dieser abstrakten Oberklasse erben. Die LabelManager sind dabei so aufgebaut, dass mit Hilfe des Labels und der Zielbreite, die übergeben werden, nichts weiter getan werden muss, als den Text des Labels zu verändern. Der AbstractKlighdLabelManager kümmert sich um die Vor- und Nachbereitung des Labels. Außerdem liefert er eine Hilfsmethode, die beim Kürzen der Label hilft.

Bei den LabelManagern kann zwischen den atomaren und den kombinierenden LabelManagern unterschieden werden. Hierbei verändern die atomaren LabelManager das Label direkt. Zu dieser Kategorie zählen insbesondere die LabelManager für die vorgestellten Basiskürzungsstrategien. Die kombinierenden LabelManager hingegen benötigen zum eigentlichen Kürzen weitere AbstractKlighdLabelManager. Dazu zählen die kombinierenden Strategien, die eine Strategie unter einer Bedingung ausführen, aber auch die Strategien, die eine Reihe von Strategien auf ein Label anwenden.

Im folgenden sollen der AbstractKlighdLabelManager und die atomaren LabelManager genauer betrachtet werden.

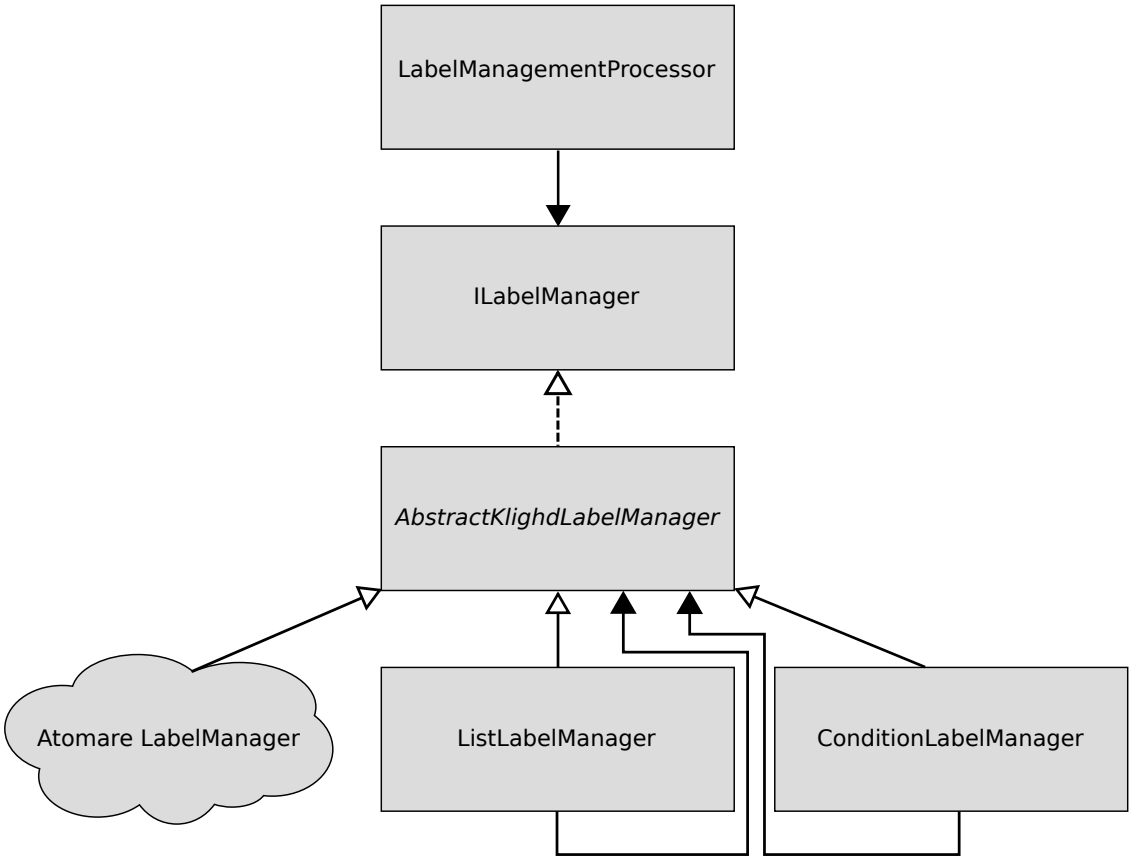

Abbildung 5.2. UML-Diagramm des Label Management Frameworks.

5.3. AbstractKlighdLabelManager

Das Klassendiagramm des AbstractKlighdLabelManagers ist in Abbildung 5.3 zu sehen. Der AbstractKlighdLabelManager ist die abstrakte Oberklasse aller LabelManager. Zudem implementiert er das ILabelManager-Interface und somit die vorgegebene Methode `manageLabels`. Diese wird mit genutzt, um das Kürzen der Label vor- und nachzubereiten. So wird überprüft, ob das Label, das als Objekt übergeben wird, tatsächlich ein Label des KGraphen, also ein KLabel ist. Auch werden nur die Label gekürzt, die im Kontext sind, dies wird mit der Methode `isInContext` überprüft.

Es bleibt die Frage, auf welche Breite gekürzt wird. Hier besteht die Möglichkeit, sich am breitesten Knoten des jeweiligen Ebene zu orientieren. Diese Breite wird direkt vom Prozessor an den AbstractKlighdLabelManager übergeben. Alternativ kann die Breite durch den Benutzer vorgegeben werden. Diese muss dann bei der Registrierung der Kürzungsstrategien mit angegeben werden. Dies geschieht über die Methode `setTargetWidth`.

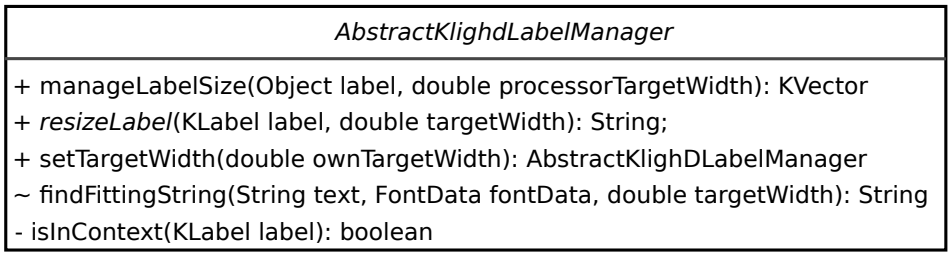

Abbildung 5.3. Klassendiagramm des AbstractKlighdLabelManager.

Darüber wird dem AbstractKlighdLabelManager mitgeteilt, dass er nun die übergebene Breite verwenden soll.

Ist das Objekt ein KLabel und liegt es im Kontext, so wird es gekürzt. Zum Kürzen wird die resizeLabel Methode aufgerufen. Diese wird von den jeweiligen LabelManagern mit den jeweiligen Kürzungsstrategien implementiert und liefert den neuen Labeltext als String zurück. Daher bedarf es einer gewissen Nachbereitung der Label. Der Labeltext muss verändert werden, außerdem wird die Größe, die das Label nun einnimmt, berechnet. Diese Information ist für die weiteren Berechnungen von KLay Layered notwendig, zum Beispiel bei der Knotenplatzierung.

Der AbstractKlighdLabelManager bietet außerdem die Hilfsmethode findFittingString. Diese liefert für einen String und einer gewissen Zielbreite, den Teil des Strings zurück, der in diese Zielbreite passt. Für die Berechnung der Größe eines Labeltextes werden Schriftart und Schriftgröße mitberücksichtigt. Diese werden über die Eigenschaften des Labels mitgeliefert.

Für die Berechnung des Teilstrings wird ermittelt, wie viele Zeichen im Durchschnitt in die vorgegebene Breite des Labels passen. Der String wird daraufhin auf die gegebene Anzahl an Zeichen gekürzt. Danach gibt es eine Unterscheidung, ob der String nun größer oder kleiner als die gewünschte Zielbreite ist. Ist er größer, wird sich von oben an die Zielbreite angenähert, indem Zeichen weggenommen werden und jeweils die neue Länge berechnet wird. Im Fall, dass das Label kleiner als die Zielbreite ist, werden solange Zeichen hinzugefügt, wie das Label gerade noch kleiner ist. Der Pseudocode dazu ist in Listing 5.1 gegeben.

5.4. Atomare LabelManager

Die atomaren LabelManager sind jene, die die einfachen Kürzungsstrategien implementieren. Hier soll genauer auf die Implementierung eingegangen werden. Die folgenden Klassen erben alle vom AbstractKlighdLabelManager, sodass sie hauptsächlich die resizeLabel Methode implementieren.

```
1  function findFittingString(String string, double targetWidth)
2          currentWidth = calculateSize(string)
3          numberOfCharacters = averageNumberOfSignsForString(string, targetWidth)
4
5          string = substring(0, numberOfCharacters)
6          currentWidth = calculateSize(string)
7
8          //The string is still too big
9          if currentWidth > targetWidth
10                 while currentWidth > targetWidth
11                         numberOfCharacters−−
12                         string = substring(0, numberOfCharacters)
13                         currentWidth = calculateSize(string)
14
15         // There is some space for more characters
16         else
17                 while currentWidth < targetWidth
18                         numberOfCharacters++
19                         string = substring(0, numberOfCharacters)
20                         currentWidth = calculateSize(string)
21
22         return string
```

Listing 5.1. Pseudocode für den findfittingString Algorithmus.

```
1  function resizeLabel(KLabel label, double targetWidth)
2          if targetWidth > ellipsisWidth
3                  return findFittingString(labelText, targetWidth−ellipsisWidth)
4          else
5                  return ellipsis
```

Listing 5.2. Pseudocode für den TruncatingLabelManager.

5.4.1. TruncatingLabelManager

Die Aufgabe des TruncatingLabelManager ist die Umsetzung des syntaktischen Kürzens mit Anfügen der drei Punkte. Dazu muss bei der Berechnung auch die Breite der Punkte mit beachtet werden. So wird die Breite der Punkte von der Zielbreite abgezogen und dann mit Hilfe von findFittingString der passende Stringteil ermittelt. Der dazugehörige Pseudocode ist in Listing 5.2 zu sehen.

```
1  function resizeLabel(KLabel label, double targetWidth)
2          restText = labelText
3
4          while !restText.isEmpty
5                  // find the part of the rest of the string which fits the line
6                  calculatedString = findFittingString(restText, targetWidth)
7
8                  // break if the targetWidth is to small to find a fitting string
9                  if calculatedString.isEmpty
10                         return ""
11
12                 //add the fitting string to result
13                 else
14                         resultText.add(calculatedString + "\n")
15                         restText = restText − calculatedString
16
17         return resultText;
```

Listing 5.3. Pseudocode für den HardWrappingLabelManager.

5.4.2. HardWrappingLabelManager

Der HardWrappingLabelManager implementiert das syntaktisch harte Umbrechen. Dazu wird wiederholt findFittingString aufgerufen. Der gefundene String wird aus dem Urspungstext entfernt und zum Ergebnis gerechnet. Außerdem wird dem Ergebnis ein Zeilenumbruch angefügt. Danach wird für den Ursprungstext erneut findFittingString aufgerufen, dies geschieht solange, bis die Zeichenkette leer ist. Der dazugehörige Pseudocode ist in Listing 5.3 zu sehen.

5.4.3. SoftWrappingLabelManager

Der SoftWrappingLabelManger setzt die Strategie des simplen semantisches Umbrechens um. Hier wird das Umbrechen mit der Orientierung anhand des längsten Wortes verwendet. Dazu wird zuerst das längste Wort gesucht. Falls dieses länger als die Zielbreite ist, wird dessen Länge zur neuen Zielbreite. Dafür werden für jede neue Zeile getestet, ob ein weiteres Wort des Labels hineinpasst. Ist die Zielbreite überschritten, wird ein Umbruch eingefügt und für die nächste Zeile ebenso verfahren, bis der Originaltext untergebracht ist. Der Pesudocode dazu ist in Listing 5.4 zu sehen.

```
1  function resizeLabel(KLabel label, double targetWidth)
2          words= getWordsOf(labelText)
3          size = max(targetWidth, biggestWordSize)
4
5          while !lastWord(words)
6
7                  // test if another word fits into this line
8                  while lineSize < size
9                          currentLineText.add(nextWord(words))
10                         lineSize = sizeOf(currentLineText)
11
12
13                 // no more words fits into the line
14                 if !islastWord(words)
15                         resultText.add(currentLineText + "\n")
16                 else
17                         resultText.add(currentLineText)
18                         break
19
20 return resultText
```

Listing 5.4. Der Pseudocode für den SoftWrappingLabelManager.

5.4.4. PriorityNumberLabelManager

Der PriorityNumberLabelManager implementiert die Kürzungsstrategie der SCCharts, bei der nur noch die Prioritätsnummern angezeigt werden. Dazu wird geschaut, ob das erste Zeichen des Labeltextes eine Zahl ist, auf die ein Doppelpunkt folgt. Der Rest des Labels wird verworfen. Gibt es keine Prioritäten, wird der gesamte Labeltext weggekürzt.

5.4.5. TruncateOnlyHostCodeLabelManager

Der TruncateOnlyHostCodeLabelManager entspricht der Kürzungsstrategie, die Hostcode-Argumente entfernt, um so eine Kürzung herbeizuführen. Dazu wird nach öffnenden Klammern gesucht, denen eine Zeichenkette vorausgeht. Der Text zwischen dieser Klammer und der nächsten gefundenen wird dann durch drei Punkte ersetzt.

5.4.6. TruncateOnlyOperatorsLabelManager

Der TruncateOnlyOperatorsLabelManager setzt das semantische Kürzen um, indem es die im Label beteiligten Signale und Hostcode-Aufrufe auflistet. Dazu wird über die Worte

des Labels iteriert und für jedes Wort überprüft, ob es ein Operator ist. Operatoren werden dann durch Kommata ersetzt. Außerdem werden doppelte Vorkommen von Signalen sowie boolesche Werte und Zahlen, die aus Zuweisungen stammen, gelöscht.

5.4.7. TruncateOnlySignalsLabelManager

Der TruncateOnlySignalsLabelManager bildet das Gegenstück zum TruncateOnlyOperatorsLabelManager: Hier werden statt der Operatoren die Signale gekürzt. Dazu wird die Anzahl der Operatoren im Label gezählt und die Zielbreite durch die Anzahl der Bereiche zwischen den Operatoren geteilt. Die Kürzung der Signale wird dann erneut mit `findFittingString` berechnet.

5.4.8. SementicalWrappingLabelManager

Der SementicalWrappingLabelManager nutzt den SoftWrappingLabelManager um das größte Wort im gesamten Label zu finden. Danach werden Trigger und Effekt einzeln danach umgebrochen. Der Schrägstrich wird in einer neuen Zeile nach dem Trigger platziert, sodass der Effekt zumeist mit ihm in einer Zeile anfängt.

5.5. Kombinierende LabelManager

Die kombinierenden Label Manager sind diejenigen LabelManager, die weitere LabelManager benötigen, um zu kürzen. Auch sie erben vom AbstractKlighdLabelManager, sodass sie nur die `resizeLabel`-Methode implementieren müssen.

5.5.1. ListLabelManager

Möchte man mehrere LabelManager kombinieren oder eine Auswahl an LabelManagern anbieten, wo die erste erfolgreiche Kürzung verwendet wird, so kann man den ListLabelManager verwenden.

Der ListLabelManager bekommt eine Liste von AbstractKlighdLabelManager und ruft deren `resizeLabel`-Methode hintereinander auf. Das jeweilige Zwiscshenergebnis wird von dem nächsten Manager weiterverwendet. Das Ergebnis aller Kürzungen wird danach zurückgegeben. Weiterhin besitzt der ListLabelManager ein boolesches Flag, das signalisiert, ob abgebrochen werden soll, sobald ein String erfolgreich gekürzt wurde oder nicht. So wird nach jedem Aufruf eines LabelManagers überprüft, ob bereits abgebrochen werden soll.

5.5.2. ConditionLabelManager

Ein weiterer kombinierender LabelManager ist der ConditionLabelManager. Der ConditionLabelManager bietet die Möglichkeit, ein Label auf eine bestimmte Bedingung zu prüfen, um zu entscheiden, ob es gekürzt werden soll.

Dafür nimmt dieser ein `Predicate<KLabel>`, also eine Bedingung, die auf einem KLabel aufgerufen wird und einen AbstractKlighDLabelManager. Weiterhin gibt es ein boolesches Flag, das angibt, ob, wenn die Bedingung nicht erfüllt ist, alle Label gefiltert werden oder der Originaltext zurückgegeben wird. Ist die Bedingung für ein Label erfüllt, so wird die `resizeLabel`- Methode des Managers aufgerufen. Eine Anwendung wäre es, nur linksbündige Label zu kürzen. Alternativ kann überprüft werden, wie es für Ptolemy geschieht, ob das Label ein PortLabel ist.

5.5.3. TransitionLabelManager

Der TransitionLabelManager bietet die Möglichkeit, verschiedene Manager für Trigger und Effekt anzugeben. Die verschiedenen LabelManager werden über verschiedene Konstruktoren an den TransitionLabelManager übergeben und dann auf das Label angewandt.

5.6. Umsetzung von Fokus und Kontext

Für die Implementierung der Fokus-und-Kontext-Strategien werden Actions verwendet. Actions sind ähnlich zu `Listenern`, sie horchen auf ein Event und werden aufgerufen, sobald dieses Auftritt. Sie erben von dem Interface `IAction` und implementieren somit die Methode `execute`. Insbesondere können Actions an das Rendering der KGraphen angeheftet werden, dies geschieht in der Synthese. Bei Selektion des entsprechende Elements wird dann die darauf registrierte Action aufgerufen.

In den Actions für Fokus und Kontext werden die Elemente gespeichert, die bereits selektiert wurden. Zudem kann mit Hilfe der neu selektierten Elemente bestimmt werden, welche Elemente nun im Fokus sind. Dass ein Element im Fokus ist, wird in den Eigenschaften des Elements festgehalten. Abbildung 5.4 zeigt einen beispielhaften Aufbau einer Action, die Fokus und Kontext umsetzt.

5.6.1. SelectedElementInFocusAction

Die SelectedElementInFocusAction implementiert die einfache Fokus-und-Kontext-Strategie, die besagt, dass ausschließlich das selektierte Element im Fokus liegt. So wird bei Selektion eines Labels seine Eigenschaft angepasst, sodass es nun im Fokus liegt. Die Action wird an das Rendering der Kantenlabel angebracht.

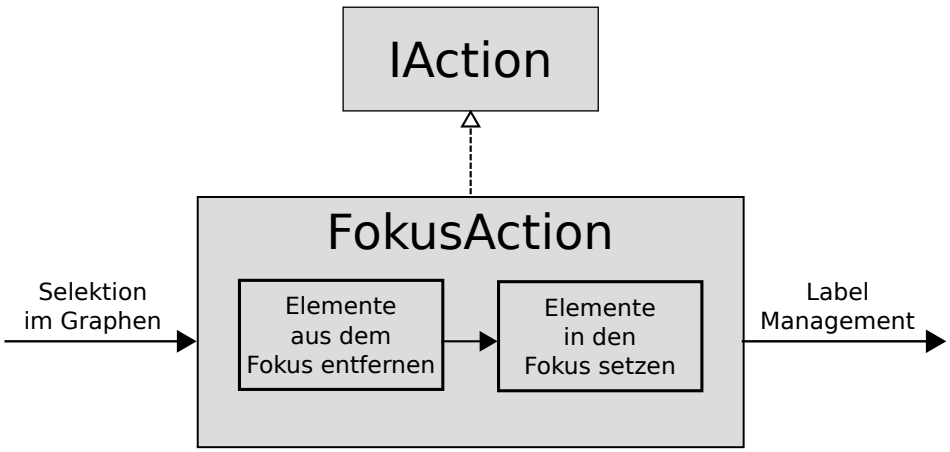

Abbildung 5.4. Ablauf beim Aufruf einer Action für Fokus und Kontext.

5.6.2. MetaFocusAction

Die MetaFocusAction implementiert den Metafokus. Der Fokus wird dabei auf ein Signal von Interesse gelegt, sodass jedes Label, das dieses Signal enthält, im Fokus liegt. Die Action wird an den KLabeln angebracht, die Deklarationen der Signale enthalten. Wird ein entsprechendes Label selektiert, reagiert die Action. Die MetaFocusAction iteriert dann über alle Label des Graphen und schaut jeweils, ob das selektierte Signal enthalten ist. Ist das Signal in einem Label enthalten, wird es in den Fokus gesetzt.

Kapitel 6

Evaluation

Die vorgestellten Kürzungsstrategien sollen hier evaluiert werden. Dazu werden anhand verschiedener Datensätzen die Auswirkungen der Strategien im Bezug auf verschiedene Metriken betrachtet.

6.1. Testumgebung

Für das Testen der Basisstrategien wurden Graphen generiert. Diese besitzen je 50 Knoten mit ein bis zwei ausgehenden Kanten mit je einem Label. Um unterschiedliche Labellängen zu betrachten, wurden je 20 Graphen für unterschiedliche Anzahlen an Zeichen generiert. Die Anzahlen unterteilen sich in die Kategorien 10–20, 20–30, 30–40 und 40–50 Zeichen. Als Labeltext dienen Ausschnitte aus „Lorem ipsum“, die per Zufallsgenerator ausgewählt wurden. Das längstmögliche Wort für ein Label hat damit eine Länge von 10 Zeichen. Die größte Labelbreite beträgt 340 Breiteneinheiten. Ausgewählt wurde dieser Datensatz, um möglichst viele verschieden große Label zu haben, aber auch, um weitere Einflussfaktoren, wie Hierarchie, ausschließen zu können.

Für die Evaluation der Strategien für SCCharts wurde eine Datenbasis aus bereits vorhandenen Graphen verwendet. Dabei handelt es sich um 19 Graphen, von denen elf aus der Steuerung für eine Modelleisenbahn stammen. Die Übrigen stammen aus verschiedenen Projekten des Lehrstuhls und wurden aufgrund ihrer langen Label ausgewählt. Für die Evaluierung wurden die Graphen zerlegt, sodass es keine Hierarchie mehr gibt. So kann ausgeschlossen werden, dass Verfälschungen der Werte durch Neuorganisation der Regionen entstehen. Dadurch ergaben sich insgesamt 245 kleinere Graphen für die Evaluation. Für diese ergibt sich eine durchschnittliche Knotenanzahl von fünf und eine durchschnittliche Anzahl von drei Labeln pro Graph. Die durchschnittliche Länge der Label beträgt 287 Breiteneinheiten, mit einem Minimum von 16 und einem Maximum von 1498. Werte über tausend treten dabei nur vereinzelt auf.

Sämtliche Daten zur Evaluation können auf Anfrage beim Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme eingesehen werden.

Um das Verhalten der Strategien zu betrachten, werden verschiedene Metriken herangezogen. So soll betrachtet werden, wie sich die Kürzung auf die durchschnittliche Labelgröße auswirkt. Weiterhin soll die durchschnittliche Kantenlänge betrachtet werden.

Zuletzt wird die Veränderung des Graphen im Bezug auf seine Fläche betrachtet. Diese Metriken sollen zum Einen demonstrieren, wie sich die Kürzungsstrategien auf die Größe der Label auswirken. Zum Anderen soll dargestellt werden, wie stark sich einer größeren Übersichtlichkeit angenähert werden kann, indem die Verkleinerung des Graphen gemessen wird.

Im Folgenden werden die einzelnen Metriken jeweils für die Basisstrategien und für die SCCharts-Strategien betrachtet. Zu den Basisstrategien zählen hier das syntaktische Kürzen, syntaktisches Umbrechen und semantisches Umbrechen, das sich am längsten Wort orientiert. Für die SCCharts werden zum Einen die zielbreitenunabhängigen Strategien evaluiert. Zu diesen zählen das Entfernen der Label, das Entfernen der Label bis auf Prioritäten, das Entfernen der Operatoren und das Entfernen des Hostcodes. Zum Anderen sollen die zielbreitenabhängigen Strategien betrachtet werden. Diese sind hier das semantische Umbrechen, das vorgestellt wurde und die Kombination aus semantischem Umbrechen, Entfernen von Hostcode und Entfernen von Operatoren. Als Vergleichswert wird zusätzlich das syntaktische Kürzen mit angegeben.

Die folgenden Graphiken sind wie folgt aufgebaut: Die Y-Achse gibt die prozentuale Veränderung in Bezug auf die durchschnittliche Originallänge der jeweiligen Metrik an. Während die X-Achse für die Balkendiagramme angibt, das Ergebnis welcher Strategie durch den jeweiligen Balken repräsentiert wird. Für die übrigen Diagramme gibt die X-Achse die verschiedenen Zielbreiten an.

Bei der Messung der Werte wurden für die Basisstrategien im Bereich von 0–340 Breiteneinheiten alle 20 Zielbreiten eine Messung durchgeführt. Bei den SCCharts wurde für die zielbreitenabhängigen Strategien von 0–200 alle 50 Zielbreiten im Bereich von 200–600 alle 20 Zielbreiten und im Bereich von 600–1000 alle 100 Zielbreiten Messpunkte genommen. Messpunkte für Werte über 1000 wurden ausgespart, da solche Breiten nur vereinzelt vorkommen. Stattdessen wurde mehr im Bereich der durchschnittlichen Labelbreite genommen.

6.2. Labellänge

Als erstes soll die effektive Veränderung der Labellängen betrachtet werden.

Labellängen für die Basisstrategien

Die Abbildung 6.1 zeigt das Diagramm mit den Evaluationsergebnissen für die Basisstrategien. Dort wird dargestellt, wie sich die Länge des Labels pro Algorithmus prozentual zur Originallänge ändert. In der Abbildung ist abzulesen, dass die Labellänge in Abhängigkeit von der Zielbreite steigt. Das bedeutet, je größer die Zielbreite ist, desto breiter sind die Label.

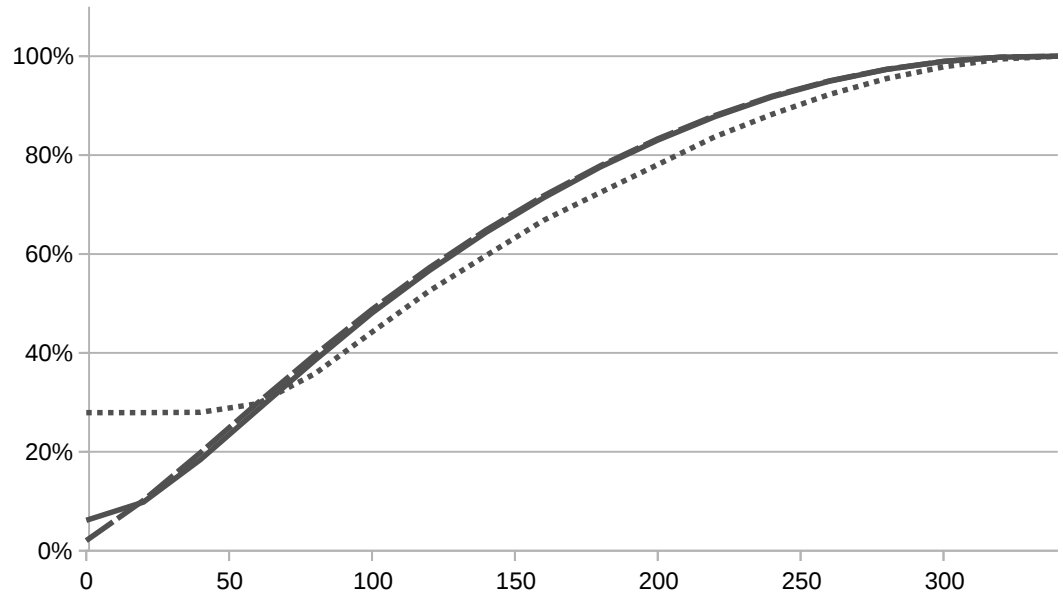

Abbildung 6.1. Prozentuale Veränderung der Labellängen nach der Anwendung der Basisstrategien auf Zufallsgraphen. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für syntaktisches Umbrechen und die gepunktete für semantisches Umbrechen.

Die Kurven für syntaktisches Kürzen und syntaktisch hartes Umbrechen sind nahezu deckungsgleich. Das syntaktisch harte Umbrechen gelangt näher an die Null-Prozent-Grenze. Dies liegt daran, dass beim syntaktischen Kürzen immer die Auslassungspunkte als Darstellung verbleiben, während beim syntaktisch harten Umbrechen bei einer Zielbreite von Null überhaupt kein Label mehr dargestellt wird. Dass die Kurve des syntaktisch harten Umbrechens nicht genau durch den Ursprung geht, liegt daran, dass aufgrund des Renderings für die Label immer ein Leerzeichen eingefügt wird.

Das Ergebnis des semantischen Umbrechens hängt von der Länge der Worte im Label ab. Geht die benutzergegebene Zielbreite gegen Null, so entspricht die Breite des Labels dem längsten Wort. Für die hier verwendeten Worte bedeutet das, dass die Breite des Labels auf höchstens 28% gekürzt wird. Außerdem ist die Kürzung bei höheren Zielbreiten stärker als bei den anderen beiden Algorithmen. Dieses beruht auf der Tatsache, dass der Algorithmus immer ganze Worte in eine Zeile einfügt, so dass die Zielbreite häufig nicht vollständig aufgefüllt wird.

Labellängen für die SCCharts-Strategien

Die Graphik zur Evaluation der zielbreitenabhängigen Strategien ist in Abbildung 6.2 zu sehen. Das syntaktische Kürzen verhält sich äquivalent zu dem Verhalten für die

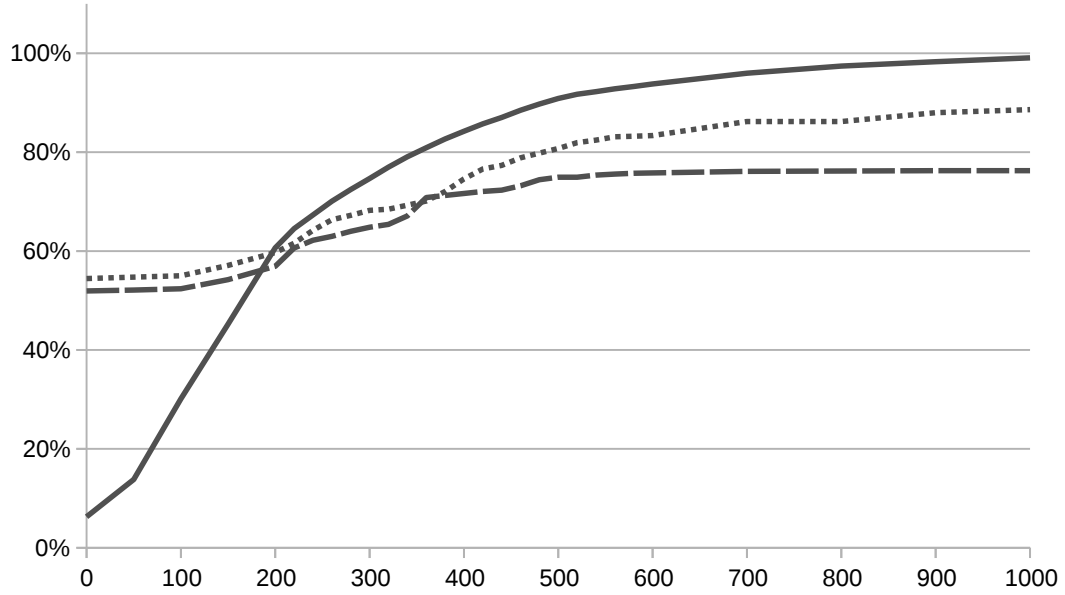

Abbildung 6.2. Prozentuale Veränderung der Labellängen für die zielbreitenabhängigen SCCharts-Strategien. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für die Kombination (Semantisches Umbrechen, Entfernen des Hostcodes, Entfernen der Operatoren) und die gepunktete für semantisches Umbrechen.

Basisstrategien. So können die Label sehr stark gekürzt werden, werden aufgrund der Auslassungspunkte aber nie auf 0% gekürzt.

Semantisches Umbrechen und die Kombination haben einen sehr ähnlichen Kurvenverlauf. Beide erreichen für niedrige Zielbreiten eine Kürzung von etwa 50%, außerdem steigen beide Kurven mit steigender Zielbreite an und flachen gegen Ende ab. Die Kombination liegt dabei immer ein paar Prozentpunkte unter dem semantischen Umbrechen. Dies ist damit zu erklären, dass die Label mit zielbreitenunabhängigen Strategien breites um einen nahezu konstanten Wert gekürzt werden, bevor ebenfalls die Zielbreite durch das semantische Umbrechen mit einfließt. So wird die Kombination immer mindestens so viel kürzen wie das semantische Umbrechen. Die konstanten Werte bei großen Breiten lassen sich der Entfernung von der durchschnittlichen Labelgröße erklären. Die meisten Label sind kleiner als die Zielbreite und werden daher nicht mehr umgebrochen.

Die Evaluation der zielbreitenunabhängigen Strategien ist in Abbildung 6.3 zu sehen. Dabei ist auffällig, dass beim Darstellen keiner Label keine 0% erreicht werden. Dies liegt an oben erwähnten Rendering-Eigenschaften. Es wird immer mindestens ein Leerzeichen als Labeltext verwendet.

Die Strategie des Kürzens auf ausschließlich Prioritätsnummern schafft es, die Label auf eine Länge von etwa 3% zu kürzen. Da große Teile des Labels oder gar das ganze Label,

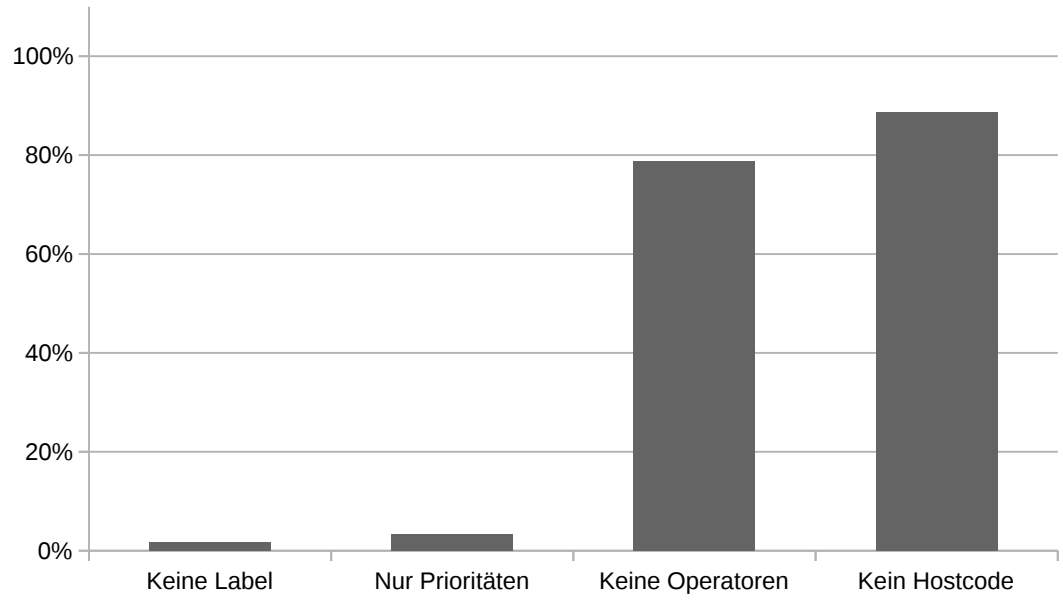

Abbildung 6.3. Prozentuale Veränderung der Labellängen für die zielbreitenunabhängigen SCCharts-Strategien.

wenn keine Prioritäten enthalten sind, nicht dargestellt werden, können sehr niedrige Größen erreicht werden. Das Entfernen der Operatoren und das Listen der Signale erreicht es hier, dass Label auf etwa 80% seiner Originalgröße zu verkleinern. Wären die Label ausschließlich aus sich wiederholenden Signalen aufgebaut, könnte hier ein noch niedriger Wert erreicht werden, da doppelte Vorkommen gelöscht werden. Die Entfernung des Hostcodes, der hier nicht in allen Beispieldiagrammen vorkommt, schafft eine Kürzung auf 90% der Originalgröße.

Vergleicht man die Ergebnisse der zielbreitenunabhängigen mit den zielbreitenabhängigen Strategien, so erreichen die zielbreitenunabhängigen Strategien für große Werte eine größere Kürzung als das semantische Umbrechen.

6.3. Kantenlänge

Die Metrik der Kantenlänge beschreibt, wie sich die Kanten verändern, wenn sich das an sie geheftete Label verändert.

Kantenlänge für die Basisstrategien

Abbildung 6.4 zeigt die Ergebnisse der Evaluation für die Basisstrategien. Zu sehen ist, dass die Kanten mit abnehmender Zielbreite ebenfalls kleiner werden. Die Kanten

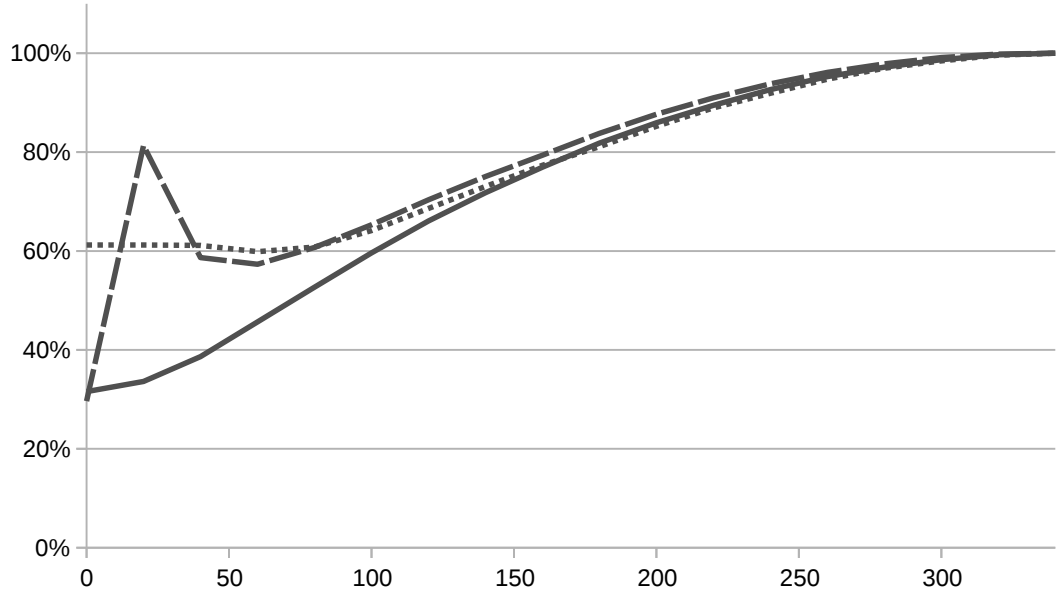

Abbildung 6.4. Prozentuale Veränderung der Kantenlänge nach der Anwendung der Basisstrategien auf Zufallsgraphen. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für syntaktisches Umbrechen und die gepunktete für semantisches Umbrechen.

werden weiterhin nie auf 0% gekürzt. Dies liegt daran, dass es für die Graphen einen Mindestabstand zwischen Knoten gibt. So werden die Kanten höchstens auf etwa 30% ihrer Ursprungsgröße gekürzt.

Syntaktisches Kürzen und semantisches Umbrechen verhalten sich im Vergleich zu ihrem Effekt auf die Labellänge ähnlich. So erreicht syntaktisches Kürzen die nahezu größtmögliche Kürzung, allerdings bleiben immer die drei Punkte übrig, die den Wert erhöhen. Semantisches Kürzen orientiert sich bei größeren Breiten stark an der Zielbreite und es gibt weniger Umbrüche, so ist die Kurve fast deckungsgleich mit den anderen beiden. Bei niedrigen Zeilbreiten werden die Kanten durch die Vergrößerung der Label in der Vertikalen ebenfalls vergrößert. Da die Verkleinerung der Labelgröße bei geringen Breiten konstant wird, wird auch das Kürzen der Kanten konstant. Weiterhin ist zu beobachten, das die Kurve, die für die Veränderung der Labellängen knapp unter der Kurve vom syntaktischen Kürzen lag, nun sehr viel näher dran liegt. Dies liegt daran, dass die Kanten gestreckt werden, um optimal um große Label zu führen.

Auffällig ist der Verlauf der Kurve vom syntaktisch harten Umbrechen. So ist sie für große Breiten deckungsgleich mit den anderen beiden Kurven, während es für geringe Breiten einen Ausschlag nach oben gibt. In diesem Bereich werden Kanten also nicht so stark gekürzt, wie zu erwarten wäre. Das liegt daran, dass die Kanten außen um das Label herumgeführt werden müssen, da diese sich vertikal strecken.

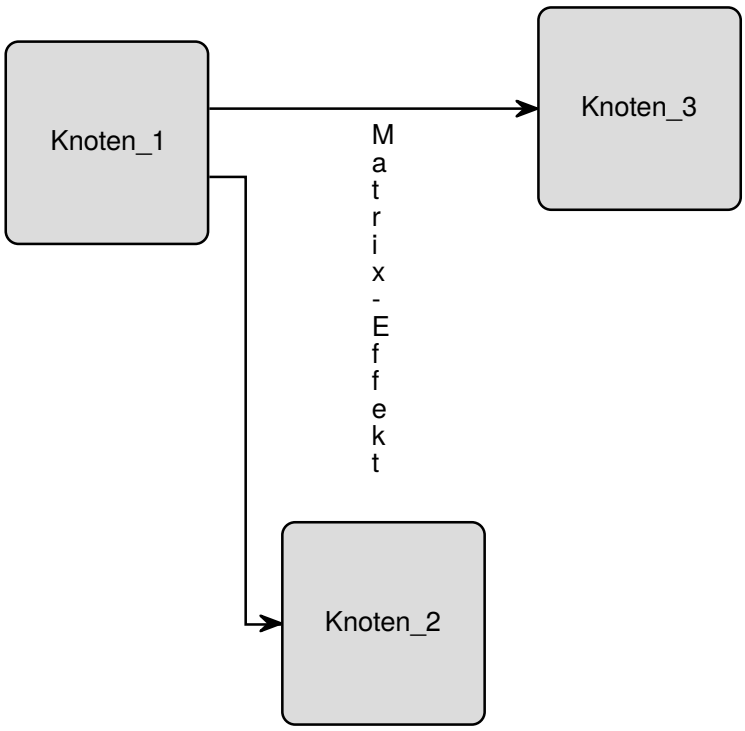

Abbildung 6.5. Beispiel für ein syntaktisch umgebrochenes Label, das die Kanten verlängert.

Abbildung 6.5 dient der Verdeutlichung des Effekts. Hier ist ein Label so umgebrochen, dass die Kante zum `Knoten_2` um dieses herumgeführt werden muss. Bei besonders kleinen Breiten werden die Kanten dann wieder stark verkürzt, da die Label nicht mehr umgebrochen werden können.

Kantenlängen für die SCCharts-Strategien

Die Evaluation der SCCharts im Bezug auf die Kantelänge für die zielbreitenabhängigen Strategien ist in Abbildung 6.6 dargestellt.

Das syntaktische Kürzen steigt hier wieder parallel zur Zielbreite an. Dabei ist aber auch zu beobachten, dass das Maximum an Kürzung hier bei etwa 40% abzüglich der Auslassungspunkte liegt, anstatt wie in Abbildung 6.4 bei etwa 30%. Das bedeutet, dass für die SCCharts weitere Faktoren die Labellänge beeinflussen, wie zum Beispiel die Mindestlänge von Kanten oder Mindestabstände zwischen zwei Knoten. Diese Faktoren sind hier etwas größer als für KGraphen.

Die Kurven für semantisches Umbrechen und für die Kombination verlaufen wieder parallel und hier nahezu ohne Steigung. Das bedeutet, dass obwohl die Breite abnimmt, die Kanten aufgrund der dazugewonnen Höhe der Label wieder verlängert werden. Die

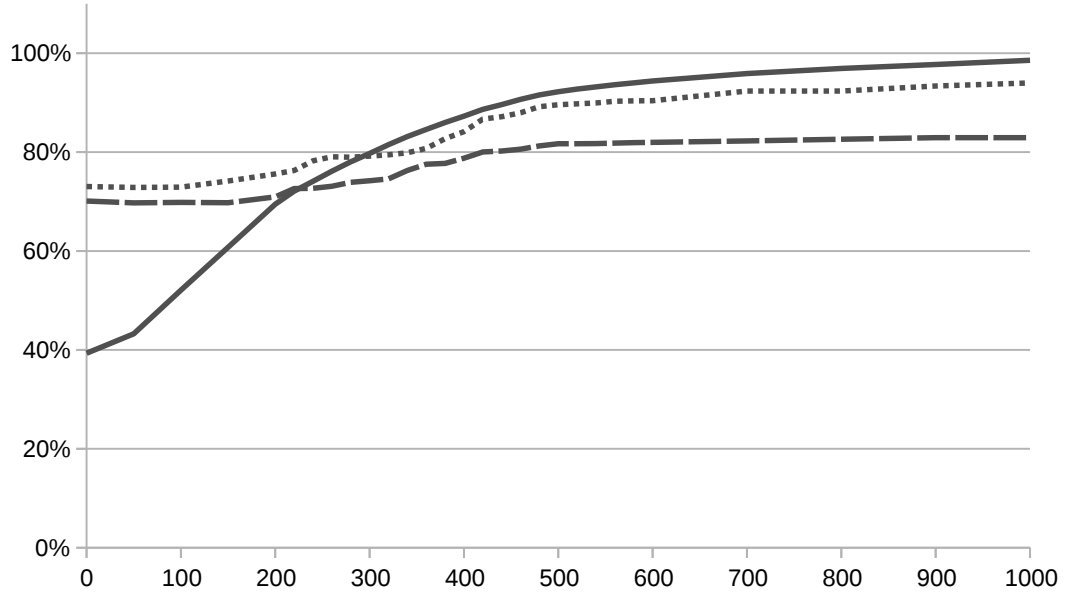

Abbildung 6.6. Prozentuale Veränderung der Kantenlänge für die zielbreitenabhängigen SCCharts-Strategien. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für die Kombination (Semantisches Umbrechen, Entfernen des Hostcodes, Entfernen der Operatoren) und die gepunktete für semantisches Umbrechen.

Effekte gleichen sich fast aus.

Für die zielbreitenunabhängigen SCCharts-Strategien ist die Darstellung der Veränderung der Kantenlänge in Abbildung 6.7 zu sehen. Hier ist erneut abzulesen, dass das Maximum an Verkleinerung bei etwa 40% liegt. Auch die Verkleinerung durch den Algorithmus für das Kürzen auf Prioritätsnummern, liegt sehr nahe an dem Ergebnis für einen Graphen ohne Label.

Für das Entfernen der Operatoren wird eine Verkleinerung auf etwa 80% erreicht, während sich die Kanten für das Entfernen des Hostcodes auf etwa 95% verkleinern. Vergleicht man auch hier die Ergebnisse der zielbreitenunabhängigen mit den zielbreitenabhängigen Strategien, so erreichen das Entfernen der Operatoren für große Werte eine größere Kürzung als das semantische Umbrechen. Das Entfernen des Hostcodes kürzt hier immer weniger als beide zielbreitenabhängigen Strategien.

6.4. Breite des Graphen

Im Folgenden soll die Metrik der Breite des Graphen betrachtet werden. Da insbesondere die KGraphen in horizontalen Ebenen organisiert sind, ist diese Metrik interessant für den Effekt auf die Fläche.

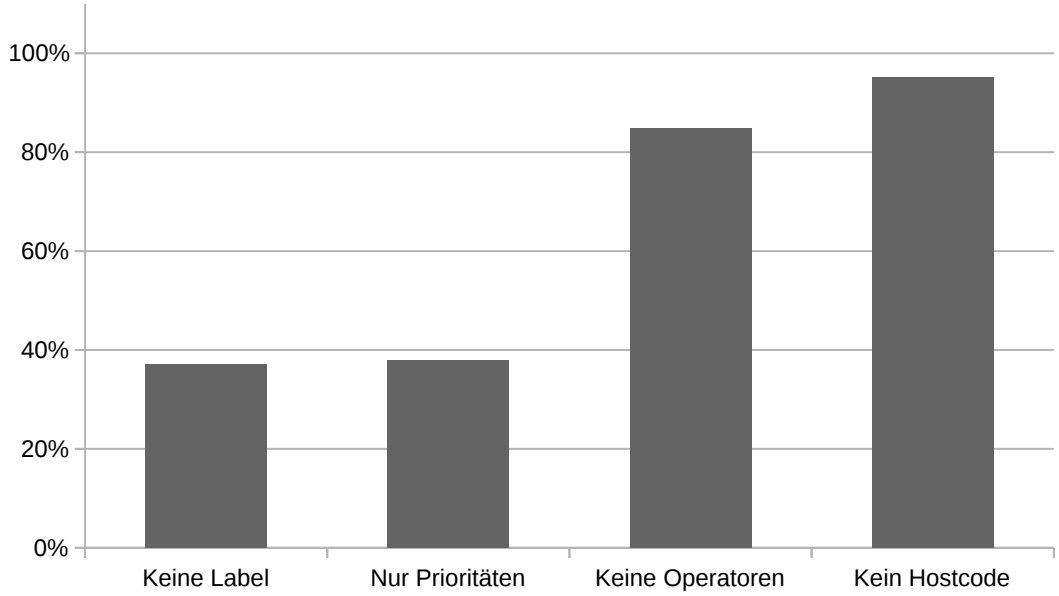

Abbildung 6.7. Prozentuale Veränderung der Kantenlänge für die zielbreitenunabhängigen SCCharts-Strategien.

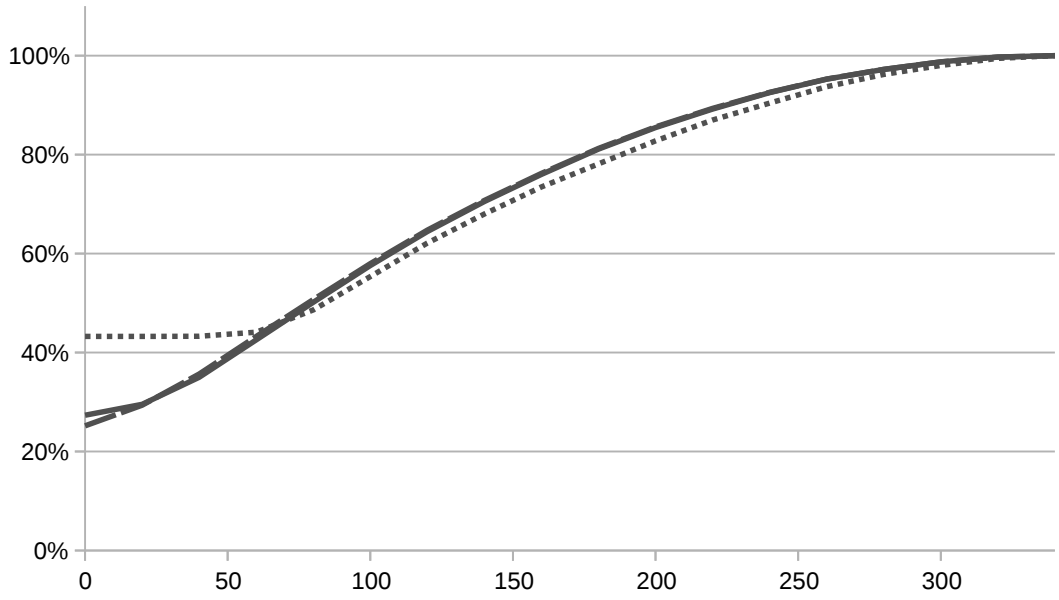

Abbildung 6.8. Prozentuale Veränderung der Graphenbreite nach der Anwendung der Basisstrategien auf Zufallsgraphen. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für syntaktisches Umbrechen und die gepunktete für semantisches Umbrechen.

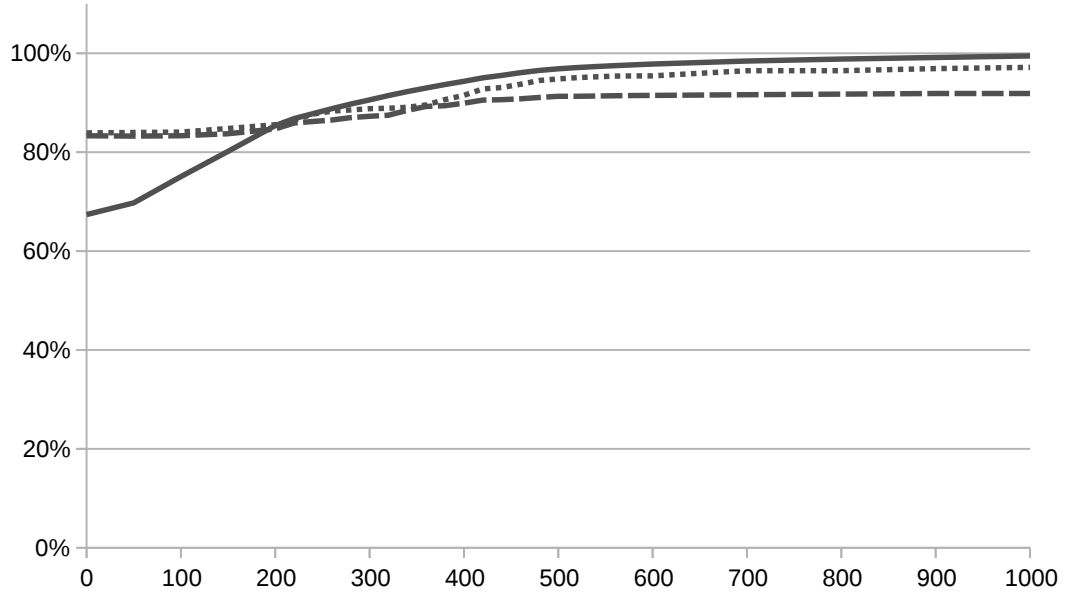

Abbildung 6.9. Prozentuale Veränderung der Graphenbreite für die zielbreitenabhängigen SCCharts-Strategien. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für die Kombination (Semantisches Umbrechen, Entfernen des Hostcodes, Entfernen der Operatoren) und die gepunktete für semantisches Umbrechen.

Breite für die Basisstrategien

Das Ergebnis der Evaluation der Breite für die Basisstrategien ist in Abbildung 6.8 zu sehen. Die Breite verhält sich nahezu identisch zur Veränderung der Labelbreite, siehe zum Vergleich Abbildung 6.1.

Alle Kurven steigen parallel mit dem Anstieg der Zielbreite an und sind dabei nahezu deckungsgleich. Nur semantisches Kürzen für niedrigen Breite entfernt sich von den anderen beiden Kurven, da das längste Wort bereits gefunden wurde.

Der Unterschied zu der Kurve der Labellängen ist, dass die Kurven nach oben verschoben sind. So kann nie auf fast 0% gekürzt werden. Dies liegt an der Größe der Knoten und Mindestlänge der Kanten. Ohne Label ist der Graph also maximal um 75% verkleinert.

Breite für die SCCharts-Strategien

Die Ergebnisse der zeilbreitenabhängigen SCCharts-Strategien ist in Abbildung 6.9 dargestellt. Die Kurven für semantisches Umbrechen und die Kombination sind hier sehr eng beieinander und habe wieder keine besonders große Steigung und durch die Nähe zu der 100%-Marke auch keine besonders große Kürzung abzulesen.

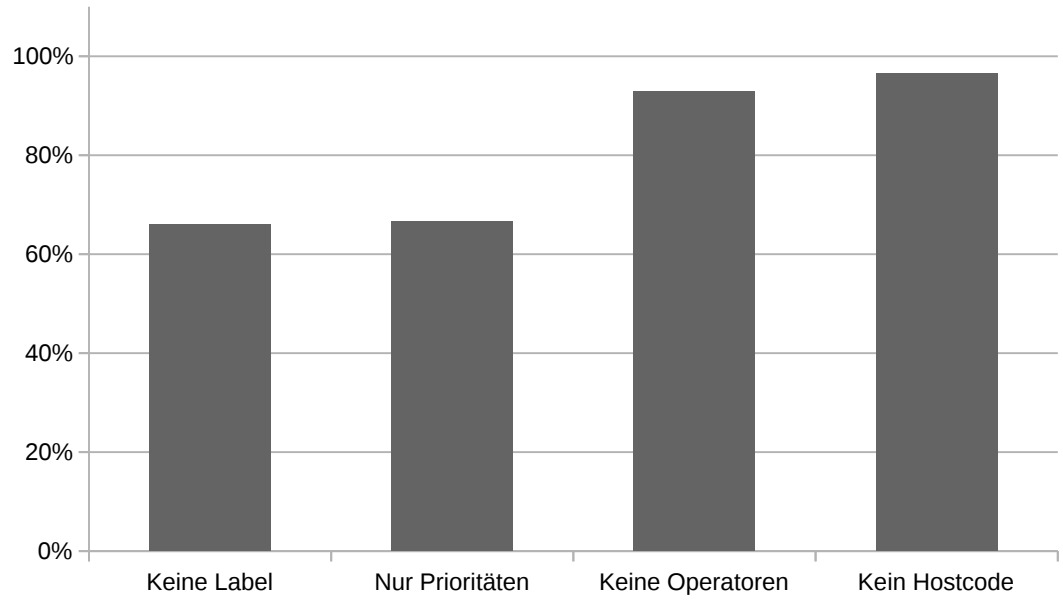

Abbildung 6.10. Prozentuale Veränderung der Graphenbreite für die zielbreitenunabhängigen SCCharts-Strategien.

Für das Kürzen der Breite des Graphen scheinen sich für diese Strategien hauptsächlich niedrigere Breitenwerte anzubieten, bei denen eine Kürzung von etwa 82% erreicht wird.

Insbesondere ist an dieser Graphik abzulesen, dass der Graph ohne Label auf 66% seiner Originalgröße dargestellt werden kann, also maximal auf diesen Wert gekürzt werden kann. Im Vergleich dazu liegt der Wert für die KGraphen bei 25%. Das bedeutet, dass die Kürzungsstrategien nicht viel Spielraum haben, den Graph zu verkleinern und gleichzeitig Informationen zu erhalten. Dies macht Werte, die Label um die 80% kürzen, wieder attraktiver, da diese die Mitte des überhaupt möglichen Wertebereichs darstellen.

Die Abbildung 6.10 zeigt die Veränderung der Breite für die zielbreitenunabhängigen Strategien. Hier ist erneut abzulesen, dass eine Kürzung der Breite bei den SCCharts auf höchstens 66% erfolgen kann. Außerdem wird aufgezeigt, dass die Strategien der Entfernung der Operatoren und des Hostcodes eine effektive Kürzung der Breite von nur um 90% und höher schaffen.

6.5. Höhe des Graphen

Für die Metrik der Höhe soll betrachtet werden, wie sich insbesondere die Umbrüche auf den Graphen auswirken.

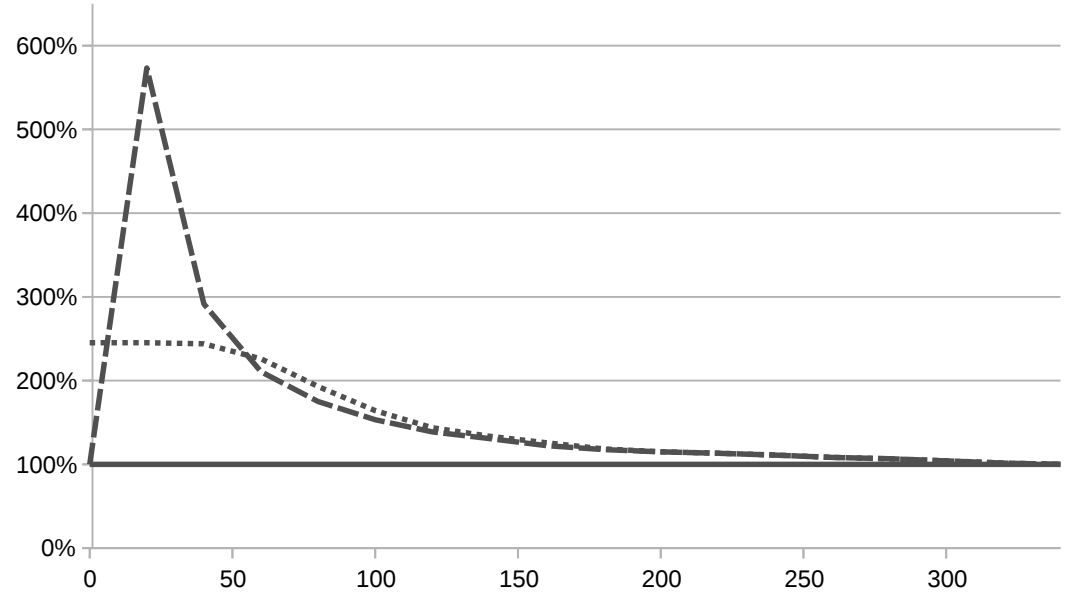

Abbildung 6.11. Prozentuale Veränderung der Graphenhöhe nach der Anwendung der Basisstrategien auf Zufallsgraphen. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für syntaktisches Umbrechen und die gepunktete für semantisches Umbrechen.

Höhe für die Basisstrategien

Die Veränderung der Höhe durch die Kürzungsstrategien ist in Abbildung 6.11 zu sehen. Auch hier wird die prozentuale Veränderung betrachtet. Die Veränderung für das syntaktische Kürzen beträgt, unabhängig von der Zielbreite, 100%. Dies liegt daran, dass beim syntaktischen Kürzen die Höhe der Label nicht verändert wird, da es keine Umbrüche gibt.

Syntaktisch hartes Umbrechen hingegen bewirkt sehr starke Veränderungen in der Höhe, sodass bei Werten um die 20 eine Vergrößerung der Höhe auf fast 600% erreicht werden. Dies geschieht durch die horizontale Streckung der Label, wie bereits anhand von Abbildung 6.5 erläutert wurde. Bei niedrigeren Zielbreiten nimmt die Veränderung wieder ab, da immer weniger Label dargestellt werden. Bei größeren Zielbreiten nimmt die Veränderung mit steigender Breite wieder ab.

Beim semantischen Umbrechen ist bei niedrigen Zielbreiten zu beobachten, dass ebenfalls eine Steigerung der Höhe erfolgt. Die Veränderung steigt dabei auf fast 250%. Die Veränderung der Höhe ist bei niedrigen Zielbreiten konstant. Dies liegt daran, dass nur nach dem breitesten Wort gekürzt wird, die maximale Anzahl an Umbrüchen und somit die maximale Höhe ist erreicht. Mit zunehmender Zielbreite orientiert sich der Algorithmus an der Zielbreite, sodass die Veränderung sich wieder 100% annähert.

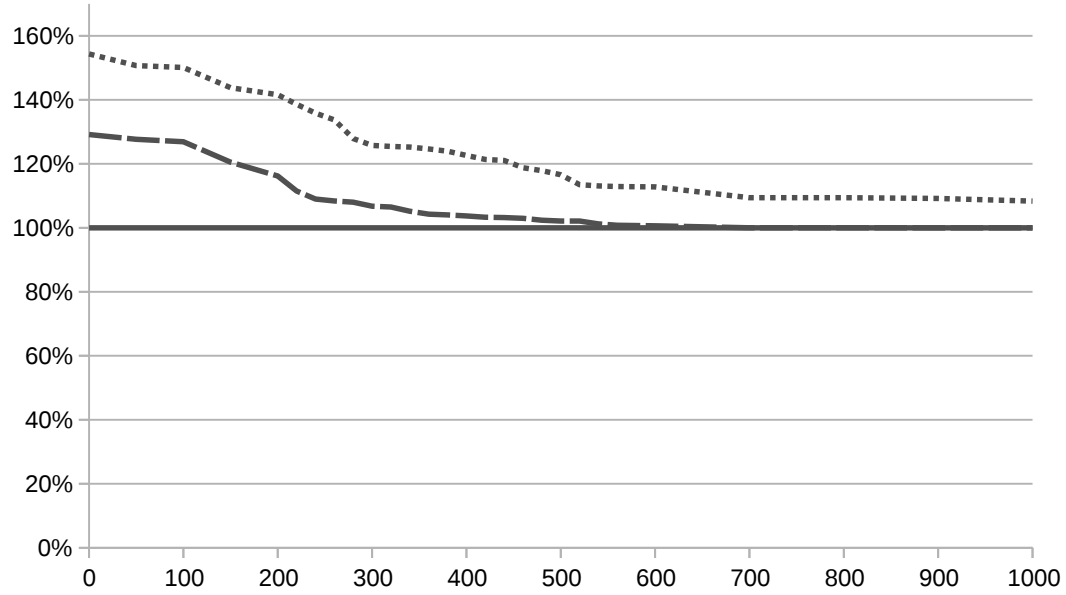

Abbildung 6.12. Prozentuale Veränderung der Graphenhöhe für die zielbreitenabhängigen SCCharts-Strategien. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für die Kombination (Semantisches Umbrechen, Entfernen des Hostcodes, Entfernen der Operatoren) und die gepunktete für semantisches Umbrechen.

Höhe für die SCCharts-Strategien

Abbildung 6.12 zeigt die Veränderung der Höhe durch die zielbreitenabhängigen Strategien. Für das syntaktische Kürzen ist klar abzulesen, dass keine Veränderung in der Höhe geschieht und die Veränderung bei 100% verbleibt. Die Kurven der beiden anderen Strategien haben einen parallelen Verlauf, der über 100% startet und sich 100% annähert. Die Höhe ist bei niedrigen Zielbreiten vergrößert, da dort vermehrt Umbrüche eingefügt werden, während bei größeren Breiten das Label in einer Zeile dargestellt werden kann. Auch hier ist die Kürzung der Kombination deutlich größer als die des semantischen Umbrechens. So wird insbesondere für größere Breiten schneller die 100%-Grenze erreicht. Dies liegt auch daran, dass die Label auf ihre Signale reduziert kürzer sind als das ganzes Label.

Für die zielbreitenunabhängigen Graphen gilt, dass alle Ergebnisse bei 100% liegen. Das liegt darin begründet, dass keine der gegebene Strategien Umbrüche verwendet, sodass die Höhe unverändert bleibt.

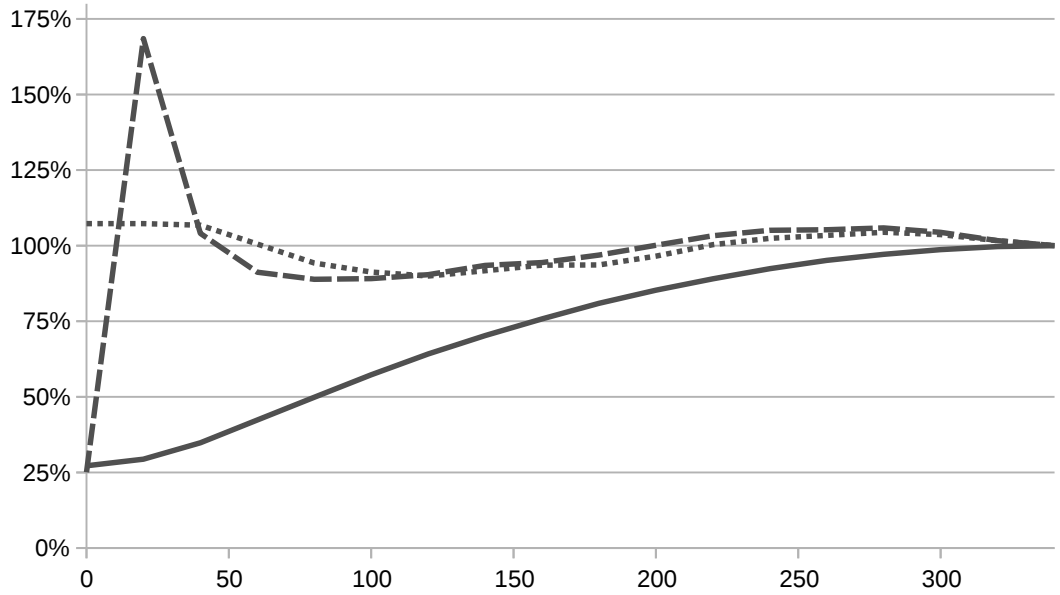

Abbildung 6.13. Prozentuale Veränderung der Fläche des Graphen nach der Anwendung der Basisstrategien auf Zufallsgraphen. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für syntaktisches Umbrechen und die gepunktete für semantisches Umbrechen.

6.6. Fläche des Graphen

Nun soll die Fläche betrachtet werden, um einen Eindruck von der Gesamtveränderung des Graphen zu bekommen.

Fläche für die Basisstrategien

Die Evaluation der Fläche ist in Abbildung 6.13 dargestellt. Die Verkleinerung der Fläche durch das syntaktische Kürzen wird allein durch die Breite bestimmt. Dies liegt daran, dass es keinerlei Höhenveränderung gibt. So ergibt es sich, dass unabhängig von der Zielbreite immer eine Verkleinerung des Graphen stattfindet.

Syntaktisches Umbrechen hingegen kann auch eine Vergrößerung des Graphen herbeiführen. Zwar wurde gezeigt, dass sich die Breite des Graphen immer verkleinert, doch ist der Ausschlag der Veränderung in der Höhe zu stark, sodass das Produkt aus Höhe und Breite für niedrige Zielbreiten zu der Vergrößerung führt. Eine Verkleinerung Fläche des Graphen findet ungefähr zwischen 40 und 200 Breiteneinheiten statt.

Bei semantischem Umbrechen ist die eigentliche Verkleinerung ebenfalls sehr gering und nur im Bereich der Breitenwerte 40 bis 200 gegeben. Der Ausschlag der Vergrößerung im niedrigen Bereich ist wie bei der Höhe geringer als beim syntaktischen Umbrechen.

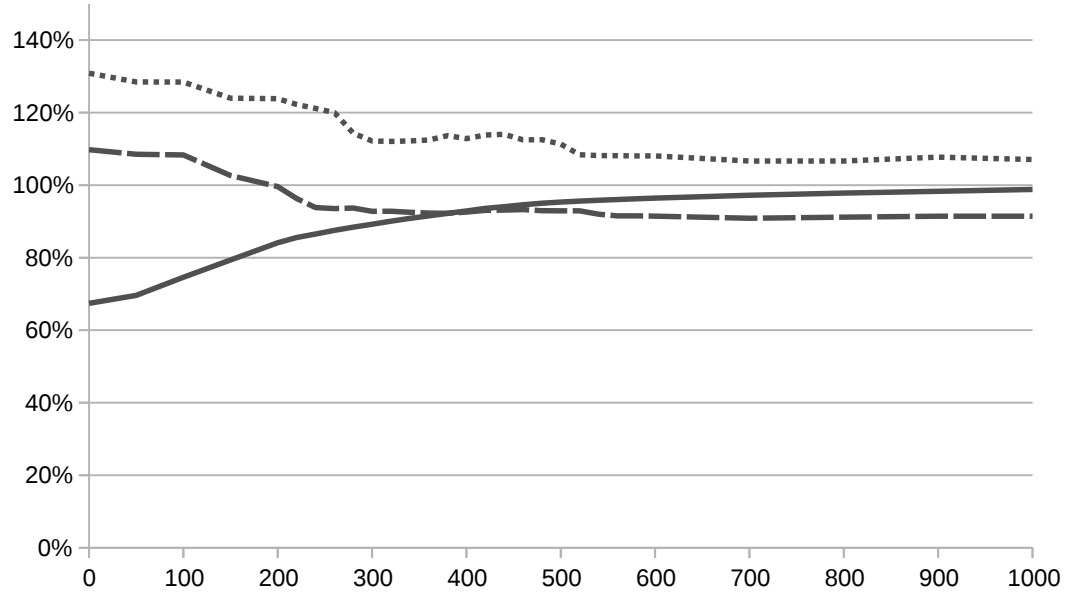

Abbildung 6.14. Prozentuale Veränderung der Fläche des Graphen für die zielbreitenabhängigen SCCharts-Strategien. Die durchgezogene Linie steht für syntaktisches Kürzen, die gestrichelte für die Kombination (Semantisches Umbrechen, Entfernen des Hostcodes, Entfernen der Operatoren) und die gepunktete für semantisches Umbrechen.

Fläche für die SCCharts-Strategien

Das Diagramm für den Effekt der zielbreitenabhängigen Algorithmen auf die Fläche ist in Abbildung 6.14 zu sehen. Auch hier ist das syntaktische Kürzen allein durch die Breite bestimmt, da die Höhe unverändert geblieben ist. So führt syntaktisches Kürzen immer zu einer Verkleinerung des Graphen.

Semantisches Umbrechen führt hingegen zu einer Vergrößerung der Fläche, da die Kürzung in der Breite nicht sehr groß war, während eine gewisse Vergrößerung der Höhe einem guten Ergebnis für die Fläche entgegenwirkt. Allerdings kann auch hier argumentiert werden, dass das Hauptaugenmerk bei dieser Evaluation auf die Breite zu richten ist, da hier ausschließlich in horizontalen Ebenen organisierte Graphen verkleinert wurden. Die Kombination hingegen bietet eine Verkleinerung der Fläche ab einer Breite von etwa 200, also ungefähr ab dem Wert, wo keine Vergrößerung in der Höhe mehr auftreten.

Die Abbildung 6.15 zeigt den Effekt der zielbreitenunabhängigen Strategien auf die Fläche. Wie für das syntaktische Kürzen gilt auch hier, das die Höhe unverändert war, also die Breite die Fläche bestimmt, so dass die Interpretation dieser deckungsgleich ist mit der für die Breite.

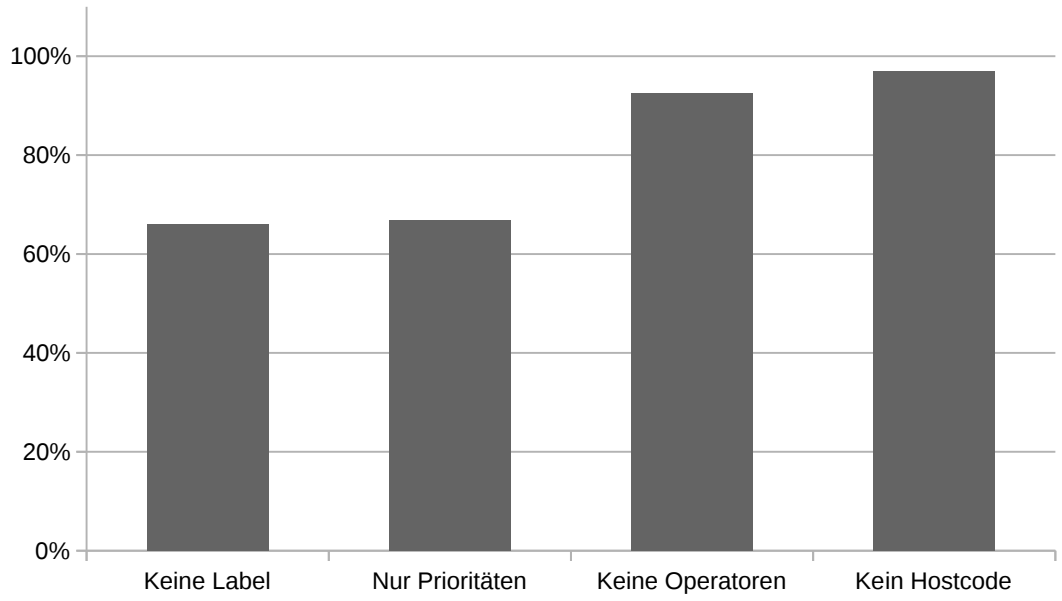

Abbildung 6.15. Prozentuale Veränderung der Fläche des Graphen für die zielbreitenunabhängigen SCCharts-Strategien.

6.7. Gesamtinterpretation

Insgesamt lässt sich für die Basisstrategien sagen, dass sich durch das syntaktische Kürzen die stärkste Verkleinerung der Fläche pro Zielbreite herbeiführen lässt. Bezogen auf die Breite ist der Kürzungsgrad bei allen drei Strategien nahezu äquivalent. Da die Sortierung der Ebene hier ebenfalls horizontal erfolgt, ist das Kürzen in der Breite für den Überblick ergiebiger, da die Knoten näher zusammenrücken. Die Vergrößerung der Höhe ist dabei ein in Kauf zu nehmender Nebeneffekt. Allerdings sollte eine untere Grenze für die Zielbreite gesetzt werden, die verhindert, dass die Label vertikal dargestellt werden.

Abschließend lässt sich für die SCCharts sagen, dass die Evaluation der SCCharts gezeigt hat, dass das Kürzen der Graphen durch das Ändern der Kantenlabel weniger Spielraum hat als bei KGraphen. Daher sind die Effekte der Kürzungsstrategien allgemein geringer.

Die effektivsten Strategien sind hier das Entfernen der Label, das Entfernen der Label bis auf Prioritätsnummern und wieder das syntaktische Kürzen. Bei den informationserhaltenden Strategien bietet die Kombination der Strategien einen guten Mittelwert.

Kapitel 7

Schlussfolgerung

Zum Schluss sollen die vorgestellten Ideen noch einmal zusammengefasst werden. Es wurde herausgestellt, dass Label Management in großen Graphen nötig ist, um einen besseren Überblick über den Graphen zu bekommen.

Für das Label Management wurde sich mit zwei Fragestellungen auseinandergesetzt: *Wie* werden Label gekürzt und *was* für Label kürzt man.

Für das *Wie* wurde einige Basisstrategien vorgestellt, mit denen man alle Label kürzen kann, unabhängig davon, was über sie bekannt ist. Dazu zählt zum Beispiel das syntaktische Kürzen, dass eine Zeichenkette einfach abschneidet.

Darüber hinaus gibt es einige Kürzungsstrategien, die sehr problemspezifisch sind. So ist die Entfernung von Hostcode nur sinnvoll, wenn auch welcher in den Labeln vorkommt. Wie Label gekürzt werden können, ist somit sehr davon abhängig, was für Graphen mit welcher Form von Labeln gegeben sind. Der optimale Kürzungsalgorithmus muss daher für jedes Problem neu bestimmt werden.

In Bezug auf das *Was* wurden Fokus-und-Kontext-Strategien für Label vorgestellt. Dafür wurden verschiedene Möglichkeiten aufgezeigt, wie mit Hilfe des Fokus die Label im Graphen verändert werden können und wie dieser gesetzt werden kann. So kann es einen einfachen Fokus geben, der nur das selektierte Label als Fokus versteht oder aber hierarchischen Fokus, der für selektierte Knoten auch den inneren Graphen als Fokus definiert. Auch das Prinzip verschiedener Detaillevel, das es ermöglicht verschiedene Küzungsstrategien im selben Graphen zu verwenden, wurde genauer erläutert.

Für die Umsetzung der verschiedenen Strategien wurde die Implementierung des Label Managements im Kontext von KLighD und KLay Layered vorgestellt. Es wurde gezeigt, dass unter Verwendung einer abstrakten Oberklasse, die Erweiterung um weitere Strategien sehr einfach ist. Hier muss ausschließlich der Labeltext verändert werden, alle weiteren Berechnungen übernimmt der `AbstractKlighdLabelManager`.

Die Evaluation hat gezeigt, dass syntaktisches Kürzen die meiste Verkleinerung von Labeln, Kanten, sowie Breite, Höhe und Fläche des Graphen bewirkt. Dieses Ergebnis geht für niedrigere Breiten allerdings immer stärker mit einem Informationsverlust einher. Syntaktisch hartes Umbrechen liefert ähnlich gute Werte für Veränderungen, welche die Breite betreffen. Bei niedrigen Breiten allerdings, steigt die Höhe so stark, dass die Fläche des Graphen sogar vergrößert wird. Bei den SCCharts hat die Evaluation gezeigt, das die Graphen im Vergleich zu den KGraphen wesentlich geringer verkleinert werden können.

Die Veränderung durch Kürzen bei zielbreitenunabhängigen Algorithmen fällt konstant und im Mittel geringer aus. Es bleibt zu evaluieren, wie die Verwendung der jeweiligen Strategien dem Benutzer gefallen.

7.1. Offene Probleme

Es besteht die Möglichkeit, weitere Kürzungsstrategien anzubieten, wenn Wörterbücher verwendet werden. Semantisches Kürzen könnten durch die Verwendung eines solchen, das die im Alltag gängige Abkürzungen enthält, verbessert werden. Auch für semantisches Umbrechen könnte ein Wörterbücher lohnenswert sein, die Silbentrennung ermöglicht.

Die Strategie des Umbennens der Label bleibt bisher noch zu implementieren und mit Benutzern zu evaluieren..

Für Fokus und Kontext fehlen Strategien, die den Gedanken des Detaillevels umsetzten. Eine spezielle Form davon wäre eine Strategie, die mit zunehmender Distanz zum Fokus, die Größe der Schrift abnehmen lässt.

Das Erweitern des Label Managements für Kürzungsstrategien ist bereits einfach gehalten worden. Für Fokus-und-Kontext-Actions gilt das allerdings noch nicht, hier könnte ebenfalls eine abstrakte Oberklasse implementiert werden.

Bisher ist die effektivste Kürzungsstrategie im Bezug auf die hier vorgestellten Metriken, dass es keine Label gibt. Daher wäre eine Analyse von Benutzerfreundlichkeit und Lesbarkeit angebracht. Diese könnte außerdem zeigen, welche Form der Bedienung und welche konkreten Kürzungsstrategien für Benutzer am lohnenswertesten sind. Insbesondere das Kürzen mit nicht zielbreitenabhängigen Algorithmen sind am besten durch Benutzer zu evaluieren.

Auch ausstehend ist eine Analyse der Performance.

Da sich hier bisher nur auf Kanten und Portlabel beschränkt wurde, ist ein mögliches Thema das Kürzen von Knotenlabel. Dies ist insbesondere für SCCharts interessant, da dort die Kürzungsstrategien weniger effektiv waren. Dort sind weniger die Namen der Konten oder die Signaldeklarationen von Interesse, als die sogenannten Entry Actions. Diese entsprechen Transitionen, werden allerdings, um den Graphen kompakter zu machen, im Knoten gelistet. Wenn diese besonders groß ausfallen, wird der Graph durch breitere Knoten wieder vergrößert.

Abkürzungen

KIELER Kiel Integrated Environment for Layout Eclipse RichClient

KLighD KIELER Lightweight Diagrams

KLay Layered Kieler Layout Layered

SCCharts Sequentially Constructive Charts

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Literaturverzeichnis

[EJL+03] Eker, Johan, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs und Yuhong Xiong: *Taming Heterogeneity—The Ptolemy Approach*. Proceedings of the IEEE, 91(1):127–144, Jan 2003, ISSN 0018-9219.

[Fuh11] Fuhrmann, Hauke: *On the Pragmatics of Graphical Modeling*. Dissertation, Christian-Albrechts-Universität zu Kiel, Faculty of Engineering, Kiel, 2011.

[Fur86] Furnas, Gerge W.: *Generalized Fisheye Views*. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, Seiten 16–23. ACM, 1986, ISBN 0-89791-180-6.

[MJ03] Musial, Benjamin und Timothy Jacobs: *Application of focus + context to UML*. In: *APVis '03: Proceedings of the Asia-Pacific symposium on Information visualisation*, Seiten 75–80, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc., ISBN 1-920682-03-1.

[SB92] Sarkar, Manojit und Marc H. Brown: *Graphical Fisheye Views of Graphs*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Seiten 83–91. ACM, 1992, ISBN 0-89791-513-5.

[vHDM+14] Hanxleden, Reinhard von, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer und Owen O'Brien: *SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications*. In: *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*, Edinburgh, UK, Juni 2014. ACM. Long version: Technical Report 1311, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, December 2013, ISSN 2192-6274.

[vHF10] Hanxleden, Reinhard von und Hauke Fuhrmann: *Taming Graphical Modeling*. Presentation at the 17th International Open Workshop on Synchronous Programming (SYNCHRON'10), Frejus, France, Dezember 2010.