CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL

Diplomarbeit

Viewmanagement für visuelles Modellieren

Nils Beckel

16. Oktober 2009

Institut für Informatik
Lehrstuhl für Echtzeitsysteme und eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:
Dipl.-Inf. Hauke Fuhrmann

Eidesstattliche Erklärung

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

Kiel, den _____

Inhaltsverzeichnis

1 Einleitung und Problemstellung

> »The future of digital systems is complexity, and complexity is the worst enemy of security.[1]« *Bruce Schneier, 2000*

Das Hauptziel in der Informatik ist die Entwicklung von Programmen. Alle Aktivitäten der Forschung, der Ausbildung und Schulung sollen den Informatiker[2] befähigen, Programme jedweder Art indirekt oder direkt neu zu entwickeln oder weiterzuentwickeln.

Dabei hat sich in den letzten Jahrzehnten im Zuge der enormen Entwicklung in den Bereichen der Elektronik und Software der Entwicklungsprozess stark gewandelt. Höhere Rechenleistung von Prozessoren, größere Möglichkeiten von Ein- und Ausgabegeräten, Sensoren, Interfaces usw. ermöglichen immer komplexere Programme mit stark angewachsenem Funktionsumfang.

Durch fortschreitende Miniaturisierung gewinnen eingebettete Systeme rasant an Bedeutung. So werden Aufgaben, die früher von einfachsten elektronischen Schaltkreisen übernommen wurden – oder auch einfach nicht realisierbar waren –, mehr und mehr durch kompliziertere, chipbasierte Systeme ersetzt. Dies ermöglicht Verbesserungen in vielen Bereichen, so zum Beispiel im Bereich der Betriebssicherheit durch die Möglichkeit einer Fehlerbehandlung auf unterster Systemebene. Für den Benutzer bestehen die Verbesserungen in erhöhtem Nutzen und Komfort durch interaktive, geführte Bedienung; durch differenzierte Systemregelung sind höhere Effizienz und Effektivität erreichbar.

[1] http://www.schneier.com/crypto-gram-0003.html

[2] Aus Gründen der besseren Lesbarkeit wurde für die vorliegende Arbeit bei Personenbezeichnungen die männliche Form gewählt. Nichtsdestotrotz adressiert der Autor damit ausdrücklich Angehörige beider Geschlechter.

Diese Entwicklung hat jedoch auch Schattenseiten. Waren früher viele Systeme rein elektrisch oder mechanisch, ist heutzutage vielfach ein eingebettetes System im Spiel, um diese Aufgaben zu übernehmen. Die Entwicklung von Autos, Haushaltsgeräten oder Maschinen vieler Art erfordert deshalb mittlerweile tief gehende Informatikkenntnisse.

Im selben Maße nahm die Komplexität von größeren Softwareprojekten zu. Während das primäre Avionic-System des Space Shuttles im Jahre 1993 etwa 420.000 Zeilen Code umfasste (*Lines of Code*, LoC)[3] und das Betriebssystem Microsoft Windows 3.1, welches im Jahre 1992 veröffentlich wurde, etwa 3 Mio. LoC, war Windows NT 4.0 (1996) schon 16,5 Mio. LoC groß, Windows Server 2003 (2003) 50 Mio. LoC[4]. Bei Open Source Betriebssystemen war der Zuwachs noch signifikanter, so wuchs Debian von Version 2.2 (2000) zu Version 4.0 (2007)[5] von 55 Mio. auf 283 Mio LoC.[6].

1.1 Problembeschreibung

Um die wachsende Komplexität der Softwareprojekte beherrschbar zu halten oder überhaupt erst zu ermöglichen, hat sich im selben Maße der Entwicklungsprozess gewandelt. Strukturbasiertes und objektorientiertes Programmieren wurde selbstverständlich, ebenso die professionelle und industrielle Handhabung eines Software-Projekts durch Spezialisierung der Entwickler, durch Projektmanagement, die Einführung von Programmier-*Stylebooks* und formalen Prozessen.

Ein sehr wichtiger Aspekt, der in den letzten Jahren mehr und mehr an Bedeutung gewinnen konnte, ist die Modellgetriebene Softwareentwicklung (*Model-Driven Software Developement – MDSD*)[7]. Diese Technik entlastet den Entwickler dahingehend, dass dieser ein abstraktes Modell seiner geplanten Software erstellt, aus dem entsprechende Codegeneratoren automatisch einen lauffähigen

[3] http://www.nap.edu/openbook.php?isbn=0309053447&page=9
[4] http://www.schneier.com/crypto-gram-0003.html
[5] https://secure.wikimedia.org/wikipedia/de/w/index.php?title=Debian&oldid=63790034
[6] https://secure.wikimedia.org/wikipedia/en/w/index.php?title=Source_lines_of_code&oldid=310286567
[7] http://en.wikipedia.org/w/index.php?title=Model-driven_engineering&oldid=298827922

Code erzeugen.

Das MDSD basiert im wesentlichen auf textuellen und grafischen Modellen, wobei letztere durch ihre besonders einfache und anschauliche Handhabung bestechen.

Jedoch ergeben sich bei grafischer beziehungsweise visueller Modellierung andere, neue Problemstellungen. Sind einfache und kompakte Modelle noch mühelos beherrschbar, steigt bei großen und komplexen Modellen die Belastung des Entwicklers mit fachfremden Aufgaben stark an. So muss sich der Entwickler mehr und mehr auf die manuelle (Re-)Strukturierung bei Bearbeitung und Erweiterung des Modells konzentrieren, das Einfügen eines einzelnen, simplen Objekts in das Modell kann zu einer aufwändigen Aufgabe werden, bei der umfangreiche Modifikationen vorgenommen werden müssen, schon um schlicht Platz für dieses neue Objekt zu schaffen.

Dabei muss die Struktur übersichtlich bleiben und natürlich sind Fehler unbedingt zu vermeiden. Dies ist bei der Vielzahl der eventuell nötigen Verschiebungen zahlreicher Objekte schwer und außerordentlich zeitaufwändig.

Um den Nutzen der visuellen MDSD nicht zunichte zu machen, sind maschinengestützte Hilfen dringend nötig. Insbesondere das automatische *Layout* ist hierbei von großem Nutzen, da es dem Entwickler eben diese fehleranfälligen und im Prinzip fachfremden Arbeiten abnimmt und gleichzeitig die Lesbarkeit des Modells verbessern kann. Das Thema Layouting wurde bereits in einigen Arbeiten behandelt, so zum Beispiel im Rahmen des KIEL-Projektes[8] von Tobias Kloss [18], Steffen Prochnow [27] und Jonas Völcker [41].

Das automatische Layouting befreit den Entwickler zwar von einer großen Last, ist jedoch noch nicht ausreichend, um ein wirklich effizientes Arbeiten zu ermöglichen.

Abgesehen vom aufwändigen Restrukturieren von Diagrammen im Entwicklungsprozess liegt das Hauptproblem, welches gelöst werden soll, an anderer Stelle.

So können *Statecharts* zur Modellierung einfacher Systeme den Entwickler be-

[8] https://www.informatik.uni-kiel.de/rtsys/kiel/

reits vor große Probleme im Bereich der Übersichtlichkeit stellen. Statecharts – oder Zustandsübergangsdiagramme – sind eine grafische Darstellung endlicher Automaten, die um die Konzepte der Hierarchie und Komposition erweitert wurden. Die Darstellung und Notation wurde maßgeblich von David Harel geprägt [12]. Sein »Wristwatch«-Beispiel demonstriert eindrucksvoll die Dimension der genannten Probleme in Bezug auf die Übersichtlichkeit. In diesem Beispiel wird ein relativ simples Modell einer Armbanduhr betrachtet (siehe Abbildung 1.1). Diese Uhr verfügt über folgende Funktionen:

- vier Knöpfe,
- ein großes und ein kleines Display,
- einen »Beeper«,
- Anzeige der Uhrzeit im am/pm-Format und 24h-Format,
- eine Stoppuhr-Funktion mit der Möglichkeit zur Aufzeichnung von Rundenzeiten,
- eine Datumsfunktion,
- zwei Wecker,
- Beleuchtung,
- ein Stundensignal,
- Warnung vor niedrigem Batteriestand.

Obwohl dies offensichtlich ein sehr simples System ist und man sich mühelos – etwa im Bereich der Maschinen- und Anlagensteuerung, der Luftfahrt oder generell im Bereich eingebetteter Echtzeitsysteme – weitaus kompliziertere, umfassendere und in der Hierarchie tiefer gehende Systeme mit einer um mehrere Größenordnungen umfangreicheren Funktionalität vorstellen kann, so stellt die grafische Modellierung des »Wristwatch«-Beispiels bereits ein Problem dar.

In den Abbildungen 1.2 und 1.3 wird deutlich, dass lange Label der Zustände in Verbindung mit zahlreichen hierarchischen Zuständen einen Statechart von gewaltigen Ausmaßen formen. Der Entwickler hat hier nur die Wahl zwischen einer großen Übersicht und einer Detailansicht.

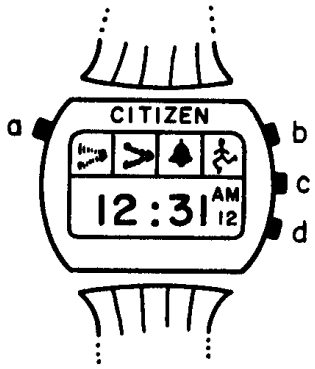

Fig. 7.

Abbildung 1.1: *Darstellung der modellierten Armbanduhr nach D. Harel*

QUELLE: D. HAREL [12]

In der Übersicht sind aber im Prinzip keine Details zu erkennen, eine Struktur ist nicht ersichtlich, irgendwelche Rückschlüsse auf Funktionalität können nicht gezogen werden. Diese »Übersicht« ist also nicht im geringsten »übersichtlich« und somit nutzlos.

Zoomt man jedoch auf einen Teil des Modells, so werden zwar Details wie kleine Substates, Label, Transitionen etc. sichtbar, es ist aber sehr schwer, häufig sogar unmöglich, in dieser Detailansicht die Funktionalität, die das Modell repräsentiert, nachzuvollziehen.

Die zahlreichen hierarchischen Zustände sorgen für eine Vielzahl von Trennlinien, sodass der Betrachter größte Mühe hat, nachzuvollziehen, in welcher Hierarchieebene er sich befindet. Auch Charles André, der mit seiner Ausarbeitung »Semantics of Safe State Machines« [1] den Grundstein für die momentan verbreitete Darstellung von *Safe State Machines* (SSM) gelegt hat, vermerkt, dass die Darstellung einer sehr tiefen Hierarchie in einer SSM die Lesbarkeit und Verständlichkeit behindert. Die in der Regel stark begrenzte zur Verfügung stehende Bildschirmfläche – es sollen schließlich auch Entwicklungswerkzeuge in einer Palette, Menuleisten, andere entwicklungsbezogene Fenster und Ansichten sichtbar sein – verschärft das Problem.

Neben einer technischen und kostenbezogenen Beschränkung der Bildschirmgröße ergeben sich hier auch menschliche Beschränkungen. Der Mensch ist nur

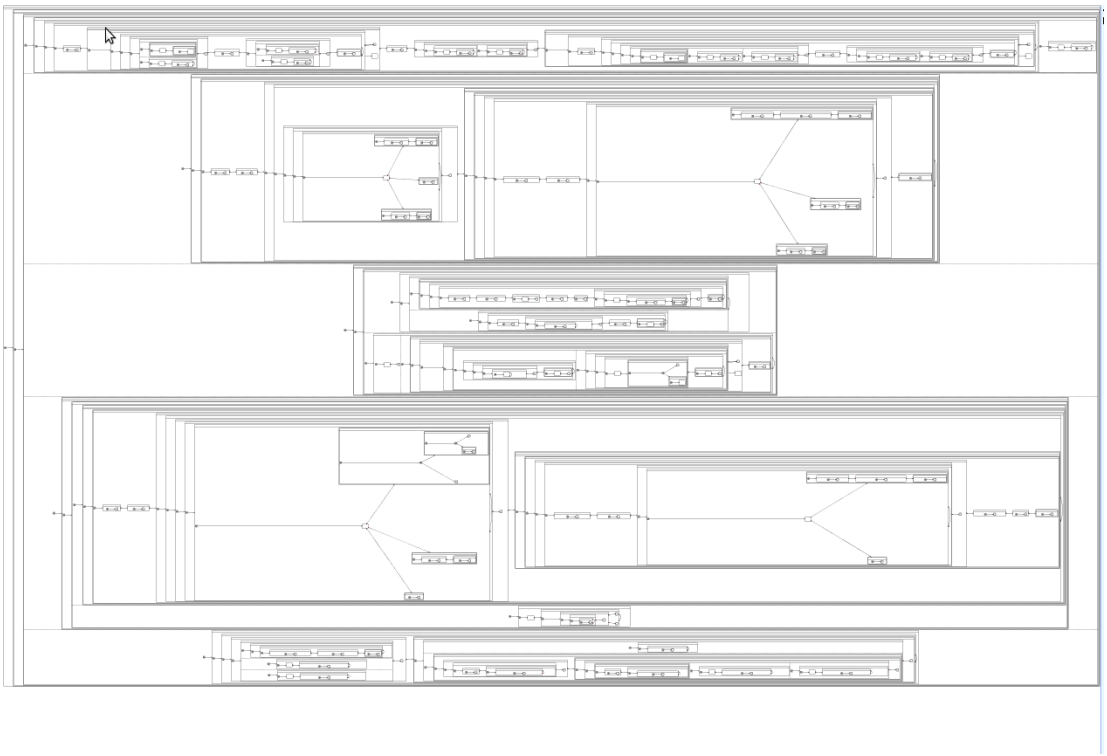

Abbildung 1.2: *Übersicht über das Wristwatch-Beispiel nach D. Harel, dargestellt als Statechart in KIEL*

in der Lage, eine gewisse Größe eines Bildschirms zu erfassen und zu überblicken, die Anzahl der Details, die er verarbeiten kann, ist somit stark limitiert [14]. Auch Tory und Möller sehen die maximal mögliche Bildschirmgröße stark begrenzt und bezeichnen dies als das *screen real-estate problem*. Sie adressieren aber noch bestehenden Forschungsbedarf, welches die optimale Bildschirmgröße sei und welchen Einfluss die Veränderung derselben auf die Verarbeitung von Informationen hat [40]. Umfassende Untersuchungen zu diesem Problem der Repräsentation großer Datenmengen existieren bereits in großer Zahl und werden im Kapitel 3 »Visualisierung von Modellen – Stand der Entwicklung« diskutiert.

Bei der Betrachtung dieser Problembeschreibung muss man sich immer wieder vor Augen halten, dass es sich bei dem »Wristwatch«-Beispiel um ein sehr simples System handelt, welches eher selten grafisch modelliert werden wird. Systeme, die grafische Modellierung sinnvoll machen, sind in aller Regel um mehrere Größenordnungen komplexer, das Problem verschärft sich hierbei exponentiell. Ein weiterer Aspekt ist die oftmals sicherheitskritische Rolle der modellierten

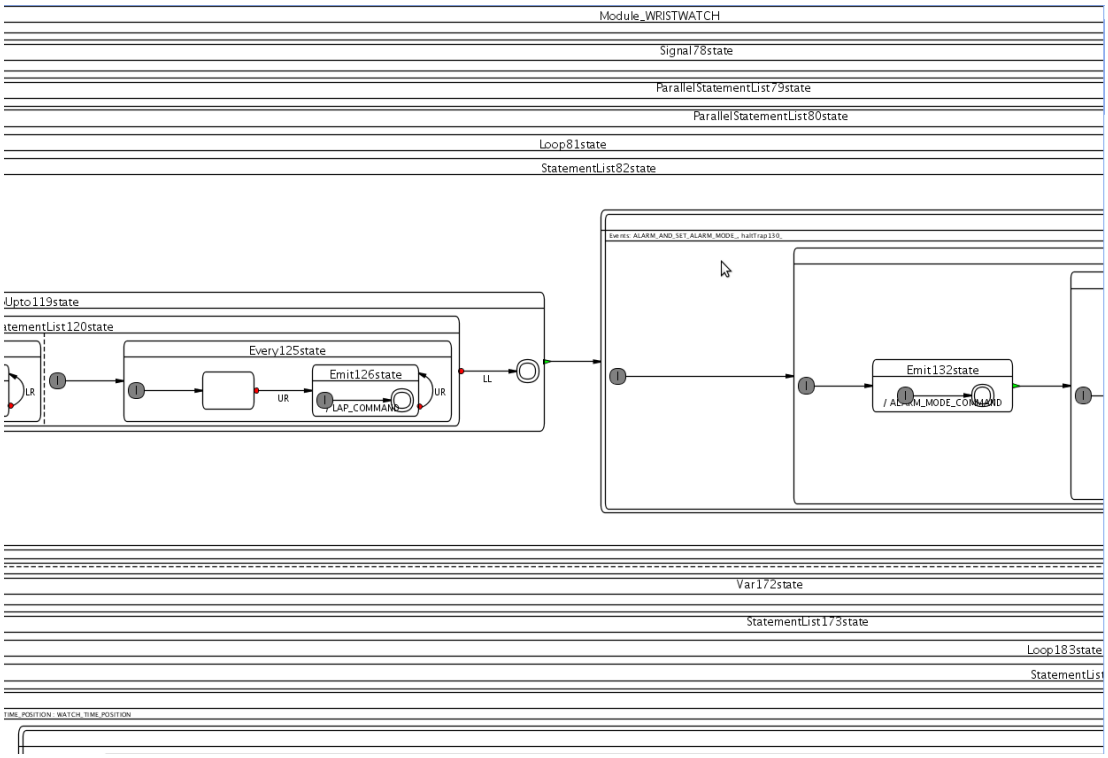

Abbildung 1.3: *Detailansicht des Wristwatch-Beispiels nach D. Harel, dargestellt als Statechart in KIEL*

Systeme, sodass mögliche Fehlerquellen – wie etwa Unübersichtlichkeit des Modells – dringend vermieden werden sollten. Eine Lösung für dieses Problem ist dringend erforderlich, um die eigentlich sehr effektive und produktive Technik der grafischen Modellierung nutzbar zu machen und nutzbar zu halten.

Fuhrmann und von Hanxleden führen in ihrem Paper »On the Pragmatics of Model-Based Design« [11] an, dass die Produktivität der modellbasierten Entwicklung noch wesentlich verbessert werden kann. Sie zitieren das Konzept der *mentalen Karte*, welches besagt, dass jeder Entwickler oder auch nur Betrachter eines visuellen Modells eine Karte des Modells im Kopf hat, mithilfe derer er weiß, welchen Teil des Modells er gerade betrachtet und wo gerade außerhalb des Bildschirmausschnitts befindliche Teile zuzuordnen sind.

Ohne diese mentale Karte müsste der Entwickler ständig nach Objekten suchen und müsste sich ständig neu orientieren. Die mentale Karte ist somit ein unverzichtbarer Bestandteil einer effizienten Entwicklungstätigkeit. Bei zunehmender Komplexität und Größe des Modells ist es jedoch immer schwerer und anstren-

gender, diese Karte aufrecht zu erhalten. Des Weiteren wird die Karte zunehmend ungenauer und fehlerbehafteter, der Entwickler ist wieder gezwungen, Objekte in mühevoller und frustrierender Kleinarbeit zu suchen. Tory und Möller nennen dies *kognitiven Overhead* [40].

Es gilt also, die mentale Karte des Entwicklers aufrecht zu erhalten und Techniken einzuführen, die bei großen und komplexen Modellen unterstützend wirken. Das Konzept der mentalen Karte ist bereits gründlich erforscht und diskutiert, es wird im Kapitel ebenfalls 3 »Visualisierung von Modellen – Stand der Entwicklung« weiter vertieft.

1.2 Zielsetzung der Arbeit

Die vorliegende Arbeit wird das KIELER-Projekt um den Bestandteil »Viewmanagement« ergänzen. Es wird untersucht werden, welche Methoden und Techniken die Darstellungen von Diagrammen so verbessern können, dass das Arbeiten mit Statecharts oder ähnlichen Darstellungen auf Diagrammbasis, das bisher durch die in Abschnitt 1.1 beschriebenen Probleme behindert wird, produktiver und effizienter erfolgen kann.

Es werden Plugins für Eclipse entwickelt, welche die wichtigsten Techniken dieser Verbesserungen realisieren und sie bei der Arbeit mit dem KIELER-Tool nutzbar machen. Diese Plugins stellen eine Experimentierbasis dar, deren Implementierung im Kapitel 5 beschrieben wird.

Die Experimentierbasis verwaltet und koordiniert die realisierten Techniken und Methoden und ermöglicht dem Benutzer einen einfachen und modularen Einsatz dieser Komponenten. Ziel ist es, dass der Benutzer auf einfache Weise empirisch herausfinden kann, welche Ausprägung der realisierten Techniken und Methoden bestmöglich zur Verbesserung seiner Produktivität beiträgt.

Weiterhin soll diese Arbeit die Basis legen für eine wissenschaftlich fundierte Erforschung der Produktivitätssteigerung der Entwicklertätigkeit beim Einsatz von Viewmanagement für visuelles Modellieren. Hierzu sei auf Kapitel 7 »Evaluation der Lösung« und Kapitel 8 »Abschluss« verwiesen.

1.3 Übersicht über die Arbeit

Kapitel 1 stellt die Problematik bei der Arbeit mit visuellen Modellen dar und die daraus resultierende Zielsetzung für die vorliegende Arbeit – die Erstellung eines Viewmanagements.

Das Kapitel 2 stellt das Umfeld, in dem das Viewmanagement entwickelt werden soll, vor. In Kapitel 3 wird der Stand der Entwicklung auf den relevanten Forschungsgebieten zusammengefasst und relevante Techniken, Methoden und Ideen herausgearbeitet.

Im folgenden Kapitel 4 wird anhand der so gewonnenen Erkenntnisse eine Lösungsstruktur skizziert.

Das Kapitel 5 beschreibt die detaillierte technische Umsetzung der Lösung in KIELER und adressiert besondere Problemstellungen und deren spezielle Lösungen.

In Kapitel 6 wird ein Vergleich zwischen dem realisierten Viewmanagement in KIELER und anderen, verwandten Arbeiten gezogen, Vor- und Nachteile werden herausgearbeitet.

Kapitel 7 umfasst eine bewertende Analyse der realisierten Methoden und Techniken und zeigt ihren Beitrag zur Lösung des Problems auf.

Schließlich erfolgt in Kapitel 8 eine Beschreibung möglicher zukünftiger Arbeiten sowie eine Zusammenfassung und ein Fazit.

2 Beschreibung der Umgebung

Am Lehrstuhl für Echtzeitsysteme und eingebettete Systeme der Christian-Albrechts-Universität zu Kiel wurde zur modellbasierten Entwicklung das Thema Statecharts schon länger und mit Nachdruck verfolgt.

Im Mai 2008 wurde das KIEL-Projekt (Kiel Integrated Environment for Layout) zum Abschluss gebracht. Ziel des Projekts war die Entwicklung eines Modellierungswerkzeugs für komplexe reaktive Systeme welches der Erforschung verschiedener Paradigmen über die Bearbeitung, Betrachtung und Simulation von Statecharts diente [27]. In KIEL wurden zahlreiche nutzbringende Techniken für das Arbeiten mit Statecharts eingeführt, so zum Beispiel

- automatisches Layouten von Statecharts mit GraphViz [18],
- Importieren und Exportieren von und zu anderen Modellierungswerkzeugen wie Esterel Studio, Stateflow usw. [30],
- strukturbasiertes Editieren [29][43],
- Erzeugen von grafischen Modellen aus textuellem Code [32],
- Simulieren von Statecharts [24] und
- automatische Robustheitsanalysen [31][33][5].

2.1 KIELER

Das KIEL Projekt wurde abgelöst durch seinen Nachfolger KIELER (Kiel Integrated Environment for Layout for the Eclipse Rich Client Platform)[1]. Es sollte

[1] https://www.informatik.uni-kiel.de/rtsys/kieler/

abweichend von der Zielsetzung des KIEL-Projekts nicht nur Statecharts bearbeiten können, sondern eine Vielzahl von Modellierungssprachen (zum Beispiel auch Datenflusssprachen). Weiterhin sollte sowohl für die Entwicklung als auch für die Integration die *Eclipse Rich Client Platform* benutzt werden. Die Teilprojekte von KIELER werden später in diesem Kapitel vorgestellt.

2.2 Eclipse

Eclipse ist ein ursprünglich von IBM entwickeltes Programmierwerkzeug, welches sich zunächst auf die Entwicklung von Java beschränkte. Es trat die Nachfolge von IBM Visual Age for Java 4.0 an[2]. Mittlerweile wurde die Entwicklung in die Non-Profit-Organisation Eclipse Foundation Inc. ausgegliedert, das Programm wird unter der offenen Lizenz EPL (Eclipse Public License)[3] veröffentlicht.

Obwohl Eclipse selbst noch in Java geschrieben ist, unterstützt es mittlerweile eine Vielzahl weiterer Programmiersprachen wie C, C++, Python und viele mehr. Grundlegendes Merkmal von Eclipse ist die Plugin-basierte Architektur, bei der Eclipse selbst nur den Kern bildet, während durch das Laden von Plugins weiterführende Funktionalitäten eingebunden werden. Auch die Unterstützung der zahlreichen Programmiersprachen wird so realisiert. Die Rich Client Platform in Eclipse ermöglicht es dem Entwickler, eigenständige Eclipse-Anwendungen zu schreiben.

Seit Version 3.0 basiert Eclipse auf einem OSGi-Framework namens *Equinox*. Die OSGi (ehemals Open Services Gateway initiative) bietet mit diesem Framework eine offene, modulare und frei skalierbare Plattform, die Java-basiert ist. Die Plattform ist Service-orientiert, die Services sind ähnlich wie Java-Schnittstellen zu verstehen[4].

Von fundamentaler Bedeutung für das KIELER-Projekt sind drei Entwicklungswerkzeuge aus dem Bereich der modellbasierten Entwicklung, nämlich das

[2] http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F
[3] http://www.eclipse.org/legal/epl-v10.html
[4] http://www.osgi.org/About/Technology#Framework

Eclipse Modeling Framework (EMF), das *Graphical Editing Framework* (GEF) und das *Graphical Modeling Framework* (GMF). Diese Werkzeuge wurden sowohl bei der Entwicklung von KIELER und seiner Teilprojekte eingesetzt als auch zur Entwicklung von grafischen Modellen in KIELER benutzt.

2.3 Eclipse Modeling Framework (EMF)

Das Eclipse Modeling Framework bietet – wie der bereits Name suggeriert – ein auf Eclipse basierendes Modellierungs-Framework mit der Fähigkeit, aus einer Modellierung eines Programms direkt Code zu generieren. Ein Modell in diesem Zusammenhang ist eine abstrakte, vereinfachte und systematisierte Beschreibung eines Programms und seiner Funktionalitäten.

Die Möglichkeit der Codegenerierung verspricht eine Reduzierung des anfänglichen Aufwands bei der Codeerstellung sowie der Fehleranfälligkeit durch die automatische Erstellung häufig wiederkehrender Codeelemente und die einfache Modifikation eines vorhandenen Projekts durch Bearbeiten des Modells und nachfolgende Regenerierung des Codes auf dem veränderten Modell. Von dem Modell abweichende spezialisierte Veränderungen des erzeugten Codes sind dennoch möglich und werden bei erneuter Erzeugung aus dem Modell nicht überschrieben.

Die benutzen Modelle können in annotiertem Java, UML, XML oder anderen, in EMF eingebundenen Modellierungssprachen beschrieben und dann importiert werden. Das Modell wird intern in XMI (XML Metadata Interchange) beschrieben.

EMF war ursprünglich eine Implementierung der *Meta Object Facility* (MOF) der *Object Management Group* (OMG), wurde jedoch weiterentwickelt und auf die Benutzung mit Java zugeschnitten. Das EMF Core Meta Model wird *Ecore* genannt. Es ist dem Essential MOF (EMOF) sehr ähnlich, lediglich kleinere Unterschiede existieren [9]. Eine Übertragung zwischen EMOF und EMF ist daher einfach möglich.

Des Weiteren kann eine Ecore-Datei einfach in ein Ecore-Diagramm (siehe Ab-

bildung 2.1) umgewandelt und somit das Modell visuell und anschaulich dargestellt werden. Das Editieren des Modells ist in dieser Ansicht einfach durch Drag&Drop aus der Palette möglich. Es erleichtert den Umgang mit einem Modell im XMI-Format enorm, macht Zusammenhänge sofort ersichtlich und senkt somit nicht nur die Belastung des Entwicklers, sondern auch die Fehlerquote während des Entwicklungsprozesses. Code-Reviews, wie sie am Lehrstuhl üblich sind, wären ohne diese modellhafte Darstellung nicht oder nur sehr schwer möglich.

Ein weiteres, sehr wichtiges Feature von EMF ist die Validierung. Es lassen sich Einschränkungen – sogenannte Constraints – definieren, die auf Erfüllung überprüft werden. Diese Einschränkungen können entweder selbst und völlig frei definiert werden, oder aber es werden vorgefertigte Einschränkungen für Sprachen – etwa Java – eingelesen und angewandt. Weiterhin lassen sich die Einschränkungen an den Kontext der jeweils betroffenen Applikation anknüpfen und somit zu validierende Objekte und Bedingungen direkt verbinden.

EMF *Compare* ist eine Funktionalität von EMF, welche den Vergleich zweier Modelle und die Darstellung ihrer Unterschiede ermöglich. Schipper setzt diese Fähigkeit als Basis für seine Implementierung ein [34][35]. Dabei wird jede Art von Metamodell unterstützt. Der Vergleichsprozess teilt sich in zwei Abschnitte: Im ersten Schritt wird ein übereinstimmendes Modell erzeugt, im zweiten Schritt wird dieses so entstandene Modell untersucht und die vorhandenen Unterschiede festgestellt. Diese Unterschiede können wiederum als Modell dargestellt werden.

2.4 Graphical Editing Framework (GEF)

Das *Eclipse Graphical Editing Framework* ist ein Plugin, welches es ermöglicht, beliebige Modelle grafisch darzustellen und zu editieren. Weiterhin werden Interaktionen mit der Maus, der Tastatur oder der Eclipse Workbench unterstützt. GEF stellt die Verbindung zwischen *Modell* und *View* – also der tatsächlichen grafischen Darstellung – dar (siehe Abbildung 2.2). Es stellt außerdem *Event Handler* zur Verfügung, die Interaktionen zwischen Modell und Benutzer in Re-

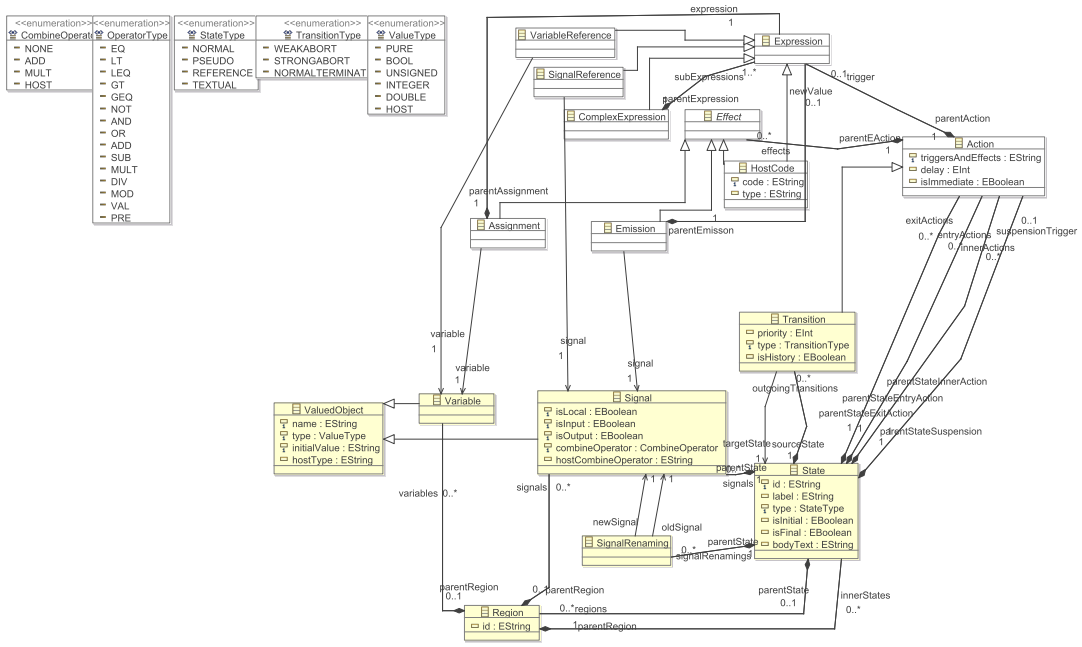

Abbildung 2.1: *Ecore-Diagramm des SyncChart-Editors von Matthias Schmeling*

QUELLE: SCHMELING[36]

quests und Commands verpacken. Das Modell kann ein beliebiges Datenmodell sein. Bei der View handelt es sich um alles, was auf der Zeichenfläche dargestellt wird.

Die grafische Darstellung wird vom Draw2D-Plugin ausgeführt. Draw2D ist ein Bestandteil von GEF und besteht im Kern aus einem *Lightweight System*. Dieses verbindet eine *Figure* mit einer SWT-Zeichenfläche (Standard Widget Toolkit) und stellt Listener für SWT-Events zur Verfügung, die dann für die entsprechende Figure übersetzt und weitergeleitet werden. Ein Update-Manager verarbeitet das Zeichnen an sich und das Basis-Layout; bei einem Update stößt er den entsprechenden repaint-Befehl an.

Der Controller ist mit einem sichtbaren Objekt aus dem Modell verknüpft und stellt somit die Brücke zwischen Modell und View dar. Die Controller heißen *EditParts* und sind in Verbindung mit *Edit Policies* verantwortlich für das Editieren. In Kapitel 5 wird genauer auf EditParts und ihre Verknüpfung mit Modell und View eingegangen.

Ein weiterer, wichtiger Bestandteil von GEF ist die Bereitstellung von Werkzeu-

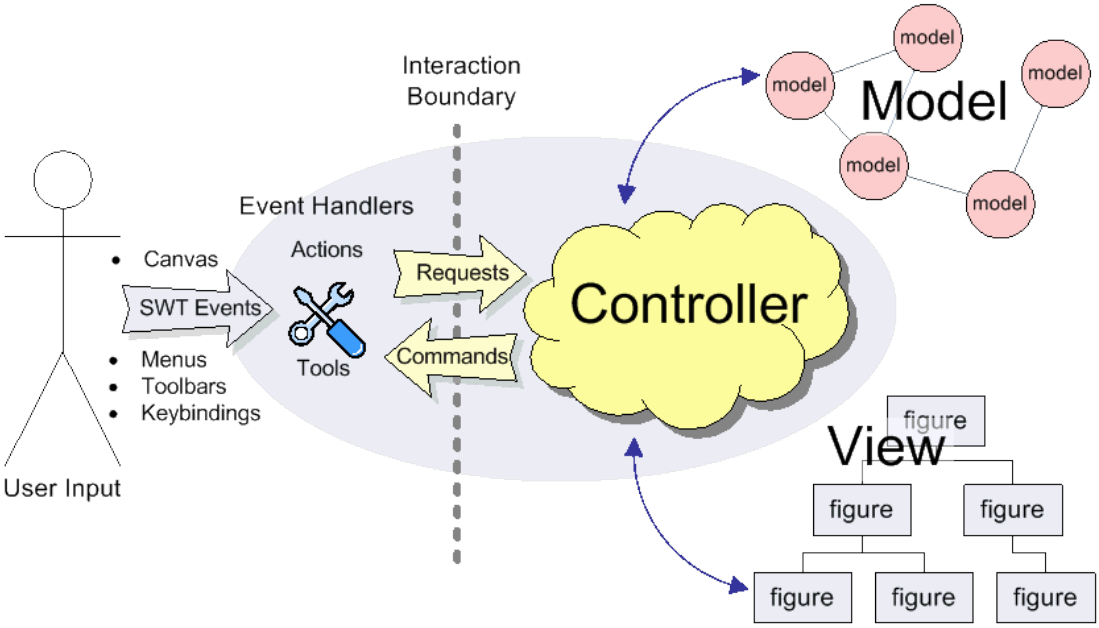

Abbildung 2.2: *GEF (graues Feld) und Kontext*

QUELLE: ECLIPSE HILFE

gen und Paletten für Editoren. Elemente können von dieser Palette direkt auf den Editor gezogen und platziert werden. GEF stellt dafür die entsprechenden *Handler* wie zum Beispiel einen *DragHandler* bereit.

Schließlich ist GEF natürlich auch für die Erstellung, die Bearbeitung und das Löschen von Objekten, Verbindungen, den *Biegepunkten* dieser Verbindungen usw. verantwortlich. Auch all diese Operationen werden über entsprechende *Requests* gehandhabt.

2.5 Graphical Modeling Framework (GMF)

Das *Graphical Modeling Framework* (GMF) ist ein Framework zur Erstellung von grafischen Modellierungseditoren in Eclipse [25]. Es kombiniert die Vorteile und Fähigkeiten von EMF und GEF und erweitert zusätzlich deren Funktionsumfang.

GMF besteht aus einem Werkzeug-Bestandteil und einer Laufzeitumgebung. Mit den Werkzeugen ist es möglich, Modelle für verschiedenste Arten von Editoren und all ihre Bestandteile (Notation, Semantik, Werkzeuge) zu definieren und

aus diesen Modellen direkt Code für ein entsprechendes Plugin zu erzeugen. Die so erstellten Plugins nutzen die GMF-Laufzeitumgebung.

Die Erstellung erfolgt in vier Schritten:

- Erstellung eines Domänen-Modells zur Beschreibung nicht-grafischer Bestandteile im Editor,
- Erstellung eines Modells zur Beschreibung der im Editor vorkommenden grafischen Elemente,
- Erstellung eines Abbildungs-Modells, welches die Verknüpfung zwischen dem Domänen-Modell und den grafischen Elementen beschreibt,
- automatische Erstellung des grafischen Editors.

Die *Runtime* ist eine gemeinsame Basis für Editoren. Die Benutzung einer einheitlichen Basis ist vorteilhaft sowohl für die Benutzbarkeit des Editors, für seine Stabilität und Fehlerfreiheit als auch für die Erweiterungsmöglichkeiten durch andere Plugins.

Die Runtime bietet darüber hinaus eine Reihe nützlicher Features, die problemlos in jedem Editor benutzt werden können:

- *Compartments* – »Schubfächer« von Objekten, die Unterobjekte oder ähnliches enthalten, können erweitert und zusammengefaltet werden.
- *Label* von Objekten können direkt durch Klicken mit der Maus editiert werden. Die vorgenommenen Veränderungen haben direkten Einfluss auf das darunterliegende Datenmodell, eventuell resultierende Veränderungen anderer Bestandteile werden direkt vorgenommen.
- *Popups* ermöglichen das Erstellen und Einfügen von Objekten an der Mausposition. Ähnlich ist das Erstellen von weiteren Verbindungen zwischen Objekten möglich.
- Ein Kontextmenu mit Befehlen wird nach (Rechts-)Klick auf ein Objekt angezeigt.
- Eine Auswahl von Standard-Werkzeugen auf der Palette (Auswahl, Notiz, ...) wird bereitgestellt.

- Ebenso werden Standard-Menus, Werkzeugleisten sowie Erweiterungsmöglichkeiten für selbstdefinierte Elemente vorgehalten.
- Animiertes Zoomen und Layout ist möglich.
- Eine Ansicht zur Veränderung von Eigenschaften für jedes Objekt wird angeboten.
- Eine Druckfunktionalität ist implementiert.
- Exportmöglichkeiten der aktuellen Diagramm-Ansicht in eine Grafikdatei (BMP, PDF, SVG, ...) sind vorhanden.

2.6 KIELER Bestandteile

Der Eclipse-Philosophie »Alles ist ein Plugin« [13] folgend ist auch das KIELER Projekt modular aufgebaut.

Die zahlreichen Plugins, welche meist im Rahmen von Diplom- oder Studienarbeiten erstellt worden sind – und im Folgenden in einer Auswahl vorgestellt werden sollen – formen in ihrer Gesamtheit ein mächtiges Werkzeug. Auch zum Zeitpunkt der Erstellung dieser Arbeit sind noch mehrere bedeutende Erweiterungen in der Entwicklung und werden daher zumeist in ihrer geplanten Funktionalität beschrieben.

2.6.1 ThinKCharts

Das ThinKCharts-Plugin wurde im Rahmen einer Studienarbeit von Matthias Schmeling erstellt [36]. Ziel der Arbeit war, einen SyncCharts-Editor zu erstellen, der zwar kompakt und übersichtlich in der Programmierung sein sollte, aber dennoch eine gute Benutzerfreundlichkeit, ein ansprechendes Äußeres und einen großen Funktionsumfang bieten sollte. Weiterhin war gefordert, auch in der grafischen Repräsentation eine Validierung mit dem *EMF Validation Framework* zu ermöglichen.

Wichtige Features dieses Editors sind:

- *Attribute Awareness*: Zustände und Transitionen sollen nicht von vornherein als bestimmter Typ erzeugt werden, sondern sie sollen bezüglich ihres Typs – zum Beispiel FINAL, INITIAL für Zustände oder WEAK ABORT, STRONG ABORT für Transitionen – jederzeit und flexibel änderbar sein. Das Aussehen der betroffenen Objekte ändert sich entsprechend, auch in Abhängigkeit von anderen Objekten zum Beispiel dem Vorhandensein von Kind-Elementen.

- *Action Parsing*: Trigger, Ausgaben etc. können in einem Label des betroffenen Elements erstellt werden. Sie werden dann geparsed und die so definierten Elemente dynamisch erzeugt.

- *Validation*: Um das Erzeugen ungültiger Diagramme zu verhindern, stellt der Editor Checkmechanismen zur Verfügung, die das Diagramm auf bestimmte Bedingungen überprüfen. Werden diese nicht erfüllt, hebt der Editor die fehlerhaften Elemente grafisch hervor.

- *Automatisches Layout*: Das Layout (bevorzugt mit Graphviz Dot[5]) wird unterstützt.

2.6.2 KIELER Visual Komparision

KIELER Visual Komparison (KiViK) wurde im Rahmen der Diplomarbeit von Arne Schipper [34] am Lehrstuhl erstellt.

Es ist ein Eclipse-Plugin, welches den Vergleich zweier grafischer Modelle ermöglicht und die Unterschiede zwischen beiden wieder grafisch darstellt.

Während der Vergleich zweier textueller Modelle mit EMF Compare von Eclipse unterstützt wird und die Unterschiede in den Modellen grafisch hervorgehoben und verdeutlicht werden[6], gab es bei rein grafischen Modellen bisher keine solche Lösung. KiViK bedient sich zwar der Fähigkeiten von EMF Compare und vergleicht die Unterschiede im Domänenmodell, benutzt dann aber die beiden Domänenmodelle, um daraus eine grafische Repräsentation der verglichenen

[5] http://www.graphviz.org/
[6] http://wiki.eclipse.org/index.php/EMF_Compare

Modelle darzustellen. Die unterschiedlichen Arten von Änderungen (Entfernen, Hinzufügen, Verändern eines Objekts) sind unterschiedlich hervorgehoben und in ihrer Farbgebung intuitiv verständlich.

2.6.3 KIEL Infrastructure for Meta Layout

Das Plugin KIEL Infrastructure for Meta Layout (KIML) wurde ebenfalls von Arne Schipper im Rahmen seiner Diplomarbeit [34] erstellt. Es ermöglicht das Anwenden von Layout-Algorithmen auf grafische Modelle. Das automatische Layouten von Diagrammen ist von immenser Bedeutung für den Entwicklungsprozess, da es ansonsten beim Entwickler liegt, ein gutes, konsistentes und übersichtliches Layout manuell zu erschaffen. Diese Tätigkeit ist in Bezug auf das eigentliche Ziel der Entwicklung nutzlos, denn es finden keinerlei inhaltliche Änderungen am Modell statt. Im Gegenteil birgt das manuelle Verschieben von Elementen ein enormes Risiko der Fehlererzeugung und ist nicht zuletzt sehr ermüdend und frustrierend. Der Aufwand, der getrieben werden muss, um lediglich Platz für ein neu einzufügendes Objekt zu schaffen, kann bei einem großen, dicht gepackten Modell schon enorm sein, der Nutzen im Vergleich dazu sehr gering. Die Möglichkeit, grafische Modelle automatisch zu layouten, ist daher von großer Bedeutung für das KIELER Projekt.

Das KIML Plugin bietet verschiedene Layout-Algorithmen an, die nicht nur auf das ganze Modell, sondern auch auf Teile desselben angewendet werden können. Es ist so generisch angelegt, dass es alle Arten von Diagrammen verarbeiten kann. KIML benutzt intern eine eigene, generische Datenstruktur, den KimlLayoutGraph, in welchen sämtliche Diagramme anfangs übersetzt werden; dafür ist für jeden Editortyp ein entsprechender Filter nötig.

2.6.4 KIELER Execution Manager

Der KIELER Execution Manager (KIEM) ist Bestandteil der momentan laufenden Diplomarbeit von Christian Motika [23].

KIEM soll einen weiteren, sehr wichtigen Bestandteil zu KIELER hinzufügen,

nämlich die Simulation und Ausführung von grafischen, domänen-spezischen Modellen, zum Beispiel EMF-Modellen. Dabei führt KIEM die Simulation nicht direkt aus, sondern integriert die Simulations-, Steuerungs- und GUI-Bestandteile in die KIELER-Umgebung.

Die Implementierung unterstützt ein *Beobachter*- und *Produzenten*schema sowie einen *Ausführungs*-Manager. Der Ausführungs-Manager bietet eine schrittweise oder fortlaufende Steuerung der Simulation und verwaltet die Beobachter und Produzenten.

Eine Tabelle stellt die während der Simulation auflaufenden Daten von Signalen dar und ermöglicht ebenso das Ab- und Zuschalten von Signalen oder das manuelle Setzen von Werten.

KIEM ist für die vorliegende Arbeit von Bedeutung und wird im Kapitel 7 »Evaluation der Lösung« als Anwendungsbeispiel näher betrachtet. Die hier erfolgte Beschreibung von KIEM ist eine Momentaufnahme zum Zeitpunkt der Erstellung dieser Arbeit und beschreibt nicht die endgültige Ausprägung des Plugins. Dem Leser sei die Verfolgung der Projektseite[7] empfohlen.

2.6.5 KIELER Structure Based Editing

KIELER Structure Based Editing (KSBasE) ist ein Plugin, welches von Michael Matzen im Rahmen seiner Diplomarbeit entwickelt wird [21]. Wie in KIEL soll auch in KIELER ein strukturbasiertes Editieren möglich sein, KSBasE soll diese Funktionalität implementieren. KSBasE ist wie KIEM von Bedeutung für die vorliegende Arbeit und wird daher ebenfalls im Kapitel 7 »Evaluation der Lösung« als Anwendungsbeipiel betrachtet.

Auch die Verfolgung der Fortschritte der Entwicklung von KSBasE auf der Projektseite sei dem Leser empfohlen.

[7] http://www.informatik.uni-kiel.de/rtsys/kieler/

2.6.6 KIELER Textual Editing

Das KIELER Textual Editing (KITE) ist ebenfalls ein in der Entwicklung befindliches Plugin, welches von Özgün Bayramoglu im Rahmen ihrer Diplomarbeit entwickelt wird [4].

Ziel dieses Teilprojektes ist es, Modellierung in einem textuellen Editor mit der grafischen Darstellung des so bearbeiteten Modells zu verbinden und eine synchrone Abbildung der beiden Darstellungen dieses Modells zu ermöglichen.

Hierzu wurde eine eigene textuelle Modellierungssprache – KITS – entwickelt; die Entwicklung basiert auf TMF Xtext[8].

[8] http://www.eclipse.org/Xtext/

3 Visualisierung von Modellen – Stand der Entwicklung

> »ἐγὼ δὲ ὀφείλω λέγειν τὰ λεγόμενα, πείθεσθαί γε μὲν οὐ παντάπασι ὀφείλω.«
>
> »Ich soll zwar die Tatsachen berichten, die bereits erwähnt worden sind, ich soll mich aber nicht ganz und gar durch sie bestimmen lassen.« *Herodot, Historien, 7, 152, 3 (Übersetzung: Nils Beckel)*

In diesem Kapitel wird die konzeptionelle und theoretische Basis für die vorliegende Arbeit geschaffen.

Es wird der Stand der Entwicklung auf dem Gebiet des Viewmanagements umrissen. Weiterhin werden die einschlägige Literatur und die wissenschaftlichen Arbeiten diskutiert, die zur Erstellung dieser Arbeit herangezogen wurden. Aus diesem Extrakt des Standes der Entwicklung wird die Ausprägung verschiedenener Bestandteile des Viewmanagements abgeleitet.

Welche der in diesem Kapitel vorgestellten und selbst erstellten Lösungsbausteine sinnvollerweise realisiert werden sollen, wird in Kapitel 4 beschrieben. Es wird dort diskutiert, welche dieser Bausteine voraussichtlich den größten Vorteil für den Entwicklungsprozess haben werden. Diese Ausarbeitung wird in Kapitel 7 noch einmal aufgegriffen, wenn es darum geht, den erreichten Nutzen zu verifizieren.

Mit ihrem Paper »On the Pragmatics of Model-Based Design« [11] bieten die Autoren Fuhrmann und von Hanxleden eine gute Ausgangsbasis für die vorliegende Arbeit. Weiterhin liefert die annotierte Bibliographie von Prochnow und

von Hanxleden [28] eine umfassende Sammlung von Arbeiten zu den im Folgenden behandelten Themen.

Fuhrmann und von Hanxleden führen zunächst den Begriff *Pragmatik* ein. Er beschreibt, wie die Eigenschaften einer (Programmier-)Sprache zum Erreichen des gesetzten Entwicklungsziels beitragen. Damit umfasst er also praktische Dinge in Bezug auf grafische Modellierungssprachen, zum Beispiel wie ein Modell erstellt wird, wie es bearbeitet und betrachtet wird und wie man letztendlich eine visuelle Darstellung erzeugen kann.

Die Autoren führen an, dass diese Dinge insbesondere bei grafischen Modellen bisher vernachlässigt worden sind und dass dies die Vorteile, die diese Entwicklungsart hat, aufhebt oder wenigstens schmälert. Zurückzuführen ist dies nach Aussage der Autoren auf die mangelnde Berücksichtigung der Pragmatik, da diese bei den bisher dominierenden textuellen Modellierungssprachen nur eine untergeordnete Rolle spielte. Das Erstellen und Verändern von textuellen Modellen unterscheidet sich von Sprache zu Sprache oder Entwicklungswerkzeug zu Entwicklungswerkzeug zwar im Detail, generell ist aber die Bearbeitung von Code einfach und effizient.

Die Folgen dieser Vernachlässigung manifestieren sich in einer stark erhöhten Belastung des Entwicklers mit fachfremden Tätigkeiten, also Tätigkeiten, die nicht dem eigentlichen Arbeitsziel, der Entwicklung eines neuen oder der Weiterentwicklung eines bestehenden Programms dienen oder sogar die Problemlösung behindern. Diese fachfremden Tätigkeiten bestehen in besonderem Maße aus dem manuellen Zeichnen des entsprechenden Diagramms. Gelingt es, den Entwickler von dieser Aufgabe weitgehend zu befreien oder ihn dabei sinnvoll zu unterstützen, so ist nach Ansicht der Autoren eine große Verbesserung der Produktivität des Entwicklungsprozesses zu erwarten. Verifiziert wurde diese Erwartung bereits durch Prochnow bei der Anwendung auf KIEL [27].

Hinzuzufügen ist, dass sich die zu erwartenden Verbesserungen nicht nur auf die Produktivität im Sinne einer Beschleunigung des Entwicklungsprozesses beschränken. Es ist anzunehmen, dass die Entlastung des Entwicklers und das Fokussieren auf den eigentlichen Arbeitszweck auch andere Effekte haben wird. Die Konzentration des Entwicklers auf das Wesentliche wird sicherlich auch der Qualität und der Reduzierung der Fehlerhäufigkeit der erzeugten Software zu-

gute kommen, ein Aspekt, der insbesondere unter dem Gesichtspunkt der Entwicklung sicherheitskritischer Systeme, wie es eingebettete Echtzeitsysteme oft sind, von besonderer Bedeutung ist.

Die Autoren Fuhrmann und von Hanxleden propagieren, bei der Enwicklung mit grafischen Modellen das *Model-View-Controller-Paradigma* (MVC) zu betrachten und platzieren die Pragmatik als zentrale, an allen drei Bereichen mitwirkende Komponente; siehe Abbildung 3.1.

Bestandteile der Pragmatik aus dem Bereich Controller sind hier zum Beispiel die Möglichkeit der Simulation oder der Korrektheitsprüfung eines Modells, aus dem Bereich Modell strukturbasiertes Editieren und Modellsynthese, aus dem Bereich View automatisches Layout, Filterung oder ein anderes Verändern der Ansicht des Modells. Die vorliegende Arbeit wird sich auf den Bereich View beziehen.

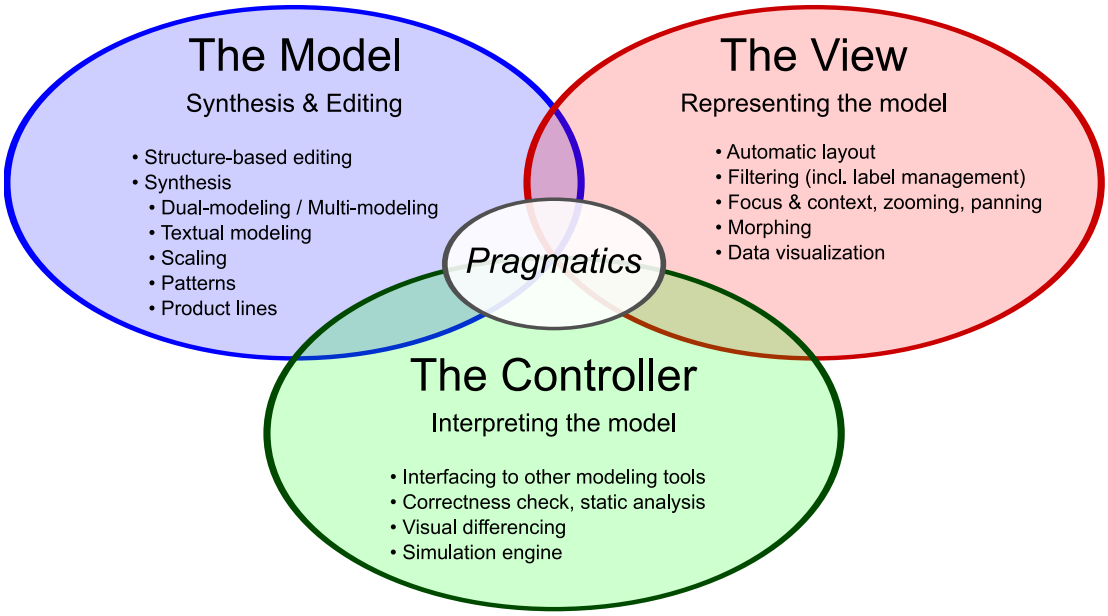

Abbildung 3.1: *Die Pragmatik im Bereich des MVC*

Quelle: Fuhrmann und von Hanxleden [11]

Die Autoren sehen in der Darstellung des Modells, der View und hier besonders im automatischen Layout eine Schlüsselrolle für die Verbesserung der Produktivität. Die dem automatischen Layout beigemessene Bedeutung schlägt sich auch in der Bezeichnung des Projektes KIELER nieder. Das »L« in KIELER steht hier explizit für »Layout« und ist ein Schwerpunkt des Projektes und zentrales

Hilfsmittel bei der Entwicklungsarbeit mit KIELER.

Wie bereits im Abschnitt 1.1 dargelegt, wird ein großer Anteil des Arbeitsaufwandes des Entwicklers darin verschwendet, dass er das Modell manuell layouten muss, das heißt einzelne Objekte – oftmals aber in großer Zahl – verändern oder verschieben muss, um lediglich eine minimale Änderung im Modell vorzunehmen. Gelingt es also, den Entwickler möglichst weitgehend von diesen Aufgabe zu befreien, ist, so die Autoren, eine enorme Steigerung der Produktivität zu erwarten.

Der Thematik des automatischen Layouts wurde bereits in der Forschung breite Aufmerksamkeit geschenkt [2] [3] und sie wurde in mehreren Diplomarbeiten am Lehrstuhl behandelt [18] [34] [37] [27]. Berührungspunkte mit der vorliegenden Arbeit zu dieser Thematik ergeben sich lediglich durch Anwendung der erstellten Techniken.

Automatisches Layout bewirkt zwar schon eine sehr beachtliche Steigerung der Produktivität, jedoch verbleibt das zweite, in Abschnitt 1.1 dargelegte Problem der hohen Informationsdichte, der mangelnden Übersicht über das Diagramm, wie auch Herman et al. beschreiben [14], oder das Verlaufen in Details.

Der Kern dieses Problems liegt also offenbar in der Regulierung der Informationsdichte. Es gilt, Methoden zu finden, um das Wesentliche, Interessante vom Unwichtigen oder wenigstens in diesem Moment nicht Benötigten zu trennen. Das Viewmanagement soll Techniken und Methoden bereitstellen, um dies zu bewerkstelligen. Von besonderem Interessse ist, wann und unter welchen Voraussetzungen und in welcher Ausprägung die durch das Viewmanagement bereitgestellten Techniken und Methoden angewendet werden sollen. Dafür soll die im Rahmen der vorliegenden Arbeit entwickelte Experimentierplattform die nötige Basis zur Erforschung dieser Fragestellung geben.

In ihrem Paper zitieren Fuhrmann und von Hanxleden vier grundlegende Techniken, mit denen die besagte Informationsdichte reguliert werden kann: *Highlighting*, *Filterung*, *selektive Aggregation* und *Verzerrung*.

Das Highlighting ist eine recht simple Technik, die jedoch schon einen enormen Zugewinn an Übersichtlichkeit, Struktur und Lesbarkeit eines Diagramms bringen kann. Objekte von Interesse werden hier durch Veränderungen von Form

und Farbe oder Hinzufügen von Hilfsobjekten hervorgehoben, sodass sie dem Betrachter aus der Menge der anderen Objekte ins Auge springen. Die selektive Aggregation abstrahiert die Ansicht auf die wichtigsten Elemente und schafft somit Raum und Struktur.

Die Filterung ist hierbei wohl die einfachste der möglichen Techniken, nicht benötigte Informationen beziehungsweise Objekte werden einfach ausgeblendet und versteckt. Der Benutzer sieht also nur noch die wesentlichen Objekte. Bei gleichbleibender Anzeigefläche können somit entweder mehr wesentliche Objekte oder mehr Details zu den verbleibenden Objekten dargestellt werden. Problematisch bei dieser Technik ist, zu entscheiden, welche Objekte wirklich unwichtig sind. Da ein solcher Filter binär ist, Objekte also entweder sichtbar oder unsichtbar sind, fällt diese scharfe Trennung unter Umständen schwer.

Allzu oft werden Elemente in einem Diagramm zwar nicht direkt betrachtet und bearbeitet, liefern aber den nötigen Kontext zu den Objekten, die im Fokus des Entwicklers stehen. Es gilt also, *Fokus und Kontext* in das rechte Verhältnis zueinander zu bringen. Eine dynamische Variante der Filterung anzuwenden ist besonders bei Diagrammen mit hierarchischen Zuständen sinnvoll. So können Objekte in den unteren Hierarchieebenen bei Bedarf ausgeblendet werden, die oberen Hierarchieebenen bieten aber noch genug Informationen zur Aufrechterhaltung des Kontextes. Die dynamische Filterung wird durch Expandieren und Komprimieren des in der Hierarchie darüber liegenden Zustandes erreicht.

Fokus und Kontext ist ein zentraler Begriff, dem besondere Aufmerksamkeit gewidmet werden soll. Mit *Fokus* sind hierbei diejenigen Bestandteile gemeint, die in einer Momentaufnahme im Interesse des Entwicklers stehen. Der *Kontext* umfasst diejenigen Objekte, die momentan nicht von unmittelbarem Interesse sind, auf die jedoch nicht verzichtet werden kann, da sie gegebenenfalls wertvolle Zusatzinformationen bereitstellen, die zum Verständnis der Objekte im Fokus beitragen. Im Interesse einer Reduzierung der dargestellten Informationen können die Objekte im Kontext jedoch mit geringeren Details dargestellt werden, zum Beispiel wie oben erwähnt durch Darstellung nur der obersten Hierarchieebenen dieser Objekte.

Ein weiteres Problem stellen die Beschriftungen der einzelnen Objekte dar, die *Label*. Auch hier ergibt sich das Problem der Informationsdichte. Aussagekräf-

tige, aber dadurch auch lange Label in großer Zahl werden schwer darstellbar und beeinträchtigen die Übersichtlichkeit stark. Dem *Label Management*, also der Darstellung und Platzierung vieler, langer Label gebührt daher Aufmerksamkeit. Dieser Problembereich ist jedoch eher im Bereich des Layouts anzusiedeln.

Die bisher aufgeführten Maßnahmen zur Regulierung der Informationsdichte, Highlighting, Fokus und Kontext, Filterung usw. lassen sich unter dem Begriff *Effekte* zusammenfassen. Diese Effekte tragen zweifellos zur Verbesserung der Produktivität bei der Arbeit mit Diagrammen bei, jedoch ist der Zeitpunkt ihres Auftretens noch undefiniert. Sie brauchen einen Anstoß, eine Ursache, einen *Trigger*.

Trigger lassen sich nach Fuhrmann und von Hanxleden in zwei Arten einteilen, *System-Trigger* und *Benutzer-Trigger*. Benutzer-Trigger sind generell Handlungen, die vom Benutzer ausgehen, und zwar durch aktive Handlung seinerseits. Denkbare Benutzer-Trigger sind beispielsweise das Klicken und Markieren eines Objektes, das Klicken eines Buttons, das Auswählen eines Menu-Befehls. System-Trigger hingegen sind solche, die automatisch vom System beim Erreichen bestimmter Bedingungen auftreten. Beispiele hierfür sind Trigger, die von einem Analyse-Tool stammen und anzeigen, dass ein bestimmtes Objekt fehlerbehaftet ist, oder solche, die von einer Simulations-Engine erzeugt werden, die ausgibt, dass ein bestimmter Zustand aktiv ist oder ein bestimmtes Signal anliegt.

Die Aufgabe des Viewmanagements ist es also, Trigger und Effekte sinnvoll zu verknüpfen und zu steuern. Auf der Effekt-Seite soll es die oben beschriebenen Standardeffekte bereitstellen. Zusätzlich soll die Möglichkeit vorhanden sein, andere Effekte anzustoßen, wie etwa Layout oder *Metalayout*, welches ein vom generellen Layout des gesamt Diagramms abweichendes Teillayout eines ausgewählten Teildiagramms darstellt.

Weiterhin soll das Viewmanagement auf eine Reihe von Triggern reagieren, sowohl auf solche wie die bereits erwähnten Benutzer-Trigger (Mausklicks, Tastaturbefehle usw.) als auch auf System-Trigger wie die oben genannten möglichen Simulations-Trigger. Generell soll das Viewmanagement für vom Benutzer spezifizierte Effekte und Trigger offen sein und einfache, generische Schnittstellen

für diese bieten.

Ein weiterer, sehr wichtiger Bestandteil des Viewmanagements ist die Kombination von Effekten und Triggern. So muss es natürlich möglich sein, zu definieren, welcher Trigger welchen Effekt oder welche Effekte hervorrufen soll und welche Ausprägung diese haben sollen. Es soll auch möglich sein, zum Beispiel im Zusammenhang mit Simulation, differenziertere Kombinationen zu definieren, etwa das Auslösen eines Effekts unter ganz speziellen Bedingungen oder erst nach mehrfachem Auftreten solcher Bedingungen.

Auch Tory und Möller stellen in ihrem Paper »Human Factors In Visualization Research« fest, dass in der Visualisierung von Daten im Allgemeinen eine große Bedeutung für die Datenverarbeitung liegt [40]. Sie machen dies unter anderem an der steigenden Anzahl wissenschaftlicher Veröffentlichungen zu diesem Thema fest.

Die Komplexität und Menge der in heutigen Systemen vorkommenden Informationen – zum Beispiel in der Medizin oder in den Geowissenschaften – ist nach Ansicht der Autoren eine kognitive Herausforderung besonderer Art. Sie meinen, dass durch ausgefeilte Mechanismen der Darstellung von Daten die Möglichkeiten menschlicher Aufnahmefähigkeit besser als bisher genutzt werden können, sofern ein maschineller, kognitiver Support geleistet wird.

Es werden als die grundlegenden Methoden, die mithilfe von Visualisierung die menschliche Kognition verbessern können, die folgenden angeführt:

- Vergrößern der Ressourcen zur Datenverarbeitung durch Übertragung von unterstützenden Aufgaben auf den Computer: Parallele Verarbeitung von Daten durch ein visuelles System zur Verdeutlichung der regulären (textuellen) Repräsentation, Auslagern gewisser Operationen, die visuell einfacher dargestellt und repräsentiert werden können, Unterstützung des menschlichen Kurzzeitgedächtnisses durch Auslagern von Informationen in visuelle Repräsentationen, erhöhte Speicher- und Darstellungsfähigkeit von visuellen Darstellungen im Allgemeinen.
- Reduzierung des Suchaufwandes: Gruppierung von verwendeten Informationen, Darstellung hoher Datendichte durch visuelle Repräsentationen,

Strukturieren der Daten zur Reduzierung der Komplexität.

- Verbesserte Wiedererkennung: Einfacheres Wiedererkennen von visuell dargestellten Informationen, Abstrahieren und Aggregieren durch Ballen oder Auslassen von gewissen Informationen.
- Präattentive – also vorbewusste, unterschwellige Wahrnehmung einer großen Anzahl von Events bei visueller Repräsentation.
- Interaktive Betrachtung und Manipulation von Daten, vielfältige Änderungsmöglichkeiten der Struktur.

Als ein wichtiges Element für eine produktivitätsfördernde, erfolgreiche Visualisierung von Daten sehen Tory und Möller insbesondere die Kontinuität der Darstellung an. Es soll nach Kräften vermieden werden, dass der Benutzer während der Bearbeitung und Betrachtung der Daten die Übersicht verliert. Sie bezeichnen dies als *visuellen Impetus*. Als Maßnahme zur Aufrechterhaltung desselben wird zum Beispiel die Beibehaltung von Formatierungen und das sanfte Zoomen und Scrollen empfohlen. Es soll für den Betrachter jederzeit einfach erkennbar bleiben, welche Beziehungen die Objekte zueinander haben, an welcher Stelle in der Darstellung er sich gerade befindet, wie die nähere Umgebung der gerade betrachteten Objekte aussieht und an welcher Stelle sich der Betrachter bisher in der Darstellung aufgehalten hat, es soll also eine Art *Chronik* erstellt werden. Die Daten, die gerade im Fokus der Betrachtung stehen, sollen unverzerrt und mit höchstmöglichem Detailgrad dargestellt werden.

Konkrete Beispiele für Techniken zur Erreichung dieser Anforderungen sind nach Ansicht der Autoren *depth of focus* (»Tiefenschärfe«, auch *depth of field*) – hierbei werden die Objekte in einem bestimmten Bereich scharf dargestellt, die Objekte außerhalb davon unscharf und verschwommen – und Highlighting durch die Benutzung von Farben, Ausrichtung und anderen grafischen Veränderungen. Dies hat den Effekt, dass die so manipulierten Objekte aus der Darstellung »heraussspringen« und die Suche dieser Objekte dem Betrachter sehr leicht fällt. Durch die Benutzung des Highlightings für alle Objekte, aber mit unterschiedlichen farbigen Ausprägungen, lassen sich Gruppierungen und Trennungen erreichen.

Für das Viewmanagement ist in jedem Fall das Highlighting als einfache, aber –

nach Aussage der Autoren – effektive Technik zur Steigerung der Produktivität beim Arbeiten auf Diagrammen von Interesse. Die Technik der Hervorhebung von Objekten mittels Tiefenschärfe erscheint viel versprechend, wird aber aufgrund der anspruchsvollen Technik zunächst nicht im Viewmanagement zum Einsatz kommen. Eventuell ergeben sich hier Möglichkeiten für zukünftige Arbeiten.

Als problematisch betrachten Tory und Möller die Darstellung von großen Datenmengen. Auch sie sehen die Schwierigkeit, bei großen Datenmengen alle Objekte zwar darstellen, dafür aber keine davon lesen zu können oder aber einige wenige Objekte im Detail darzustellen, dafür aber die globale Orientierung zu verlieren. Sie weisen zusätzlich auf die kognitive Anstrengung hin, die der Betrachter aufbringen muss, um diese Schwächen in der Darstellung auszugleichen. Auch die Erweiterung des Bildschirms auf eine Größe, auf der alle gewünschten Objekte lesbar mit allen Details darstellbar sind, wird aus technischen, aber auch Gründen der begrenzten menschlichen Auffassungsmöglichkeit, kritisch gesehen. Jedoch weisen Tory und Möller darauf hin, dass bisher keine Studien zum Einfluss der Bildschirmgröße auf die Wahrnehmungsmöglichkeiten existieren. Es ist also zur Zeit nicht erforscht, wann, in welcher Situation und in welchem Umfang eine Erhöhung der Bildschirmgröße und/oder -auflösung die Wahrnehmungsleistung steigert.

Eine mögliche Lösung für dieses Problem, welches sie als *screen real-estate problem* bezeichnen, sehen die Autoren im *Detail und Übersicht*-Prinzip an, welches aus einer Hauptansicht mit dem momentan relevanten Ausschnitt der Gesamtdarstellung in vollem Detailgrad und einer Übersichtskarte, die eine vereinfachte, schematisierte Darstellung der Gesamtdarstellung repräsentiert, besteht. Auf der Übersichtskarte wird der gegenwärtig angezeigte Ausschnitt mit einem Icon kenntlich gemacht.

Doch auch diese Lösung kann bei gewissen Anwendungen unnötigen kognitiven Aufwand bedeuten, nämlich dann, wenn der Detailgrad von Übersichtskarte und Arbeitsansicht zu stark voneinander abweichen und der Benutzer den Zusammenhang zwischen beiden Karten nicht oder nur schwer erkennen kann. Nach diesem Prinzip müsste jetzt eine dritte Karte mit einem Abstraktionsgrad zwischen Übersichtskarte und Arbeitsansicht eingeführt werden, was aber we-

nig sinnvoll erscheint und langfristig auch durch das Hinzufügen von immer mehr Karten höchstwahrscheinlich wenig Nutzen bringen wird.

Daher wird von Tory und Möller zusätzlich die Benutzung der Fokus und Kontext-Technik empfohlen, bei der zusätzlich zu den im Fokus stehenden Objekten ein erklärender Kontext von Objekten bereitgestellt wird.

Ziel all dieser Techniken ist immer die Aufrechterhaltung und Verbesserung der mentalen Karte, die auch nach Tory und Möller jeder Anwender im Kopf hat und mit der er die angezeigten Bilder vergleicht. Generell legen die Autoren Wert auf die Feststellung, dass Visualisierung die Zusammenarbeit von Mensch und Computer verbessern und die Stärken der beiden Systeme vereinen soll.

Die mentale Karte (*mental map*) sehen viele Autoren [34] [39] [10] [38] [6] – wie auch schon in Abschnitt 1.1 angemerkt – als zentralen Bestandteil bei der Betrachtung und Bearbeitung großer, visuell dargestellter Datenmengen an. Hochmair und Frank beschreiben in ihrem Paper »A Semantic Map as Basis for the Decision Process in the www Navigation« [15] das Navigieren zu einem unbekannten Ziel hin als übliche und seit langem verbreitete Aufgabe des Menschen, die er auch schon vor dem Aufkommen virtueller, computergestützter Darstellungen gemeistert hat.

Dies ist einsichtig, da doch der Mensch in jedem Fall, in dem es der Orientierung in einem nicht zu überschauenden Gebiet bedarf, eine Vorstellung davon hat, wo er sich aufhält und wo sich sein Ziel befindet. Einfache Beispiele finden sich in alltäglichen Situationen wie etwa im Supermarkt, in einer Bibliothek, im Straßenverkehr oder aber auf höherem Niveau in der Nautik und Luftfahrt.

Die für viele Situationen vorhandene mentale Karte des Menschen ist hierbei nicht starr und unflexibel, sondern nimmt laufend Änderungen auf und passt die Karte entsprechend an. Mit einer Situation konfrontiert, vergleicht der Mensch das sichtbare Bild mit seiner mentalen Karte und ermittelt daran seine momentane Position und die nötigen Navigationsschritte zu seinem Ziel.

Jedoch verkraftet die mentale Karte nur eine begrenzte Menge an Änderungen. Je nach Komplexität der durch die Karte abstrahierten Realität und dem nötigen

Detailgrad der Karte ist die Möglichkeit zur Aufnahme von Informationen über Änderungen beschränkt.

Ein weiteres Kriterium für die Nutzbarkeit der mentalen Karte ist, dass der Mensch in der Lage ist, die Änderungen überhaupt wahrzunehmen. Finden etwa große Änderungen außerhalb des Sichtbereiches des Menschen statt, die gravierende Auswirkungen auf die mentale Karte haben, stimmen Realität und Karte nicht mehr überein; die Positionsbestimmung oder Navigation ist fehlerhaft, der Mensch somit orientierungslos.

Die Erhaltung der mentalen Karte ist für das Viewmanagement eine wichtige Aufgabe bei der Verbesserung der Produktivität beim Arbeiten mit visuellen Modellen.

Keim sieht wie auch Tory und Möller eine große Bedeutung in der besseren Verknüpfung der Systeme Mensch und Computer und ihrer Stärken. Er bezieht sich hierbei auf den Bereich *Data-Mining*, wie er in seinem Paper »Information Visualization and Visual Data Mining« [17] darlegt. Der Mensch habe die Flexibilität, die Kreativität und das Allgemeinwissen, Eigenschaften also, welche, verknüpft mit der Speicherkapazität und Rechenleistung eines modernen Computers, ausgezeichnet geeignet seien, große Mengen an Daten in visueller Darstellung zu verarbeiten.

Der Autor sieht die Vorteile der visuellen Datendarstellung insbesondere in der Möglichkeit, inhomogene Daten einfach darzustellen und diese auf eine intuitive Weise, die wenig Fachwissen – etwa auf dem Gebiet der Statistik – erfordert, abzubilden. Er propagiert das *Information Seeking Mantra*, das den dreistufigen Prozess der Informationsexploration beschreibt: »Overview first, zoom and filter, and then details on demand.« Der Benutzer verschafft sich in diesem Prozess zuerst eine Übersicht über die dargestellten Daten, zoomt dann auf einen Bereich von Interesse und betrachtet schließlich die zum erwünschten Verständnis nötigen Details.

Keim widmet der Klassifikation von visuellen Data-Mining-Techniken große Aufmerksamkeit, um so eine geeignete Kombination aus Daten und Darstellungstechnik zu finden. Zusätzlich bezieht er eine Interaktions- und Verzerrungstech-

nik in die Bewertung ein, denn eine unmittelbare Interaktion mit den Daten hält er für unverzichtbar.

Bei der Interaktion mit den Daten trennt er zwischen der *dynamischen* und der *interaktiven Interaktion*, abhängig davon, ob sie automatisch oder manuell durch den Benutzer stattfindet. Keim führt hierbei das interaktive Zoomen an, welches die Objekte nicht nur vergrößert, sondern auch den Detailgrad anpasst, sowie die interaktive Verzerrung mittels Fischauge et cetera. Letztgenannte Technik wird im weiteren Verlauf des Kapitels näher beschrieben.

Das interaktive Zoomen erscheint ebenfalls wichtig und soll daher durch die im Viewmanagement realisierten Techniken ermöglicht werden. Es muss also ein Weg gefunden werden, der nicht nur Zoomen als schlichte Vergrößerung darstellt, sondern auch intelligent den Detailgrad der entsprechenden Objekte anpasst.

Storey et al. beschreiben in ihrem Paper »Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization« [38] die Erstellung einer solchen mentalen Karte als kognitives Modell, welches den kognitiven Prozess und die zur Erstellung benutzte Informationsstruktur beschreibt.

Sie unterscheiden zwischen einem *bottom-up* und einem *top-down* Ansatz, mit dem solch eine mentale Karte für die Funktionalität eines Programms geschaffen wird. Beim *bottom-up* Ansatz befasst sich der Betrachter mit der detailliertesten Darstellung des Programms, dem Quellcode. Er liest diesen und abstrahiert eigenständig die generelle Funktionalität des Programms. Der *top-down* Ansatz geht den anderen Weg, bei dem der Betrachter zunächst eine Hypothese über die Funktionalität des Programms aufstellt und diese dann durch andere, hierarchisch darunter liegende Hypothesen verfeinert. Diese Hypothesen versucht er dann zu verifizieren, wobei nicht bis auf den Quellcode ins Detail gegangen werden muss.

Von Interesse für das Viewmanagement ist der top-down Ansatz, da dieser mit der modellbasierten Entwicklung zu vereinbaren ist. Das Viewmanagement soll also Methoden zur Verfügung stellen, mit Hilfe derer der Detailgrad angepasst werden kann. Der bottom-up Ansatz dagegen ist für ein Modell, welches per

Definition eine Abstrahierung der Wirklichkeit darstellt, nicht zu realisieren.

Storey et al. führen an, dass Tools zur Visualisierung von Daten einen der beiden Ansätze oder auch eine Kombination aus beiden verfolgen können, es jedoch wichtig ist, entsprechende Maßnahmen zur Erhaltung und Verstärkung der mentalen Karte zu ergreifen. Die Autoren bezeichnen eine Darstellung als kohärent, wenn der Betrachter eine mentale Karte erzeugen kann, die im Zusammenhang mit der realen Welt steht. Weiterhin trennen sie zwischen lokaler und globaler Kohärenz.

Sie sprechen von lokaler Kohärenz, wenn der Betrachter aus der Visualisierung ein Verständnis von der dargestellten Funktionalität im Detail erlangen kann, von globaler Kohärenz, wenn das Gesamtverhalten des dargestellten Systems aus der Darstellung verständlich wird.

Konkret werden Hilfsmittel vorgeschlagen, die die Abstraktion unterstützen sollen – etwa das Kollabieren von einer Menge an Objekten in ein einzelnes, ein Subsystem repräsentierendes Objekt oder das Anzeigen von Übersichtsfenstern.

Dieses Hilfsmittel erscheint wirkungsvoll und soll als eine der Methoden im Viewmanagement realisiert werden.

Weiterhin weisen die Autoren darauf hin, dass die mentale Karte von Betrachter zu Betrachter individuell verschieden sein kann, die Hilfsmittel zur Erstellung, Bewahrung und Verstärkung dieser Karte also ebenso individuell anpassbar sein sollten.

Der Reduzierung des kognitiven Mehraufwandes bei der Navigation mittels der mentalen Karte widmen Storey et al. ebenfalls ihre Aufmerksamkeit. Durch Highlighting soll der momentane Fokus des Betrachters festgehalten und verdeutlicht werden. Weiterhin soll eine Art Chronik der Navigationsschritte angezeigt werden, um deutlich zu machen, wie der Betrachter zu diesem momentanen Fokus gelangt ist.

Dem Problem der Behandlung langer und zahlreicher Beschriftungen (Label von Knoten und insbesondere von Kanten) widmen sich Wong et al. in ihrem Paper »Dynamic Visualization of Graphs with Extended Labels«. [44]

Wong et al. haben erkannt, dass das Problem des Labelmanagements insbesondere in der Zukunft ein drängendes Problem werden wird. Sie sehen die Notwendigkeit, Label mit einer Länge von bis zu einem gesamten Absatz sinnvoll abzubilden. Jedoch wollen sie in ihrem Paper zunächst nur Labels mittlerer Länge betrachten.

Die Autoren bearbeiten damit indirekt auch das Problem der Darstellung zahlreicher Objekte auf stark beschränktem Platz, denn Label sind in diesem Zusammenhang ebenfalls Objekte. Um dieses Problem durch die Einbindung langer Label nicht noch zu verschärfen, propagieren sie die Benutzung des bereits vergebenen Platzes auf der Bildschirmfläche zur Darstellung von Labeln. Die Darstellung soll sozusagen überladen werden, ein Element – also hier die Label – soll mehrere Funktionen übernehmen.

Der dazu von Wong et al. entwickelte Prototyp namens *GreenArrow* bietet eine Reihe von Techniken zur intelligenten Einbindung von Labeln, als erste einfache Lösung die Ersetzung der Verbinder von Objekten durch die Label selbst. Die Richtung der Verbinder wird dabei durch Verkleinerung der Schriftgröße in Richtung Verbinder-Spitze erreicht. Sollte das Label zu kurz für den darzustellenden Verbinder sein, wird es wiederholt. Geschwungene Verbinder können ebenfalls dargestellt werden, hierzu werden die Buchstaben des abzubildenden Labels leicht gedreht, sodass sie nicht mehr senkrecht stehen und somit ein flüssiges und gut lesbares Schriftbild darstellen. Dies scheint bei geschwungenen Verbindern gut zu funktionieren, jedoch lassen die Autoren Verbinder mit mehrfachen, gegenläufigen Biegungen oder gar rechtwinkligen Abzweigungen unbetrachtet.

Bei zu langen Labeln schlagen die Autoren vor, diese bei statischer Anzeige abzuschneiden, alternativ das Label dynamisch zu animieren und so den gesamten Text darzustellen. Das *GreenArrow*-System zeichnet die Label ab einer gewissen Größe, nämlich wenn sie nicht mehr lesbar sind, als entsprechend geformte Linie.

Als weiteres Problem, welches es zu lösen gilt, identifizieren Wong et al. die Darstellung von Knoten mit zahlreichen ein- und ausgehenden Verbindern, die sich überlappen. Für den Betrachter ist es schwer oder gar unmöglich, die einzelnen Verbinder und ihre Laufrichtung auseinander zu halten. Die Lösung, die

GreenArrow für dieses Problem bereit hält, besteht darin, die zahlreichen, gebündelten Label semitransparent zu machen. Somit sind die oben liegenden Verbinder durchscheinend und man kann die darunter liegenden trotzdem sehen. Dies dient der Lesbarkeit von Diagrammen mit stark zentraler Orientierung nach Aussage der Autoren in beträchtlichem Umfang.

Bei der Platzierung von Labeln an Knoten schlagen die Autoren zwei Lösungsmöglichkeiten vor. Anstatt die Label neben den Knoten in horizontaler Schreibweise zu platzieren, sollte man sie besser um den Umriss des Knotens herum legen. Dies ist insbesondere bei kreisförmigen Knoten sehr effizient. Bei rechteckigen Knoten wird die Darstellung des Labels innerhalb des Knotens propagiert. Sollte das Label hierfür zu groß sein, empfehlen Wong et al. das dynamische Scrollen des Textes oder das Anzeigen eines Popup-Fensters bei Bedarf.

Dieses Thema berührt das Viewmanagement eher am Rande, da es sich mehr um ein Layout-Problem handelt. Jedoch ist es für die Pragmatik beim Arbeiten mit grafischen Modellen zweifellos von Bedeutung und somit auch für das gesamte KIELER-Projekt, dessen zentrales Hilfsmittel – wie in Kapitel 2.1 bereits erwähnt – das Layout ist. Für das Viewmanagement wäre jedoch die Möglichkeit, unterschiedliche Darstellungen von Labeln anstoßen zu können, von großem Interesse. Konkret traten Probleme mit sehr langen Labeln schon im Rahmen des *Modellbahn*-Projektes[1] am Lehrstuhl auf.

Dem Thema Fokus und Kontext widmen sich eine Reihe von Autoren. Kosara et al. beschreiben in ihrem Paper »Focus+Context Taken Literally« [19] eine Technik dafür, *depth of field* (DOF), also die Tiefenschärfe. Tiefenschärfe findet als Stilmittel in der Fotografie bereits seit geraumer Zeit Anwendung, hierbei wird ein Objekt im Vordergrund, welches betont werden soll, mit besonders kleiner Blende fotografiert, ein Vorgehen, welches in geringer Tiefenschärfe resultiert. Das so entstandene Foto zeigt das Objekt im Vordergrund scharf, der gesamte Hintergrund ist jedoch unscharf und verschwommen. So wird auf natürliche und intuitive Weise das Objekt im Vordergrund hervorgehoben und betont, es springt dem Betrachter unmittelbar ins Auge. Die präattentive Kognition des

[1] https://www.informatik.uni-kiel.de/rtsys/modelleisenbahn/

Menschen ermöglicht somit eine sehr schnelle und effiziente Erkennung zahlreicher Objekte im Fokus, ohne aber andere Kognitionsprozesse zu stören.

Kosara et al. beschreiben eine auf Visualisierung zugeschnittene Weiterentwicklung dieser Technik, das *semantic depth of field* (SDOF). Hierbei ist nicht die Entfernung der Objekte zu einer Kamera entscheidend für die verschwommene oder scharfe Darstellung eines Objektes, sondern ihre Relevanz. Den Autoren zufolge trennt das menschliche Gehirn automatisch das betrachtete Bild in Vorder- und Hintergrund und somit in wichtige und unwichtige Stimuli, jedoch nicht basierend auf der tatsächlichen Entfernung zum Betrachter, sondern anhand der Fokussierung durch das Auge.

Beim SDOF werden Objekte anhand ihrer semantischen Bedeutung scharf oder verschwommen dargestellt, das Gehirn ordnet instinktiv die scharf dargestellten Objekte dem interessanten Vordergrund zu, alle verschwommen dargestellten Objekte dem weniger wichtigen Hintergrund – selbst dann, wenn sich der Vordergrund räumlich gesehen über die ganze Tiefe des Raumes erstreckt.

Nach Erkenntnissen von Kosara et al. ist diese Technik außerordentlich effektiv und effizient bei Darstellungen beliebiger Art, nicht nur der bei Visualisierung, sondern auch bei textueller Darstellung.

Diese offenbar effektive und viel versprechende Technik wird beim Viewmanagement aufgrund der anspruchsvollen technischen Umsetzung zunächst nicht implementiert werden, ist aber für spätere Weiterentwicklungen von Interesse. Auch für andere Teilprojekte des KIELER-Projektes könnte diese Technik hilfreich sein, etwa für das textuelle Modellieren und seine Verknüpfung mit der grafischen Repräsentation des so erzeugten Modells.

Zum Thema Fokus und Kontext haben Lukka et al. eine einfache, aber effektive Methode entwickelt, um Daten in hoher Dichte darzustellen, dabei aber die Über- bzw. Unterschneidung von Knoten durch Kanten eindeutig abzubilden. Sie beschreiben diese Technik in ihrem Paper »Fillets: Cues for Connections in Focus+Context Views of Graph-like Diagrams« [20].

Die Autoren beschreiben das Problem, dass man bei dicht gepackten Diagrammen unter Umständen nicht unterscheiden kann, ob ein Verbinder zwischen

zwei Knoten einen dritten verbindet oder unterschneidet, also keine Verbindung zu den Randknoten existiert (siehe Abbildung 3.2).

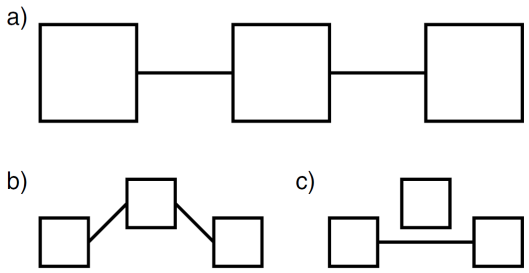

Abbildung 3.2: *Doppeldeutige Darstellung bei großer Objektdichte, a) die angezeigte Darstellung, b) und c) die Interpretationsmöglichkeiten*

QUELLE: LUKKA ET AL. [20]

Die herkömmliche Darstellung mit simplen Strichen, die höchstens Pfeilspitzen haben, ist hier nicht eindeutig oder wenigstens schwer zu lesen und missverständlich. Die einfachste Lösung wäre es nach Ansicht von Lukka et al., im Falle einer Unterschneidung den Verbinder links und rechts vom unterschnittenen Knoten zu unterbrechen und damit eindeutig eine Anknüpfung dieses Knotens auszuschließen (siehe Abbildung 3.3). Problematisch bei dieser Darstellung ist allerdings die Wahl der Größe dieser Unterbrechung. Bei verkleinerter Darstellung kann dies wieder schwer zu erkennen und damit missverständlich sein.

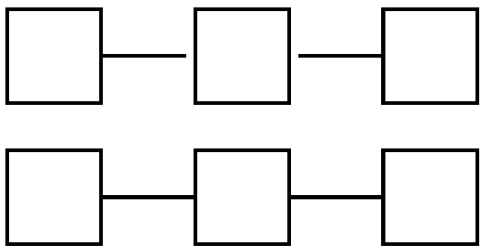

Abbildung 3.3: *Lösungsansatz für das Problem aus Abbildung 3.2*

QUELLE: LUKKA ET AL. [20]

Besser ist hier die von den Autoren entwickelte Darstellung mit *Fillets*. Diese Technik ist aus der mechanischen Industrie bei der Formung von Gegenständen bekannt. Scharfe Ecken und Winkel sind beim Formen anfällig für Defekte und sollten – wenn möglich – vermieden werden. Die Zielsetzung ist daher, dass sich

Flächen nicht in spitzem, scharfem Winkel treffen, sondern in sanften Kurven. Wendet man diese Technik auf eckige Knoten und Verbinder an, so existieren an deren Verknüpfungspunkten keine rechten Winkel mehr, auch eine Trennlinie fällt weg. Bei Unterschneidung eines Knotens ist jedoch ein rechter Winkel vorhanden und eine Trennlinie existiert – die Unterscheidung zu einer Verknüpfung ist eindeutig möglich (siehe Abbildung 3.4).

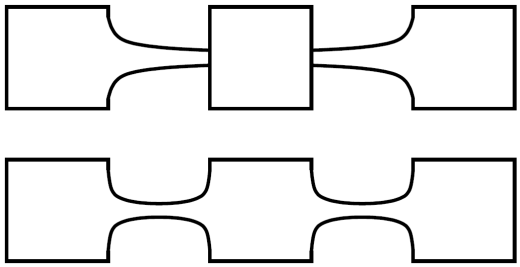

Abbildung 3.4: *Fillets als Lösung des Unterscheidungsproblems*

QUELLE: LUKKA ET AL. [20]

Experimentelle Untersuchungen, bei der Teilnehmer verbundene von unterschnittenen Knoten unterscheiden sollten, zeigten eine exzellente Performance dieser Lösung im Vergleich zur herkömmlichen Darstellung. Besonders bemerkenswert ist die relative Unabhängigkeit der Suchdauer und Fehlerhäufigkeit von der Größe der insgesamt dargestellten Objekte.

Diese nachweislich effektive und effiziente Technik fällt – wie auch das Labelmanagement – eher in den Aufgabenbereich des Layouts und wird daher nicht im Viewmanagement berücksichtigt. Jedoch ist sie im Hinblick auf das Gesamtziel – die Verbesserung der Produktivität beim Arbeiten mit KIELER – durchaus von Interesse und sollte als Gegenstand zukünftiger Forschung in Betracht gezogen werden.

Herman et al. bieten mit ihrem Paper »Graph Visualization and Navigation in Information Visualization: A Survey« [14] eine umfassende Studie auf dem Gebiet der Visualisierung. Sie adressieren die bereits genannten Probleme von Übersicht vs. Details und Platzmangel auf dem Bildschirm und widmen dem Layout von Graphen große Aufmerksamkeit.

Von besonderem Interesse ist das Kapitel »Navigation and Interaction«. Die Autoren stellen hier die These auf, dass Layout allein ohne funktionierende Navigations- und Interaktionsmechanismen die Probleme großer Graphen nicht lösen kann. Dies gilt ihrer Meinung nach im Besonderen für die Struktur eines Graphen.

Herman et al. führen *Zoom & Pan* als eine der traditionellen, aber unverzichtbaren Techniken der Visualisierung an. Die einfachen, im Graph vorkommenden grafischen Elemente machen typische Probleme beim Zoomen, etwa *Aliasing* vermeidbar, da die Objekte aus einfachen geometischen Figuren bestehen, die problemlos vergrößert werden können. Das Zoomen ist daher gut anwendbar. Hierbei werden zwei Arten von Zoomen unterschieden; das *geometrische Zoomen* vergrößert einfach die vorhandenen Elemente, das *semantische Zoomen* zeigt zusätzlich mehr Details an. Bei letzterem ist der *Level of detail* für die jeweilige Zoomstufe sorgfältig zu definieren. Jedoch adressieren die Autoren mehrere Probleme, welche das Zoomen aufwirft: Zoomt man hinein, um Details zu erkennen und will dann einen anderen Punkt im Detail betrachten, muss man herauszoomen und dann auf den gewünschten neuen Punkt hinein zoomen.

Als Abhilfe schlagen Herman et al. den Einsatz von *Fischaugen-Verzerrung* vor, mit der Objekte im Fokus vergrößert werden, Objekte außerhalb davon aber verkleinert sichtbar bleiben. Somit ist der Kontext noch sichtbar. Obwohl diese Technik bereits aus der Optik bekannt ist, birgt sie für Graphen weitere Probleme: Beispielsweise werden rechtwinklige Knoten verzerrt und können für Irritationen beim Betrachter sorgen. Abhilfe bringt eine Variation der Fischaugen-Verzerrung, die nicht optisch verzerrt, sondern nur die Größe der Objekte einer solchen Verzerrung entsprechend verändert. Die Geometrie der Objekte bleibt unverändert.

Eine ähnliche Technik, nämlich das Verringern und Erhöhen des Detailsgrads von Objekten, wird im Viewmanagement zum Einsatz kommen, jedoch vorerst nicht eine semantische Fischaugen-Verzerrung. Die Realisation dieser Technik sollte jedoch keine größeren Schwierigkeiten darstellen und könnte somit leicht zum Gegenstand zukünftiger Forschung werden, bei der auch die Effektivität dieser Technik untersucht werden könnte.

Die von Jacobs angefertigte Diplomarbeit »Konzepte zur besseren Visualisierung grafischer Datenflussmodelle« [16] bezieht sich zwar – wie der Titel besagt – speziell auf Datenflussmodelle, einige der beschriebenen Verfahren und Vorgehen sind aber generisch genug, um auf Diagramme im allgemeinen angewandt werden zu können.

Der Autor unterscheidet bei der Navigation in hierarchischen Modellen zwischen logischer und physikalischer Navigation und versteht unter ersterer vor allem das *explosive* oder *Fisheye-Zoomen*. Explosives Zoomen ist ein Verfahren, bei dem der Fokus auf ein hierarchisches Element wechselt; sämtliche anderen, vorher angezeigten Elemente werden ausgeblendet, das heißt, es wird eine völlig neue Sicht generiert. Zur physikalischen Navigation zählen *Scrollen* und *Scaling*. Auch hier wird das Navigieren als nicht-essentielle Aufgabe des Benutzers beschrieben, sondern als mechanischer und kognitiver Ballast. Eine Minimierung dieses Ballastes hält auch Jacobs für sinnvoll; idealerweise soll das benutzte Werkzeug – die benutzte Software – die vom Benutzer gewünschte Sicht vorhersagen und automatisch einstellen können.

Das explosive Zoomen stellt besondere Anforderungen an den Betrachter, da die Aufrechterhaltung der mentalen Karte hierbei besonders schwer ist und Fokus und Kontext hierbei völlig vernachlässigt werden.

Jacobs beschreibt weiterhin unterschiedliche Konzepte zum Fenstermanagement. Ist die Darstellung in einem einzigen Fenster wenig flexibel und schränkt die Möglichkeiten der Navigation stark ein, so hat die Anwendung des Multi-Fenster-Konzeptes viel versprechende Hilfsmittel zu bieten. Eines davon ist das Anzeigen einer Übersicht, also einer stark verkleinerten Totalansicht des betrachteten Diagramms in einem kleinen Fenster, und einer Arbeitssicht, welche frei zoombar ist, in einem großen Fenster. Dieses Hilfsmittel ist geeignet, den kognitiven Aufwand der Navigation zu verringern. Jedoch geht durch das Übersichtsfenster Platz verloren, der bei ohnehin beschränkter Bildschirmfläche in der Arbeitsansicht dringend gebraucht wird. Abhilfe könnten *Übersichtsschichten* (auch [26]) bieten. Hierbei werden Sichten mit unterschiedlichem Detailgrad übereinander gelegt und überlagern sich so transparent. Somit sind die darunter liegenden, den Kontext darstellenden Sichten noch erkennbar, die Details auf oberster Ebene aber ebenfalls. Problematisch ist die Sicherstellung der

Lesbarkeit von allen Elementen auf allen Ebenen.

Weiteren kognitiven Aufwand, der möglicherweise reduziert werden kann, sieht Jacobs in der Festlegung des Fokus. Damit ein Objekt fokussiert werden kann, muss der Betrachter dies beschließen, das heißt dieser Entscheidung geht das Navigieren zu diesem Objekt, das Identifizieren des Objekts und seiner Relevanz voraus. Neben der Minimierung des Aufwandes für manuelles Fokussieren beschreibt der Autor ebenfalls Methoden zur automatischen Fokussierung von Objekten mit Relevanz. Er sieht besonders bei der Simulation Potential für maschinengetriggertes Fokussieren, etwa bei dem Eintritt eines Fehlers oder speziell bei Statecharts beim Aktivieren eines Zustandes.

4 Ein Ansatz zum Viewmanagement

Aus den in Kapitel 3 erlangten Erkenntnissen gilt es nun, Elemente und Techniken auszuwählen, die dem Ziel des Viewmanagements dienen. Es muss eine Struktur geschaffen werden, die diese Elemente und Techniken unterstützt.

Im folgenden werden die aus oben genanntem Abschnitt extrahierten Anforderungen an das Viewmanagement beschrieben:

Erhöhung der Produktivität durch Verbessern der Pragmatik: Die Anwendbarkeit des KIELER-Tools soll durch den Beitrag des Viewmanagements erleichtert werden, große Diagramme sollen übersichtlicher und überhaupt erst handhabbar werden.

Erhaltung der mentalen Karte: Die mentale Karte ist von großer Wichtigkeit für die Verbesserung der Pragmatik.

Interaktions- und Anpassungsmöglickeiten für den Benutzer: Nach Herman et al. [14], Keim [17] und Storey et al. [38] ist die Anpassung der Visualisierungshilfen an die Vorlieben und individuellen Ansprüche des Benutzers unverzichtbar.

Breiter Einsatzbereich: Die Möglichkeiten des Viewmanagements sollen nicht nur auf einen spezifischen Typ von Graphen anzuwenden sein, sondern auf Diagramme im Generellen.

Unterstützung des gesamten Entwicklungsprozesses: Das Viewmanagement soll die Entwicklung, Betrachtung, Fehleranalyse und auch Simulation unterstützen können. Andere Plugins sollen das Viewmanagment problemlos ansprechen und einbinden können. Es soll einfache und leicht verständliche Methoden zur Anpassung der Techniken und Methoden bieten und ebenso eine Erweiterung des Funktionsumfangs ermöglichen.

Weiterhin wird das Viewmanagement wie auch das ganze KIELER-Projekt in Eclipse entwickelt, es gilt daher die entsprechenden Design-Regeln zu beachten, welche im folgenden als Auszug aufgeführt sind [13]:

- *Contribution*: Das Viewmanagement soll ein Modul sein, aber kein Kern.
- *Invitation*: Die Möglichkeit einer einfachen Erweiterbarkeit durch andere ist Pflicht.
- *Sharing*: Viewmanagement soll KIELER ergänzen, aber nichts ersetzen.
- *Conformance*: Schnittstellen werden korrekt bedient.
- *Lazy Loading*: Nur wirklich benötigte Beiträge sind zu laden.

Während an der Beschreibung von für die visuelle Darstellung von Daten hilfreichen Effekten kein Mangel herrscht, wird oft offen gelassen, wie, von wem und wann diese Effekte ausgelöst werden sollen. Fuhrmann und von Hanxleden [11] schlagen vor, dass das Viewmanagement auf *Trigger* reagieren soll.

Zur weiteren Ausarbeitung dieses Trigger-Begriffs gibt Harel [12] in seinem Paper »STATEMATE: A Working Environment for the Developement of Complex Reactive Systems« wertvolle Hinweise. Er definiert einen Statechart als reaktives System, welches Eingaben von der Umwelt empfängt, diese verarbeitet und dann Ausgaben an die Umwelt abgibt. Auch bei einer SSM werden Wechsel von einem Zustand zum anderen in der Regel durch Signale von außen ausgelöst. Auf Transitionen wird dies in der Notation *Trigger/Effekt* festgehalten. Auch *Frappé*, eine in Java geschriebene Sprache des *Functional Reactive Programming* (FRP) stützt sich auf ein ähnliches Muster [8]. Hier wird von *Behaviours* und *Events* gesprochen, analog zu Effekten und Triggern. Frappé wird noch im Kapitel 6 »Verwandte Arbeiten« ausführlicher diskutiert werden.

Auch beim Viewmanagement für KIELER wird daher ein Trigger-Effekt-Muster implementiert. Trigger lösen *Ereignisse* aus, die dann zu Effekten führen. Jedoch müssen auch noch die in obiger Auflistung erwähnten Anpassungsmöglichkeiten an Anwendungsfall oder persönliche Vorlieben realisiert werden. Das heißt, es muss die Möglichkeit geben, Trigger und Effekte in einem *Kombinations*-Objekt – weitgehend – frei miteinander zu kombinieren und eventuell

noch weitere Bedingungen für einen Effekt vorzugeben. Denkbar wären folgende Szenarien:

- Bei der Simulation eines Statecharts sollen nicht alle aktiven Zustände von Effekten betroffen sein, sondern nur bestimmte Objekte. Eine Kombination könnte eine solche Abfrage realisieren und die Effekte nur auf die gewünschten Objekte zulassen.
- Erst beim mehrmaligen Auftreten eines Triggers sollen Effekte ausgelöst werden.
- Auf einen Trigger sollen mehrfache Effekte folgen.
- Auf einen Trigger sind Berechnungen nötig, bevor Effekte ausgelöst werden sollen.

Es gibt also drei Hauptkomponenten im KIELER Viewmanagement: *Trigger*, die aus der Umwelt vom Viewmanagement registriert werden; sie lösen intern Ereignisse aus. *Kombinationen* empfangen diese Ereignisse, verarbeiten sie und lösen *Effekte* aus, welche visuell sichtbar sind.

Um eine möglichst einheitliche Strukturierung der einzelnen konkreten Trigger, Effekte und Kombinationen zu erhalten und auch um den Aufwand für den Entwickler weiterer Lösungen nach dem Java-Grundsatz der Modularität und Wiederverwendbarkeit gering zu halten, wird möglichst viel von der Struktur und wenn möglich der Funktionalität in einer abstrakten Kombination, einem abstrakten Trigger und Effekt definiert.

4.1 Effekte

In Kapitel 3 sind eine Fülle von Effekten vorgestellt worden. Es stellt sich nun die Frage, welche dieser Effekte den größten Nutzen bringen und bevorzugt zu realisieren sind. Folgt man dem *Information Seeking Mantra* »Overview first, zoom and filter, and then details on demand.« [17] ergibt sich, dass ein Zoom-Effekt nötig ist. Er muss sowohl das gesamte Diagramm darstellen, als auch weit hereinzoomen können, um alle Details sichtbar zu machen.

Weiterhin ist ein Filter-Effekt erforderlich, das heißt, es muss die Möglichkeit gegeben sein, Objekte, die für unwichtig erachtet werden, auszublenden. Eine dynamische Methode, die auch relativ einfach rückgängig zu machen ist, wäre das Kollabieren von unwichtigen Objekten, das heißt das Verbergen von Kind-Objekten, von deren Verbindern, Beschriftungen et cetera. Dies erleichtert dann auch den nächsten Schritt, »details on demand«, bei dem Objekte von Interesse einfach soweit wie gewünscht expandiert werden. Effekte zum Expandieren – *Expand* – und Kollabieren – *Collapse* – sind daher wünschenswert.

Vielfach wird auch das Highlighting als wichtiges Instrument der Kenntlichmachung von wichtigen Objekten und dem Fokus erwähnt [11] [40] [38] [7]. Dieses Highlighting besteht hierbei in der Manipulation der visuellen Darstellung der betroffenen Objekte in erster Linie durch Einfärbung oder Hinzufügen von visuell hervorstechenden Objekten.

Bei der Simulation besteht vielfach der Bedarf, Werte, etwa von Signalen, Eingängen, Daten oder ähnlichem textuell darzustellen und einem Objekt zuzuordnen. Es soll daher einen Effekt geben, der solche vom Trigger übergebenen Werte darstellt – *Textual Representation*.

Weiterhin besteht der Bedarf, unter dem Begriff *Metalayout* [11] das Layouten gewisser Bestandteile des Diagramms anzustoßen. Der Benutzer will hier gegebenenfalls Teile des Diagramms anders layouten als den Rest desselben, um auch so eine Abgrenzung und ein Clustering darzustellen.

Ebenso soll das angesprochene Konzept Zoom und Pan [14] oder *Scrolling* und *Scaling* [16] realisiert werden. Während der dafür nötige Zoom-Effekt schon erwähnt wurde, ist noch ein Scroll-Effekt nötig, bei dem der sichtbare Bildausschnitt an eine bestimmte Stelle verschoben wird.

Unter Berücksichtigung der zu erwartenden Nützlichkeit werden also in der vorliegenden Arbeit vorrangig folgende Effekte realisiert werden:

- Zooming
- Filtering
- Collapse/Expand

- Highlighting
- Textual Representation
- Layout
- Scrolling

Ein Effekt hat offenbar in der Mehrzahl der Fälle (außer bei Zooming) ein Objekt, auf den er sich bezieht. Gegebenenfalls benötigt er Parameter, so zum Beispiel bei Textual Representation den darzustellenden Text, beim Zooming die gewünschte Zoomstufe; andere Effekte können Parameter optional verwenden (Anpassung der grafischen Ausprägung des Highlight-Effekts oder ähnliches). Schließlich benötigt der Effekt einen Ausführungsbefehl, damit seine Darstellung von einer Kombination ausgelöst werden kann. Jeder Effekt muss also mindestens zwei Funktionalitäten beeinhalten:

- Setzen des betroffenen Objekts und damit des Ziels für den Effekt sowie
- Ausführen des Effekts.

4.2 Trigger

Die Auswahl von zu implementierenden Triggern gestaltet sich im Gegensatz zu den Effekten einfacher. Da Trigger von außen angestoßen werden und das Viewmanagement möglichst generisch, das heißt offen für beliebige Systeme mit visueller Darstellung, sein soll, können nur wenige Trigger vorausgesetzt werden und müssen daher im Zweifelsfall von den angrenzenden Projekten geliefert werden.

In jedem Fall ist aber die Interaktion mit dem Benutzer durch Maus- und Tastatur vorgesehen. Er soll zum Beispiel durch einfache Interaktion den Fokus setzen können, indem er ein Objekt auswählt [16] [38]. Ein *Selection Trigger*, der diese Funktionalität implementiert, erscheint daher nötig und sinnvoll.

Analog zum Topcased-Project [7] und in Bezug auf das KIEM-Projekt (siehe Abschnitt 2.6.4) ist ein Simulationstrigger sinnvoll, mit dem die Simulation Effekte

auf relevanten Objekten auslösen kann. Dieser wird jedoch nicht im Viewmanagement implementiert, sondern stößt das Viewmanagement von außen an.

Storey et al. schlagen außerdem eine Art *Chronik* vor, in der Navigationsschritte aufgezeichnet werden sollen, um den Benutzer durch deren Wiedergabe in seiner Orientierung und Erhaltung der mentalen Karte zu unterstützen [38]. Dieses Verfahren könnte einen Chronik-Trigger sinnvoll machen, jedoch ist eine solche Funktionalität in KIELER bisher nur angedacht worden. Nötig wäre hierzu ein Mechanismus, der die Navigationsschritte des Benutzers aufzeichnet und sie bei Bedarf wiedergeben kann. Ein ähnlicher Wiedergabemechanismus wird im Simlations-Plugin (2.6.4) eingesetzt, eventuell ergeben sich hier Möglichkeiten zur Erweiterung des Funktionsumfangs.

Da Trigger Ereignisse anzeigen und weiterleiten sollen, muss die Möglichkeit bestehen, diesen Trigger zu beobachten bzw. sich als Beobachter anzumelden. Ebenso muss eine Abmeldung möglich sein. Bei einem Ereignis sollen alle angemeldeten Beobachter über dieses Ereignis benachrichtigt werden, das betroffene Objekt und eventuell interessante Parameter müssen übermittelt werden.

Zielsetzung dabei ist es, für die Dauer eines Ereignisses einen Effekt auszulösen. Während die Benachrichtigung über ein aufgetretenes Ereignis nicht weiter kompliziert ist, ist die Frage, wie eine Benachrichtigung über das Verschwinden des Ereignisses stattfinden soll, noch zu definieren.

Eine Möglichkeit wäre, permanent in einem gewissen Zyklus abzufragen, ob das Ereignis noch stattfindet, also zum Beispiel der Zustand noch aktiv ist. Dies ist jedoch aufwändig, ineffizient und widerspricht dem Schema des Beobachtens und Benachrichtigens (*Listener Pattern*). Effizient und einfach ist es, beim Verschwinden eines Ereignisses erneut eine Benachrichtigung abzusenden, die der Trigger an alle Beobachter verteilt. In dieser Nachricht muss dann aber zusätzlich enthalten sein, ob der Trigger aktiv geschaltet ist – das Event also aufgetreten ist – oder inaktiv ist – das Event also gerade beendet ist.

Eine durch den Trigger an seine Beobachter versendete Nachricht hat also drei Bestandteile:

- Ist der Trigger aktiv oder inaktiv?

- Welches Objekt ist betroffen?
- Wie sehen die Parameter aus?

Der Trigger selbst muss folgende Funktionalitäten bereithalten:

- Hinzufügen und Entfernen eines Beobachters (*Listener*)
- Benachrichtigen eines Beobachters
- Abschalten des Triggers, sobald er nicht mehr benötigt wird, da er sonst weiterhin Benachrichtigungen sendet

4.3 Verknüpfung der Objekte und Trigger

Die Trigger und Effekte sollen in Kombinationen miteinander verknüpft werden. In den Kombinationen muss also folgendes spezifiziert werden:

- Liste der zu beobachtenden Trigger
- Liste der auszulösenden Effekte
- Auf welche Weise bzw. wodurch werden die Effekte ausgelöst?
- Berechnungen, die vor dem Auslösen der Effekte vorgenommen werden müssen bzw. die Bedingungen, die vor Auslösung erfüllt sein müssen.
- Zustand der Kombination: aktiv oder inaktiv
- An- und Ausschalten der Kombination
- Verhalten beim Auftreten eines Ereignisses

4.4 Verwaltung aller Elemente

Die nach den obigen Vorgaben erstellten Trigger, Effekte und Kombinationen müssen nun zentral verwaltet und erreichbar sein. Es muss eine zentrale Ein-

heit, eine *Run Logic* geben, die den Anstoß für das Viewmanagement an sich gibt und die von anderen Projekten und Plugins benutzt werden kann.

Diese Run Logic muss alle implementierten Trigger, Effekte und Kombinationen einlesen, sie muss verwalten, welche Kombinationen der Benutzer aktiviert haben will und diese dann einschließlich des betroffenen Triggers aktivieren.

Schließlich muss die Run Logic auch die Möglichkeit zur Abschaltung des Viewmanagements bereit halten und ein sauberes Beenden gewährleisten.

4.5 Zusammenfassung

Dieses Kapitel beschreibt die Verarbeitung der in Kapitel 3 gewonnenen Erkenntnisse zur Erstellung eines Lösungsentwurfs. Es werden konkrete Ziele der zu erstellenden Lösung definiert und Grundsätze für die Erstellung dieser Lösung festgelegt. Der Funktionsumfang der einzelnen Komponenten wird abstrakt definiert und ebenso die Verwaltung dieser Komponenten geplant.

5 Viewmanagement in KIELER

»Measuring programming progress by lines of code is like measuring aircraft building progress by weight.« *Bill Gates*

Der praktische Bestandteil der vorliegenden Arbeit besteht in der Entwicklung eines Plugins, welches das KIELER Projekt um den Bestandteil Viewmanagement erweitert. Dieses Kapitel beschreibt die Umsetzung der in Kapitel 3 theoretisch beschriebenen Lösung. Die Implementierung wird, wie in Kapitel 2 beschrieben, in Java erfolgen. Als Entwicklungsplattform und gleichzeitig Zielplattfrom wird *Eclipse 3.5 Galileo* zum Einsatz kommen. Abschnitt 5.1 beschreibt Aufbau und Struktur der Lösung, die Erstellung der grundlegenden Struktur, der einzelnen Bestandteile und ihrer Verknüpfung mit- und Abhängigkeit untereinander.

5.1 Erstellen der Struktur

Ein hervorstechender Vorteil von Eclipse ist der offene und strikt komponentenbasierte Aufbau der Plattform, welcher Eclipse von der Masse der Entwicklungsumgebungen abhebt [13]. Durch Plugins kann die Funktionalität von Eclipse nahezu beliebig erweitert werden. Die Kopplung von Eclipse-Plugins untereinander ebenso wie die Einbindung neuer Plugins erfolgt durch den *Extension Point* Mechanismus.

Normalerweise ist in Java die Vererbung üblich, bei der der Erbe eine Spezialisierung des Erblassers darstellt, der Erbe ist abhängig von Erblasser, ersterer ist aktiv, letzterer passiv. Die Verknüpfung der beiden ist festgelegt und starr, bei Plugins sollte eine Vererbung daher vermieden werden. Auch die Aggrega-

tion oder Komposition folgt dem gleichen Prinzip, wenn auch in der Koppelung loser. Das Problem ist hier jedoch dasselbe.

Der Extension Point Mechanismus implementiert eine Art Registrierungsstelle, an der Instanzen eine Ergänzung des Plugins vermerken lassen können, diese Registrierungsstelle heißt Extension Point. Soll also eine Klasse in einem Plugin ergänzt werden, wird der Extension Point auf registrierte Klassen aus einem anderen Plugin abgefragt. Sind diese vorhanden, wird die Erweiterung ausgeführt. Nutzung eines Plugins und Abhängigkeit sind daher entgegengesetzt. Somit lässt sich ein Plugin ohne Veränderung desselben beliebig erweitern (siehe Abbildung 5.1).

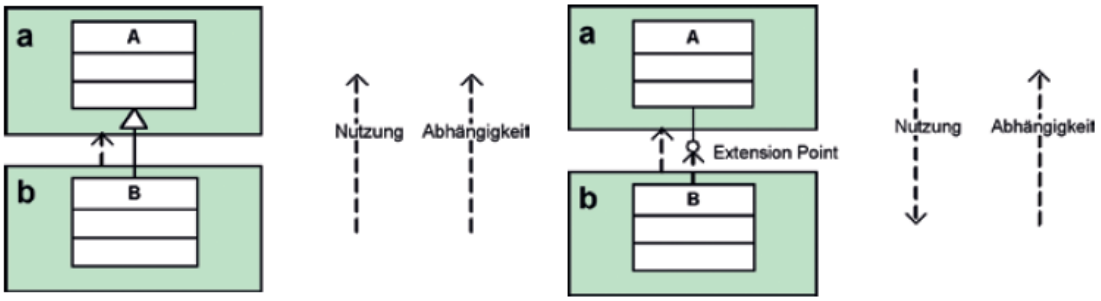

Abbildung 5.1: *Nutzung und Abhängigkeit bei Vererbung (l.) und Extension Points (r.).*

QUELLE: EINFÜHRUNG IN EXTENSION POINTS [13]

5.1.1 Plugins

Bei der Implementierung des Viewmanagements sind drei sinnvolle Java-Plugins aus dem Abschnitt 4 klar ersichtlich: *triggers*, *effects*, *combination*. Weiterhin wird die zentrale RunLogic benötigt, sowie einige andere zentrale Komponenten, wie zum Beispiel ein zu übermittelndes Objekt. Es folgt also die Aufteilung der Implementierung in vier Plugins, die der Namenskonvention von Eclipse und dem KIELER Projekt entsprechend wie folgt heißen:

- de.cau.cs.kieler.viewmanagement
- de.cau.cs.kieler.viewmanagement.combination
- de.cau.cs.kieler.viewmanagement.effects

- de.cau.cs.kieler.viewmanagement.triggers

Das Plugin viewmanagement stellt dabei Extension Points für die drei anderen Plugins combination, effects und triggers zur Verfügung, es ist also das zentrale Plugin. Die jeweiligen Effekte, Trigger und Kombinationen werden als Erweiterungen der jeweiligen Plugins eingebunden. Die für diese Einstellungen nötigen Informationen werden pro Plugin in einer plugin.xml abgelegt, für die Eclipse einen übersichtlichen und benutzerfreundlichen Editor zur Verfügung stellt (Abbildung 5.2).

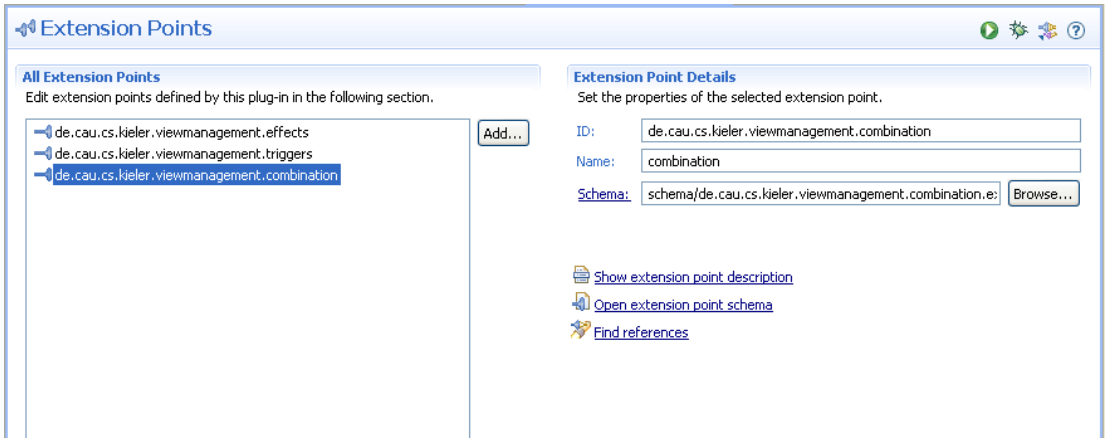

Abbildung 5.2: *Ansicht der plugin.xml Darstellung in Eclipse – Extension Points von viewmanagement*

5.1.2 Viewmanagement

Dieses Plugin ist – wie oben erwähnt – die zentrale Komponente des gesamten Viewmanagements. Hier wird die Struktur, auf die die anderen Plugins zurückgreifen, definiert. Dazu zählt nicht nur die RunLogic, sondern auch, wie in Kapitel 4 beschrieben, die durch die Trigger versendete Nachricht, ein TriggerEventObject und eine abstrakte Beschreibung von Trigger, Effekt und Kombination. Schließlich wird die Implementierung des Listener Patterns hier realisiert. Die abstrakten Klassen werden ATrigger, AEffect und ACombination genannt.

Das TriggerEventObject ist ein speziell für das Viewmanagement erstelltes Objekt, welches bei Auftreten eines Events vom Trigger verschickt wird. Es enthält

ein affectedObject, das Zielobjekt des Events, parameters für zusätzliche Informationen, wie zum Beispiel den darzustellenden Text bei der *TextualRepresentation* und triggerActive, wodurch anzeigt wird, ob das Event gerade auftritt oder beendet wurde. Zusätzlich gibt es für alle drei Objekte *getter* und *setter Methoden* zum Auslesen und Setzen der Parameter.

Die abstrakte Trigger-Klasse ATrigger implementiert die Funktionalität zum Einbinden eines neuen ITriggerListener – einer Erweiterung des generischen EventListener –, der Entfernung eines solchen Listeners und natürlich der Benachrichtigung aller Listener mit einem entsprechenden TriggerEventObject. Schließlich beinhaltet ATrigger eine Methode translateToURI, deren Funktionalität im Abschnitt 5.3 beschrieben wird.

AEffect als abstrakte Effekt-Klasse definiert nur eine abstrakte Methode, nämlich den execute-Befehl für das Auslösen des Effekts. Die konkrete Implementierung muss in den jeweiligen Effekten erfolgen. Weiterhin sind die Methoden setTarget und setParameter in einer Basisimplementierung vorhanden, können jedoch im jeweiligen Effekt überschrieben werden.

Für die Kombinationen stellt ACombination die abstrakte Klasse bereit. Hier werden abstrakte Methoden definiert:

evaluate: Bietet in der konkreten Kombination die Möglichkeit, Berechnungen oder Überprüfungen von Bedingungen vor Ausführung der Effekte auszuführen. Gibt boolean zurück und entscheidet damit über das Ausführen von execute.

execute: Soll ausgeführt werden, wenn evaluate true zurückgibt, realisiert das Erzeugen oder Wiederverwenden der Effekte, Setzen von target, parameters und schließlich Ausführen der Effekte.

getTriggers: Gibt eine Liste der von der Kombination angeforderten Trigger zurück, die beim Initialisieren verarbeitet wird.

initialize: Holt sich die Liste der Trigger, registriert die jeweilige Kombination als Listener.

finalize: Holt sich ebenfalls die Liste der Trigger und entfernt die jeweilige Kombination wieder von der Listener-Liste des Triggers.

Die RunLogic und VMControl sind die zentralen Verwaltungsbestandteile des Viewmanagements. Die RunLogic verwaltet alle anderen Beiträge des Viewmanagements, steuert die Initialisierung der Plugins, ebenso das saubere Beenden. Da es natürlich von dieser zentralen Steuereinheit nur eine Instanz geben darf, um konsistente und eindeutige Zuweisungen zu garantieren, ist bei der RunLogic das Singleton-Pattern implementiert, das heißt externe Plugins rufen die RunLogic mit dem getInstance-Befehl auf, der ihnen entweder die bereits laufende RunLogic zurückgibt oder aber die RunLogic startet.

Beim Start der RunLogic werden durch Aufruf von registerListeners zunächst alle Kombinationen, Trigger und Effekte eingelesen. Wird ein neues Element in einem der drei Plugins erstellt, muss es in der plugin.xml als Erweiterung eingetragen werden, damit Eclipse es findet und benutzen kann. Aus dieser *Registry* werden die Erweiterungen – also Trigger, Effekte usw. – ausgelesen und als Liste der verfügbaren Elemente abgelegt.

Die initialize-Methode der jeweiligen Kombination gibt der RunLogic Strings mit den Bezeichnungen der Trigger, die beobachtet werden sollen; die RunLogic liefert die Instanz des jeweiligen Triggers zurück. Damit wird vermieden, dass für jede Kombination ein separater Trigger erstellt wird, der dann redundant wäre, unnötig Ressourcen verbrauchen würde und separat beendet werden müsste.

Das Beenden des Viewmangement erfolgt durch Aufruf von unregisterListeners, das analog registerListeners die finalize-Methode der aktiven Kombinationen aufruft, die sich dann von den Triggern als Listener abmelden. Schließlich werden auch die Trigger durch Aufruf ihrer finalize-Methode beendet.

VMControl dient als Schnittstelle zum Benutzer. Mit den zugehörigen Klassen TableData, TableDataContentProvider, TableDataEditing, TableDataLabelProvider und TableDataList realisieren sie eine Sicht in Eclipse, die die verfügbaren Kombinationen, die die RunLogic ausgelesen hat, in Form einer Tabelle auflistet und ihr Starten und Beenden ermöglicht. Somit ist für den Benutzer ein einfaches manuelles Steuern des Viewmanagements möglich. Dieser Bestandteil wurde initial von Christian Motika für sein in Abschnitt 2.6.4 vorgestelltes Plugin entwickelt und mit Modifikationen für dieses Plugin übernommen.

5.1.3 Beispiel für eine Trigger-Effekt-Kombination

Als Beispiel für die konkrete Realisation einer Kombination mit Trigger und Effekt wird im Folgenden die SelectionHighlightCombination vorgestellt. Die Aufgabe der Kombination ist es, beim Markieren eines Objektes mit der Maus auf der Zeichenfläche dieses Objekt hervorzuheben, in diesem Fall ein rotes Rechteck um das Objekt zu zeichnen.

Der schematische Ablauf des Viewmanagements ist auch in Abbildung 5.3 dargestellt.

Der Trigger macht sich den *selection service* von Eclipse zunutze, auf dem er sich als Listener registriert (siehe Quelltext 5.1).

```
62  PlatformUI.getWorkbench().getActiveWorkbenchWindow()
63      .getSelectionService().addSelectionListener(this);
```

Quelltext 5.1: *Registrierung als Listener auf SelectionService*

Sobald sich die Markierung auf dem aktiven Fenster der *Workbench* ändert, tritt ein entsprechendes Event auf und es wird die dazugehörige Methode des Triggers aufgerufen (siehe Abbildung 5.3, *Event1 happens*). Dies ist die Funktionalität, die vom selection service bereitgestellt wird. Von Interesse ist die vom selection service zurückgegebene selection. Aus dieser selection wird im Trigger nach Sicherheitsabfragen und Umwandlung in das benötigte Format ein neues TriggerEventObject erstellt. Es wird weiterhin geprüft, ob bereits ein Objekt markiert war, in diesem Fall wird eine entsprechende Invalidierung des Events gesendet, damit die bestehende Markierung entfernt wird. Anschließend oder – wenn noch keine Markierung vorhanden war – unmittelbar wird die momentane Markierung für die spätere Aufhebung des Events gespeichert. (siehe Quelltext 5.2)

Es wird das TriggerEventObject mit dem Objekt aus der selection bestückt, weiterhin wird ein Boolscher Wert gesetzt, um das Auftreten eines neuen Events anzuzeigen und schließlich die Benachrichtigungsmethode mit dem so erstellten TriggerEventObject aufgerufen. Dies bewirkt, dass alle auf diesem SelectionTrigger registrierten Listener mit dem TriggerEventObject benachrichtigt werden.

Durch Anwendung dieses Listener-Schemas ist eine Verteilung an alle »Interessenten« (Listener) gewährleistet, aber ohne dass diese aktiv Abfragen nach neuen Events tätigen müssen – dies wäre ressourcenintensiv und würde eine unerwünschte zeitliche Verzögerung zwischen Auftreten des Events und dessen Registrierung bedeuten.

```
75  public void selectionChanged(IWorkbenchPart part, ISelection selection) {
76
77      if ((selection instanceof IStructuredSelection) && !(selection instanceof TreeSelection)) {
78          this.selectionEvent = new TriggerEventObject();
79          List<?> selectionList = ((IStructuredSelection)selection).toList();
80          Object selectedObject;
81
82                                  selectedObject = selectionList.get(0);
83
84          if (selectedObject instanceof EditPart) {
85              if (currentSelection != null) {
86                  selectionEvent.setTriggerActive(false);
87                  selectionEvent.setAffectedObject(translateToURI(currentSelection));
88                  selectionEvent.setParameters("Test");
89                  notifyTrigger(selectionEvent);
90
91              }
92              currentSelection = selectedObject;
93              selectionEvent.setAffectedObject(translateToURI(selectedObject));
94              selectionEvent.setParameters("Test");
95              selectionEvent.setTriggerActive(true);
96              notifyTrigger(selectionEvent);
```

Quelltext 5.2: *selectionChanged-Methode*

Die RunLogic hat die SelectionHighlightCombination während der Initialisierung als Listener auf dem SelectionTrigger registriert (siehe Abbildung 5.3, oberes Drittel). Ein SelectionEvent sorgt also dafür, dass die Benachrichtigungs-Methode dieser Kombination aufgerufen wird.

Dadurch wird innerhalb der Kombination eine Auswertungsmethode angestoßen (evaluate()). Diese liest aus dem vom Trigger übergebenen Wert das betroffene Objekt sowie eventuell gesetzte Parameter und den Zustand des Events – aktiv oder inaktiv – aus. Hier bietet sich ebenfalls die Möglichkeit, weitere Überprüfungen durchzuführen, die die Ausführung der Kombination blockieren oder zulassen. Wird die Ausführung zugelassen, erzeugt die execute-Methode – sofern noch nicht vorhanden – einen neuen HighlightEffect, setzt das Ziel des

Effekts und dessen Parameter entsprechend und führt den Effekt aus.

Der HighlightEffect wird initialisiert, bei Aufruf der Ausführungsmethode (execute) holt sie sich den Zeichenlayer, auf dem der Effekt dargestellt werden soll – hier den PAGE_BREAKS_LAYER. Zur Auswahl des Layers sei auf Abschnitt 5.2.1 verwiesen, wo das Thema näher behandelt wird.

Die Parameter des zu zeichnenden Rechtecks werden gesetzt; diese können bei jeder Ausführung des Effekts geändert werden. Die Position des Effekts wird berechnet und schließlich wird das so konfigurierte Rechteck dem *Layer* hinzugefügt und das Zeichnen des Effekts angestoßen – der Effekt somit sichtbar gemacht.

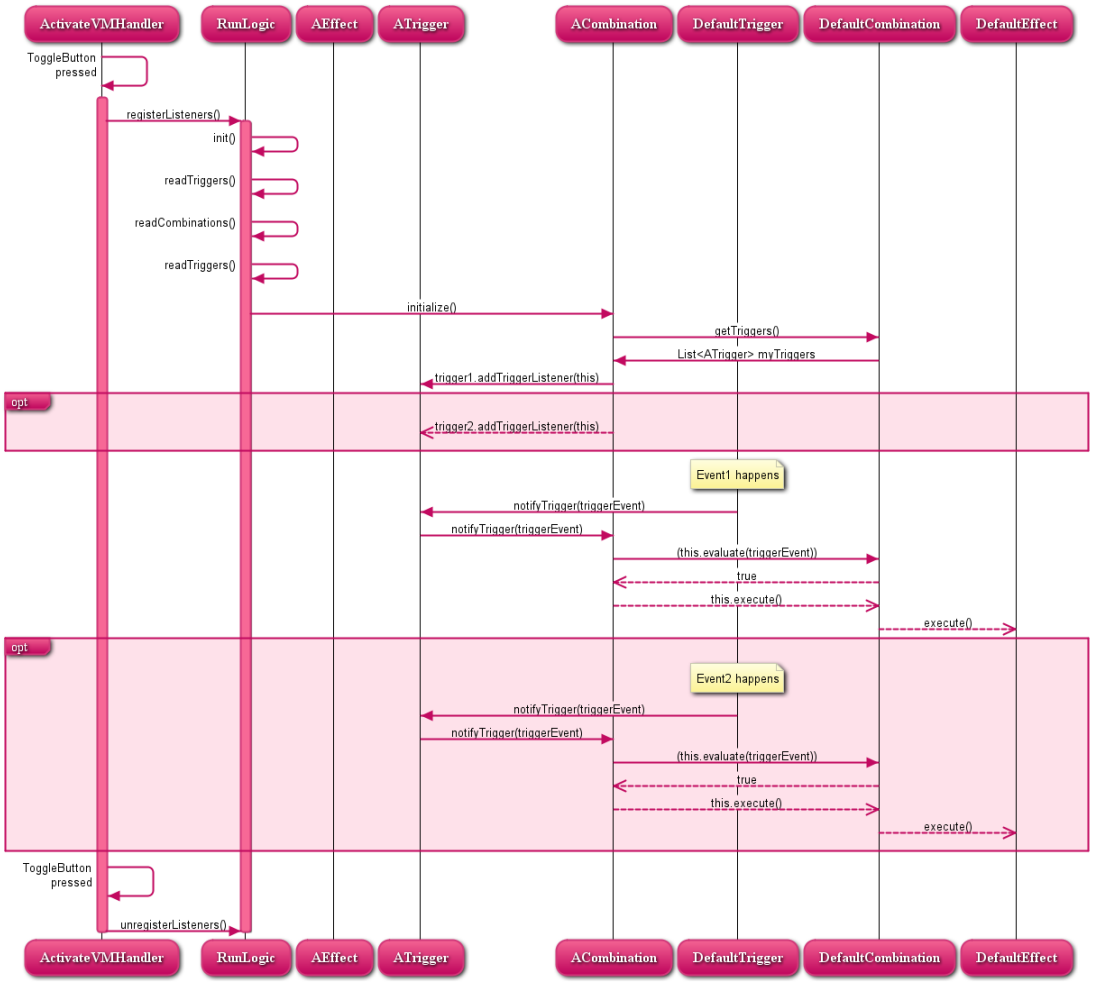

Abbildung 5.3: *Schematischer Ablauf des Viewmanagements*

5.2 Besondere Herausforderungen und Überlegungen

Bei Anfertigung der vorliegenden Arbeit traten eine Reihe von speziellen und erwähnenswerten Problemen auf, die betrachtet werden mussten und für die besondere Lösungen geschaffen wurden. Diese Lösungen sollen im Folgenden vorgestellt werden.

5.2.1 Layer

Eine spezielle Fragestellung, die es insbesondere bei der Erstellung des Highlight-Effekts zu klären galt, war die Auswahl eines geeigneten Zeichenlayers.

Layer [22] sind transparente *Figuren*, die benutzt werden, um unterschiedliche Funktionalitäten bereit zu halten. Diese Figuren werden in sogenannten *LayerPanes* verwaltet. Dies sind Figuren, die Layer beinhalten. LayerPanes speichern die beinhalteten Layers in einer Map mit einem String als Key. Um das *HitTesting* von Draw2D nicht zu behindern, werden die entsprechenden Methoden (findFigureAt() usw.) »durchgeschleift«, sodass das HitTesting auf der rootFigure stattfindet.

Das HitTesting ist eine Abfrage, ob sich ein Objekt an einem bestimmten Punkt befindet, es wird also gewissermaßen auf einen Punkt »geschossen«, alle »getroffenen« Objekte werden zurückgegeben. Da die Layer aber nicht als wirkliche Figur auftreten sollen, sondern für den Benutzer und Entwickler mehr oder weniger transparent bleiben sollen, findet hier besagtes »Durchschleifen« statt.

Es gibt zwei Subklassen zu LayerPanes (Abbildung 5.4): FreeformLayeredPane und ScalableFreeformLayeredPane. FreeformLayeredPane enthält eine Menge von beliebig in alle Richtungen erweiterbaren Layer. Die in ScalableFreeformLayeredPane enthaltenen Layer unterstützen zusätzlich Zooming. ScalableLayeredPane ist ein LayerPane, welches zwar Zooming unterstützt, aber eine endliche, fixe Größe hat.

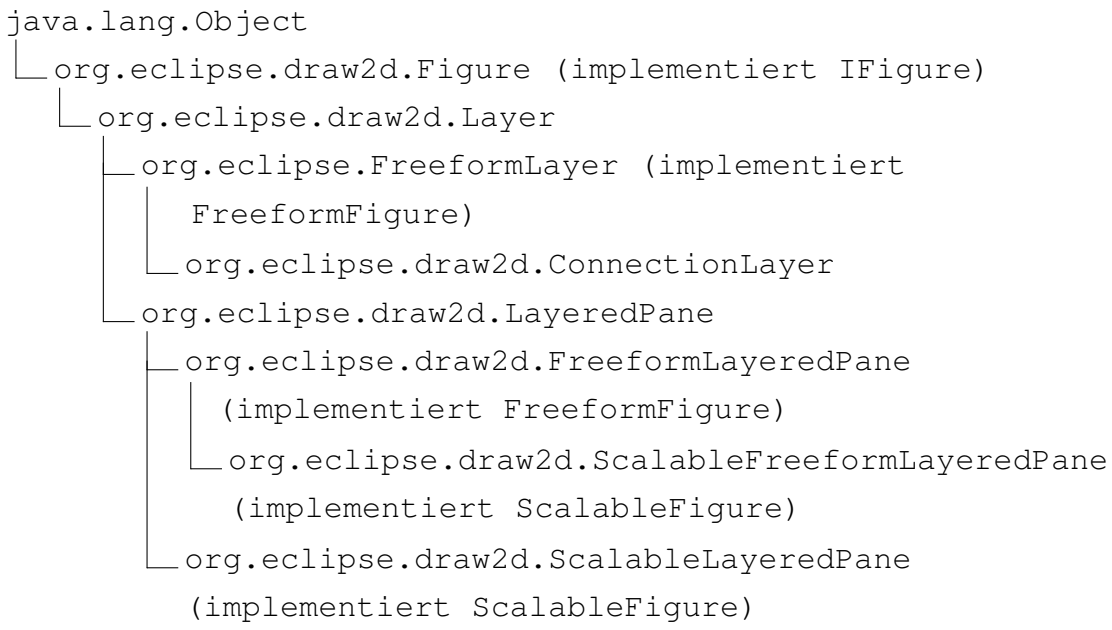

Abbildung 5.4: *Klassenhierarchie der Layer-Figuren*

Vorgefertigte Layer sind :

- FEEDBACK_LAYER: Stellt Popup-Icons dar, die schnelle Benutzeraktionen ermöglichen, zum Beispiel Hinzufügen von Objekten.
- CONNECTION_LAYER: Stellt die Verbinder zwischen Objekten dar.
- DECORATION_PRINTABLE_LAYER: Stellt die druckbaren Dekorationen dar, wie zum Beispiel bei Icons (*modified, conflicted,...*) oder Text und Labeln.
- DECORATION_UNPRINTABLE_LAYER: Wie vor, nur für nicht druckbare Dekorationen.
- GRID_LAYER: Stellt ein Gitternetz im Editor dar.
- HANDLE_LAYER: Stellt Markierungen von Objekten dar.
- PRIMARY_LAYER: Basis-Layer.
- PAGE_BREAKS_LAYER: Berechnet Seitenumbrüche zum Drucken der Ansicht.
- SCALED_FEEDBACK_LAYER: Stellt skalierte Feedback-Objekte dar.

Hierbei sind beispielsweise in der Gruppe SCALABLE_LAYERS die folgenden Objekte:

- FreeformLayer
- FreeformLayeredPane
- GridLayerEx
- FreeformLayer

Die Struktur der Layer (aber nicht deren Anordnung) stellt sich wie folgt dar[1]:

```
Root Freeform Layered Pane
 ├─Guide Layer
 ├─Feedback Layer
 └─Scalable Layers (ScalableFreeformLayeredPane)
    ├─Scaled Feedback Layer
    ├─Printable Layers
    │  ├─Connection Layer
    │  └─Primary Layer
    └─Grid Layer
```

Abbildung 5.5: *Struktur der Layer*

Bei der Auswahl des Layers für das Zeichnen des Highlight-Effekts wurden einige Eigenschaften gefordert. Der Layer sollte standardmäßig sichtbar sein, somit schied etwa der GRID_LAYER aus, da dieser nur bei eingeschaltetem Gitternetz sichtbar ist. Weiterhin sollten die auf dem Layer gezeichneten Effekte mit den so hervorgehobenen Figuren zoombar sein, die Größe des Highlight-Effekts stets zum hervorgehobenen Objekt passen. Der FEEDBACK_LAYER etwa stellte sich als nicht zoombar heraus, auch wenn er vom Aufgabenfeld her zu den Effekten gepasst hätte – auf dem FEEDBACK_LAYER werden Hilfsobjekte, wie etwa Popup-Icons für das schnelle Hinzufügen von Objekten dargestellt. Die Annahme, dass der SCALED_FEEDBACK_LAYER dieses Manko beheben könnte, bestätigte sich nicht, hier ließen sich keine Objekte darstellen. Ebenso schied

[1] http://help.eclipse.org/help32/topic/org.eclipse.gef.doc.isv/reference/api/org/eclipse/gef/editparts/ScalableFreeformRootEditPart.html

der HANDLE_LAYER aus – er ist für die Darstellung der Markierungsrahmen an Objekten zuständig.

Der CONNECTION_LAYER stellt hier eine Besonderheit dar, er soll die Verbindungen oder Transitionen zwischen den Objekten darstellen und wird als alleroberster Layer gezeichnet.

Letztendlich fiel die Wahl auf den PAGE_BREAKS_LAYER, der eigentlich die Seitenumbrüche beim Drucken verwaltet, also errechnet, wie viele Zeilen und Spalten auf eine Druckseite passen und das zu druckende Dokument entsprechend in druckbare Seiten umbricht. Er ist zoombar und stellt die auf ihm gezeichneten Objekte jederzeit standardmäßig dar.

Eventuell wäre für die Weiterentwicklung des Viewmanagements die Implementierung eines eigenen Layers speziell für das KIELER-Projekt denkbar, jedoch gibt es vom jetzigen Standpunkt dafür keine unmittelbare Notwendigkeit.

5.2.2 Positionierung der Objekte

Bei der Positionierung der Objekte stellte sich eine weitere Herausforderung. Ziel war es, die Effekte – insbesondere den Highlight-Effekt – exakt an der Position des hervorzuhebenden Objekts zu platzieren. Hierbei musste eine Besonderheit des Zeichensystems Draw2D berücksichtigt werden. Draw2D gibt die Positionsangaben nicht in absoluten Werten auf der Zeichenfläche an, sondern in Relation zum übergeordneten Eltern-Objekt. Standardmäßig stellt Eclipse eine Umrechnungsmethode translateToAbsolute zur Verfügung, welche die Hierarchie der Objekte durchläuft, deren relative Positionsangaben auswertet und absolute Koordinaten zurückgibt.

Bei Tests stellte sich jedoch heraus, dass translateToAbsolute weder mit Zoomen noch Scrollen auf der Zeichenfläche zurechtkommt, das heißt es ergaben sich bei einem anderen Zoomwert als 100% oder beim Scrollen auf der Zeichenfläche vom oberen linken Anschlag weg fehlerhafte Berechungen. Es waren nach Umrechnung auf absolute Koordinaten mittels translateToAbsolute also Korrekturberechnungen für Zoomwert und Scrollposition nötig.

Während der Zoomwert relativ einfach vom ZoomManager abgefragt werden konnte, gestaltete sich die Berechnung der Scrollwerte schwieriger.

Grund dafür war, dass nicht nur Scroll-Möglichkeiten auf der Hauptzeichenfläche existieren, sondern auch einzelne Knoten-Objekte bei Bedarf scrollen können. Die Scroll-Funktionalität wird in Eclipse durch ScrollPane zur Verfügung gestellt, Viewport ist ein flexibles Fenster auf einem ScrollPane, das den momentan sichtbaren Bereich darstellt. Von einem Viewport kann man die momentane Scrollposition abfragen und entsprechende Korrekturen für die translateToAbsolute-Funktion errechnen. Da es – wie oben erwähnt – mehrere solcher ScrollPane geben kann, muss die Hierarchie durchsucht werden und alle über dem betroffenen Objekt angesiedelten Viewport nach ihrem Scrollwert befragt werden.

5.3 Adressierung der Objekte

Die Adressierung der Objekte stellte eine weitere, grundlegende Entscheidung dar. Während die Effekte in der Regel EditParts oder genauer ShapeEditParts als Argumente verlangen und auch der SelectionTrigger entsprechende der selektierten Objekte auf der Zeichenfläche EditParts zurückgibt, existiert von seiten anderer Plugins – insbesondere der Simulation im KIEM-Projekt – die Notwendigkeit einer anderen Art der Adressierung. Aber auch im Hinblick auf die gewünschte Erweiterbarkeit und die Möglichkeit der Einbindung quasi beliebiger anderer Plugins mit dem Thema der visuellen Modellierung erscheint eine möglichst einfache und allgemein gültige Form der Adressierung erstrebenswert.

Anforderungen an die Adressierung waren:

Eindeutigkeit: Selbstverständlich sollte jedes Objekt eindeutig adressierbar sein.

Dauerhaftigkeit: Die Adressierung sollte unabhängig von der Laufzeit des Programmes persistent sein, das heißt die Adresse auch in der nächsten Instanz des Programmes gültig und korrekt sein.

Bestimmter Typ: Da das KIEM-Plugin mittels *JSON*-Objekten (JavaScript Object Notation) kommuniziert, muss die Adressierung in Form eines Strings

übertragen werden.

Lesbarkeit: Auch wenn eine menschenlesbare Adressierung in diesem Kontext nicht unbedingt Pflicht ist, so erleichtert sie doch in einigen Fällen Fehlersuche und Verständnis. Dies gewinnt an Bedeutung, wenn das Viewmanagement von der grafischen Darstellung losgelöst angesteuert werden soll.

Idealerweise sollte die Adressierung auch die Möglichkeit beinhalten, ausgehend von einem Element andere zu adressieren, etwa alle direkt erreichbaren Elemente oder in der Hierarchie die Elemente auf selber Ebene, Kind-Elemente oder ähnliches. Hierdurch wären einfache Beschreibungen von Verhaltensweisen einer Kombination möglich, zum Beispiel in einem Statechart das Hervorheben aller ausgehenden Transitionen, um deren Prüfung deutlich zu machen. Auch die Clusterbildung wäre so einfach möglich.

Die Wahl fiel auf die Adressierung durch *URIFragments*. Diese werden auf Modellebene von den EObjects bereitgestellt, sind eindeutig und entsprechend in der das grafische Modell repräsentierenden textuellen Datei – der Ressource – aufzufinden. Diese Datei liegt im XML-Format vor und ist somit für Menschen lesbar.

Der Vorschlag für diese Adressierungsmethode sowie die im Folgenden beschriebene Übersetzungsmethode kam von Christian Motika und wurde nicht nur für sein Projekt KIEM und das Viewmanagement sondern auch für KSBasE von Michael Matzen übernommen. Dies war notwendig um eine konsistente Adressierungsmethode innerhalb des KIELER-Projekts zu gewährleisten.

Um eine einfache Übersetzung zu gewährleisten, wurde in der ATrigger-Klasse eine Methode translateToURI implementiert, die einen EditPart in ein URIFragment übersetzt. Ein beliebiger Trigger, der standardmäßig EditParts zurückgibt, kann also diese Übersetzungsmethode aufrufen und das TriggerEventObject mit einem String als affectedObject versehen.

Für die Rückübersetzung zu einem EditPart – welchen die Mehrzahl der Effekte als Argument benötigen – wurde eine Methode translateToEditPart generiert. Diese Methode durchläuft ausgehend von einem Eltern-Element alle Objekte des Diagramms und vergleicht das URIFragment, welches als Argument mitgegeben wurde, mit dem jeweiligen URIFragment des gerade überprüften

Objekts. Bereits gefundene Übereinstimmungen werden zur Verbesserung der Such-Effizienz gecached, bei erneutem Übersetzen sind sie wesentlich schneller auffindbar.

Das Eltern-Element wird entweder durch das zweite Argument der Methode spezifiziert oder – sollte dieses nicht gesetzt sein – liest die Methode den RootEditPart als Eltern-Element aus dem aktiven Editor aus.

Rückgabewert ist also der EditPart, der dem im Argument spezifizierten URIFragment zugeordnet ist.

5.4 Realisierte Effekte

In diesem Abschnitt werden die im KIELER Viewmanagement realisierten Effekte, ihre Vorteile und Anwendungsbreiche beschrieben, auf Besonderheiten bei der Implementierung wird hingewiesen.

5.4.1 Highlight-Effekt

Der Highlight-Effekt ist – wie bereits erwähnt – einer der wichtigsten aber auch einfachsten Effekte. Er soll durch Hinzufügen von grafischen Informationen das so markierte Objekt aus der Masse der anderen »heraussspringen« lassen.

Die simpelste Ausprägung dieses Effekts ist das Zeichnen eines farbigen – in diesem Fall – roten Rahmens um das Objekt. Zur Platzierung des Rahmens werden die Koordinaten des betroffenen Objekts herangezogen.

Bei Erstellung eines neuen Highlight-Effekts wird ein Rechteck erstellt, entsprechende Parameter, die jeder Highlight-Effekt haben soll – nicht gefüllt, durchgehende Linie et cetera – werden gesetzt, ebenso eine Standardfarbe und -linienstärke.

Der execute-Befehl des Effekts sucht sich zunächst den *RootEditpart* des hervorzuhebenden Objekts. Von ihm wird der entsprechende Zeichenlayer der PA-

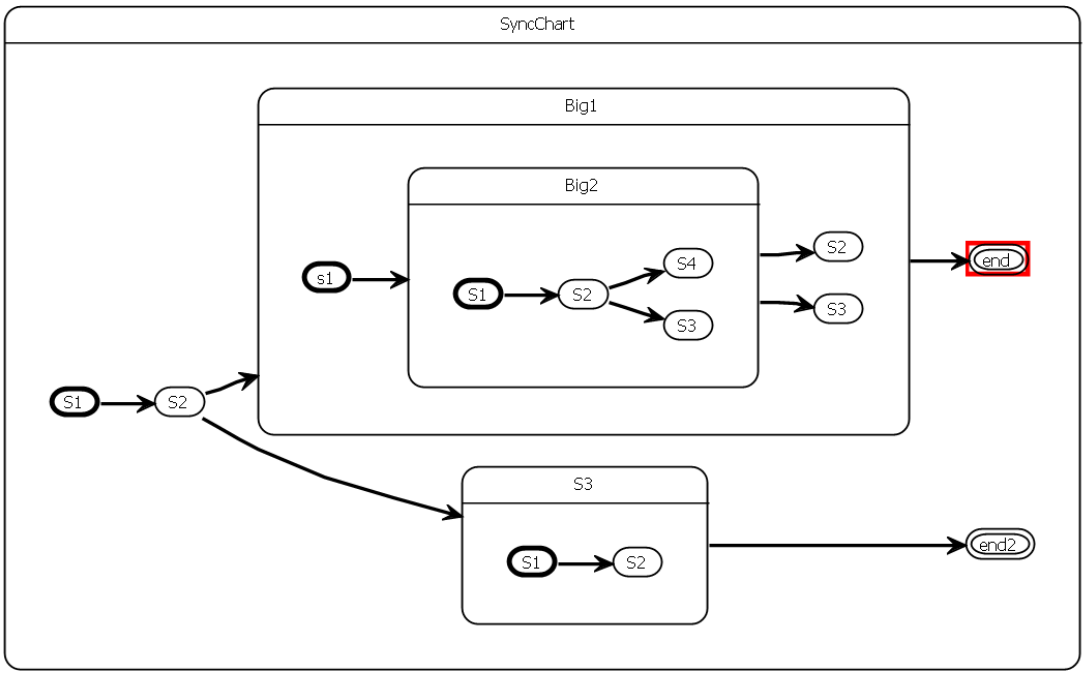

Abbildung 5.6: *Statechart mit Hervorhebung durch Highlight-Effekt*

GE_BREAKS_LAYER abgefragt – wie vorher in diesem Kapitel beschrieben. Die gesetzte Linienfarbe und -stärke wird erneut abgefragt.

Von der Figure des hervorzuhebenden Objekts werden dessen Koordinaten abgefragt, sie sind jedoch – wie weiter oben schon beschrieben – relativ und müssen in absolute Koordinaten übersetzt werden. Zusätzlich müssen die – ebenfalls oben erwähnten – korrigierenden Berechnungen für Zoom- und Scrollwert angewandt werden.

Ist dies geschehen, werden die Koordinaten des Highlight-Objekts gesetzt, das Objekt dem Zeichen-Layer hinzugefügt und eine Aktualisierung desselben angestoßen.

Der Highlight-Effekt verfügt über eine undo-Methode, die den letzten Effekt entfernt, weiterhin eine setHighlightFigure-Methode, bei der die Werte für Linienfarbe und -stärke geändert werden können. Die execute-Methode fragt diese Werte bei jedem Aufruf ab, sodass jeder Effekt anders aussehen kann.

5.4.2 Filter-Effekt/UnFilter-Effekt

Der Filter- und UnFilter-Effekt sind kaum weniger wichtig als der Highlighteffekt, da sie unmittelbaren und wirksamen Einfluss auf die Informationsdichte im Diagramm haben. Jedoch sind sie noch einfacher in ihrer Funktion. Betroffene Objekte werden ganz einfach versteckt oder wieder sichtbar gemacht.

Die execute-Methode umfasst daher lediglich das Auslesen der Figure des betroffenen Objekts, eine Abfrage, ob es schon sichtbar beziehungsweise unsichtbar ist und schließlich das Setzen des Parameters, um den gewünschten Effekt zu erreichen.

5.4.3 CompartmentCollapse-Effekt und CompartmentExpand-Effekt

Dieser Effekt ist von großer Bedeutung für die Regulation der Informationsdichte durch das Viewmanagement.

In einem Diagramm sind Informationen oft hierarchisch abgelegt und in sogenannte *Compartments* verpackt – quasi Schubfächer, die durch Klicken mit der Maus auf ein entsprechendes Symbol ein- und ausgeklappt werden können. Eclipse bietet diese Funktionalität sowohl bei den bereits vorhandenen Editoren an als auch zur Implementierung in selbst erstellte Editoren. Auch in KIELER und dem ThinKCharts-Editor kommt diese Technik verstärkt zu Einsatz.

Das Ein- und Ausklappen ermöglicht ein einfaches Ausblenden von momentan nicht benötigten Informationen und somit eine einfache Beeinflussung des Detailgrades. Durch Anwendung von Layout nach Einklappen von Compartments kann oftmals eine Kompaktierung des betroffenen Objekts erreicht werden, da es nicht mehr so viele untergeordnete Objekte beinhalten muss.

Ausgehend vom betroffenen Objekt ruft der CompartmentCollapseEffect in seiner execute-Methode eine Hilfsmethode auf, die die Kinder des betroffenen Objekts daraufhin abprüft, ob sie eine Instanz von ResizeableCompartmentFigure sind. Ist dies der Fall, werden sie in einem Feld gespeichert. Nun besteht die

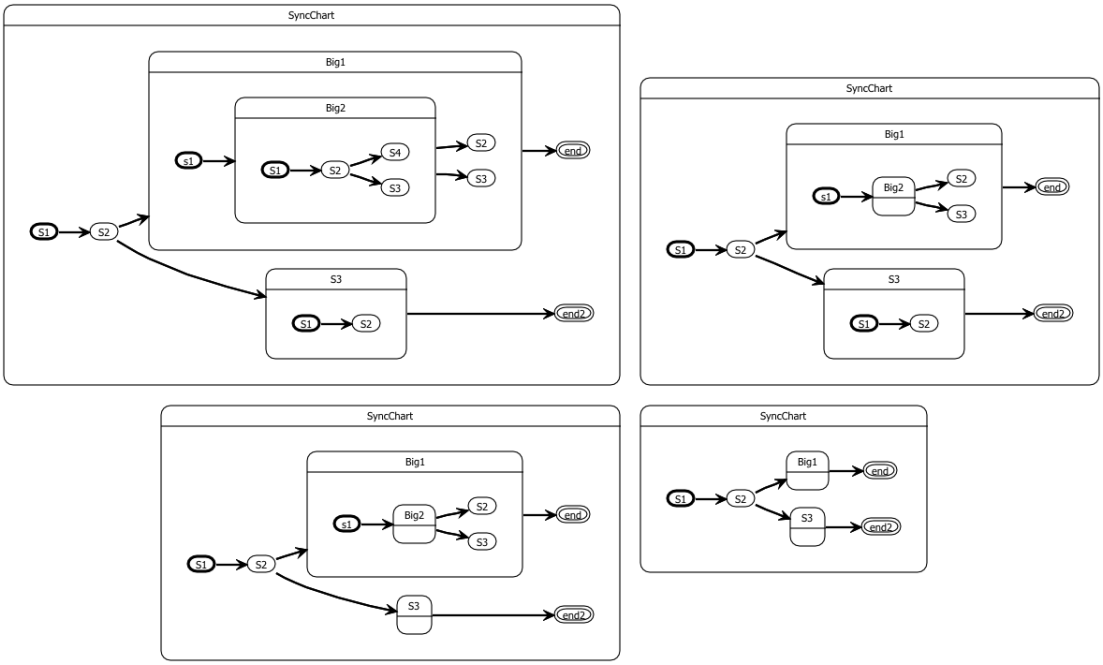

Abbildung 5.7: *Statechart mit Collapse-Effekt – Stufenweise Verflachung der Hierarchie*

Möglichkeit alle oder nur einige der Objekte in diesem Feld einzuklappen, standardmäßig jedoch werden alle Objekte eingeklappt. Dies geschieht durch Aufruf der collapse-Methode der entsprechenden ResizeableCompartmentFigure.

Das Einklappen der Compartments wird von Eclipse standardmäßig animiert und ist somit für den Benutzer leicht nachzuvollziehen.

Analog existiert eine expand-Methode, die eine ResizeableCompartmentFigure wieder aufklappt. Sie kommt beim CompartmentExpand-Effekt zum Einsatz.

5.4.4 Layout-Effekt

Der Layout-Effekt bedient sich hauptsächlich der Mechanismen vom KIML (2.6.3) indem es aus der Klasse DiagramLayoutManager die Methode layout aufruft. Als Argumente kann ein betroffenes Objekt und der entsprechende Editor dazu mitgegeben werden, die Schalter für Animation und einen Fortschrittsbalken sind nicht gesetzt. Standardmäßig jedoch wird das Layout auf das gesamte Diagramm ausgeführt. KIML entscheidet selbsttätig über den anzuwendenden

Layout-Algorithmus und führt das Layout aus.

5.4.5 TextualRepresentationEffect

Dieser Effekt soll textuelle Informationen jeglicher Art an einem Objekt platzieren und anzeigen. Denkbare Einsatzgebiete sind die Anzeige von Simulationsdaten (Signalwerten, Fehlermeldungen et cetera), erklärende Hinweise usw.

Die Realisation des Effekts ist ähnlich der des HighlightEffekts, nur dass statt einem Rechteck eine textualFigure erzeugt wird.

Zeichenlayer, Berechnung der Koordinaten des Objekts, Umrechnungen und Korrekturberechnungen sind analog. Die anzuzeigende Information muss vom Trigger mittels des parameter-Objekts übergeben werden. Der Trigger setzt die Parameter auf den anzuzeigenden String, der Effekt liest die Parameter wieder aus und verwertet sie als anzuzeigenden Text in seiner execute-Methode.

Verbesserungswürdig ist hierbei die Platzierung des Effekts – dieser erscheint momentan noch zentriert auf dem betroffenen Objekt. Es sollte berücksichtigt werden, wie der Effekt die bestmögliche Lesbarkeit erreichen kann, aber dennoch wenig Platz belegt. Eine eindeutige visuelle Zuordnung zum verknüpften Objekt sollte jederzeit und zweifelsfrei möglich sein.

5.4.6 Zoom-Effekt

Der Zoom-Effekt hat das Ziel, den Zoomwert des Editors so zu setzen, dass das gesamte Diagramm angezeigt wird. Gemeinhin wird dies als *Zoom To Fit* bezeichnet.

Die execute-Methode liest hierbei zunächst den aktiven Editor aus, prüft, ob dies eine Instanz des DiagramEditors ist und liest davon wiederum den entsprechenden EditPart aus. Von dessen RootEditPart wird der ZoomManager ausgelesen. Der Zoom Manager ist verantwortlich für die Vorgehensweise beim Ändern des Zoomlevels, er kontrolliert eine ScaleableFigure, die die tatsächliche Veränderung des Zooms darstellt und einen ViewPort, der eventuell beim Zoomen

nötig werdendes Scrollen verarbeitet. Der Zoom Manager verarbeitet nicht nur numerische Werte für den Zoom, sondern berechnet auch bei Eingabe von textuellen Konstanten den nötigen Zoomwert. So wird in diesem Fall die FIT_ALL-Konstante als Zoomwert gesetzt – daraufhin berechnet der Zoom Manager den nötigen Zoomwert, um das ganze Diagramm auf der Zeichenfläche abbilden zu können.

Der Zoom-Effekt ist neben dem Layout-Effekt der einzige Effekt, der ohne betroffenes Objekt auskommt. Um jedoch nicht wegen eines Effekts die Struktur ändern zu müssen, wird in diesem speziellen Fall das betroffene Objekt einfach nicht weiterverarbeitet.

5.4.7 ZoomAndScrollTo-Effekt

Der ZoomAndScrollTo-Effekt soll eine für Fokus und Kontext sehr wichtige Funktionalität realisieren. Bei Fokus und Kontext wird – wie der Name bereits andeutet – der Fokus auf ein bestimmtes Objekt gesetzt, jedoch soll das Objekt nicht allein und isoliert im Fokus stehen, da dies ein Erkennen des Zusammenhangs stark erschwert und auch die Erhaltung der mentalen Karte großer Anstrengungen bedarf.

Abhilfe bietet das Prinzip, neben dem Objekt im Fokus auch ein gewisses Maß an Kontext anzuzeigen, um Rückschlüsse auf die Umgebung und die Funktionalität der umgebenden Objekte zu ermöglichen. Dieses Verhalten wird als *Zoom and Scroll* oder *Zoom and Pan* bezeichnet, im implementierten Effekt wurde die erstere Bezeichnung gewählt.

Die Implementierung der execute-Methode führt zuerst das Zoomen und dann das Scrollen aus. Der Zoomwert wird anhand der Höhe oder Breite des betroffenen Objekts berechnet, dazu wird von den Eltern des betroffenen Objekts der Viewport, der an oberster Stelle in der Objekthierarchie steht – also für Scrollen et cetera auf Editorebene verantwortlich ist – gesucht. Von ihm wird die Größe abgefragt, die auch der Größe des sichtbaren Fensters entspricht. Anhand dieser Werte wird der theoretische Zoomwert errechnet, bei dem das Objekt den sichtbaren Bereich des Editorfensters voll ausfüllen würde – theoretisch deshalb,

weil Eclipse standardmäßig keine Zoomwerte größer 400% zulässt.

Bei der Berechnung dieses Wertes wird auf die Objektgröße ein Korrekturwert aufgeschlagen. Er bestimmt, wieviel von der Umgebung um das Objekt, also vom Kontext, sichtbar ist. Wird der Wert auf Null gesetzt, ist – bis zum Erreichen des maximalen Zoomwerts – nur das jeweilige Objekt sichtbar.

Der so berechnete Zoomwert wird gesetzt. Nun muss noch der Scrollwert so gesetzt werden, dass das betroffene Objekt auch im Fokus steht. Dazu wird analog zum Highlight-Effekt eine absolute Positionsberechnung des betroffenen Objekts durchgeführt, inklusive Korrekturberechnungen für Scroll- und Zoomwerte. Schließlich werden die Werte für horizontales und vertikales Scrollen des Viewports auf die Positionswerte des betroffenen Objekts gesetzt, es rückt damit in den Fokus.

5.4.8 ShapeHighlight-Effekt

Der ShapeHighlight-Effekt hat denselben prinzipiellen Ansatz wie der Highlight-Effekt, hat jedoch bei der Ausprägung einen anderen Ansatz. Fügt der Highlight-Effekt der normalen Darstellung eines Objekts zusätzliche Objekte hinzu, um die Hervorhebung zu erreichen, manipuliert der ShapeHighlight-Effekt das Objekt selbst – etwa durch Änderung der Linienfarbe und -stärke der Objektbegrenzung. Vorteilhaft bei dieser Vorgehensweise ist die Anpassung an die Form des betroffenen Objekts, der Effekt nimmt keinerlei zusätzlichen Platz ein und kann zu optisch gefälligeren Ergebnissen führen.

Nachteilig ist die technisch anspruchsvollere Umsetzung. Das betroffene Objekt muss Methoden zur Änderung der erwähnten Parameter bereithalten, die der Effekt dann aufrufen kann. Da dies auch nicht bei allen Objekten und Editoren gleichermaßen vonstatten geht, ist eine individuelle Anpassung an den speziellen Einsatzort des Effekts nötig. Beispielsweise unterstützte der ThinKCharts-Editor diesen Effekt auf Knoten nicht, die Farbgebung und Linienstärke sowie andere für diesen Effekt interessante Parameter waren hart codiert – Änderungen wurden augenblicklich überschrieben. Es waren Änderungen im Editor nötig um entsprechende Methoden zum Ändern der Farben bereitzustellen. Die

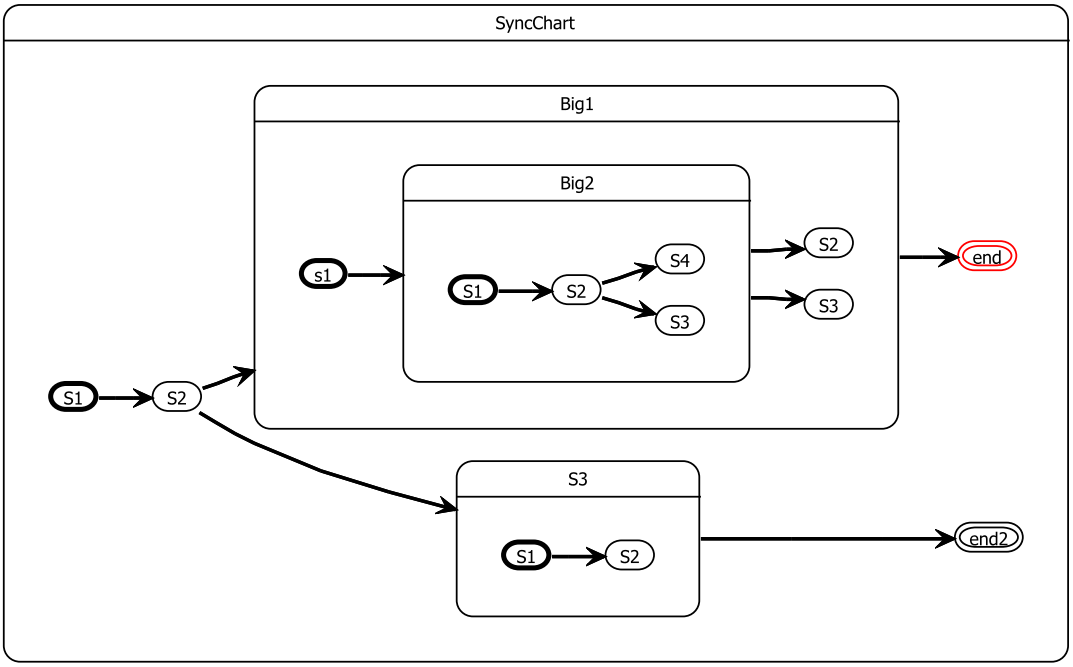

Abbildung 5.8: *Statechart mit Hervorhebung durch den ShapeHighlight-Effekt*

Änderung anderer Parameter – wie etwa der Linienstärke – stellt dagegen eine größere technische Schwierigkeit dar und ist daher nicht realisiert worden.

Die eingeschränkte Einsetzbarkeit des Effekts ist nachteilig zu bewerten. Abhilfe könnte die Einführung einer vereinheitlichten Schnittstelle zum Setzen entsprechender Objektparamter in allen eingesetzten Editoren leisten. In Anbetracht der zu erwartenden Nützlichkeit des Effekts sollte dies in Betracht gezogen werden.

5.5 Zusammenfassung

In diesem Kapitel wird die konkrete technische Umsetzung des Viewmanagements beschrieben.

Abschnitt 5.1 beschreibt, wie die in Kapitel 4 geplante Struktur in der Entwicklungsumgebung Eclipse umgesetzt wird. Die Erstellung der einzelnen Plugins und ihrer Bestandteile wird dargelegt sowie beispielhaft der Ablauf eines Durch-

laufs des Viewmanagements beschrieben.

Abschnitt 5.2 hat besondere Probleme, die bei der Realisierung des Viewmanagement auftraten, sowie deren Lösung zum Inhalt.

Die Adressierung der vom Viewmanagement verarbeiteten Objekte des jeweiligen Diagramms ist Thema des Abschnitts 5.3. Dies ist von besonderer Bedeutung, muss die Adressierung doch mit allen Plugins im KIELER-Projekt kompatibel sein, um eine breite Nutzbarkeit zu ermöglichen.

Die realisierten Effekte werden schließlich in Abschnitt 5.4 im Einzelnen beschrieben und ihre genaue Funktionalität dargelegt.

6 Verwandte Arbeiten

In diesem Kapitel werden Arbeiten diskutiert, die ähnliche Problemstellungen behandelt haben. Insbesondere werden die im Rahmen dieser Arbeiten entstandenen Softwareprojekte mit dem KIELER Viewmanagement, das Thema der vorliegenden Arbeit ist, verglichen. Es wird herausgearbeitet, welche Vor- und Nachteile die unterschiedlichen Lösungen haben und wie das Viewmanagement von den bei der Betrachtung dieser Arbeiten gewonnenen Erkenntnissen profitieren konnte.

Zunächst wird der Vorgänger von KIELER – das bereits in Kapitel 2 vorgestellte KIEL-Projekt – behandelt werden, danach ein Bestandteil von KIELER, das KiViK-Projekt (2.6.2).

Auf theoretischer Seite wird eine reaktive Sprache – Frappé – vorgestellt, die zur Entwicklung des Trigger-Event-Effekt-Schemas beigetragen hat.

Schließlich stellt das Topcased-Projekt eine artverwandete Parallelentwicklung dar, die beschrieben werden wird. Entwicklungswerkzeug und grundsätzliches Projektziel – das visuelle Modellieren von potentiell sicherheitskritischen Systemen – sind identisch mit denen von KIELER, jedoch hat das Topcased-Projekt eine abweichende Ausprägung.

6.1 Das KIEL-Projekt

Im Rahmen des KIEL-Projektes wurde am Lehrstuhl ein Statechart-Editor entwickelt. Dieser Editor diente zur grafischen und textuellen Modellierung von

Statecharts und deren Simulation. Auch im KIEL-Editor waren Techniken des Viewmanagements implementiert, die im Folgenden vorgestellt werden sollen:

- Highlighting: Während des Entwickelns werden die selektierten Zustände und während der Simulation die aktiven Zustände hervorgehoben. Dies geschieht durch Manipulation der gegebenen Figur, nämlich durch Ändern der Füllfarbe auf gelb beim Entwickeln und blau beim Simulieren. Weiterhin werden Transitionen hervorgehoben, wiederum gelb während der Entwicklung, grün beziehungsweise blau während der Simulation – Details dazu werden weiter unten aufgeführt.

- Filterung: Während der Simulation werden – bei aktivem Auto-Layout – Sub-Zustände von inaktiven Zuständen versteckt.

- Zooming: Eine Funktionalität *Fit-to-Screen* ist vorhanden, die die Zoomstufe permanent so anpasst, dass das gesamte Diagramm auf die Zeichenfläche passt.

- Layout: Ein permanentes Auto-Layout ist aktivierbar.

- Fokus und Kontext: Aktive Zustände werden mit der Funktionalität *Dynamic Chart* in den Fokus gerückt und mit maximalen Details dargestellt. Inaktive Zustände bilden den Kontext und werden mit minimalen Details abgebildet.

- Label-Management: Ist die Zoomstufe zu hoch, sodass Label zu klein und damit unlesbar werden, blendet der Editor sie komplett aus.

- Popups: Wird der Mauszeiger über ein Objekt gehalten, zeigt ein Popup den Objekt-Identifikator an.

Der entscheidende Nachteil dieser Implementierung eines Viewmanagements ist die mangelnde Anpassbarkeit von Effekten, Triggern und deren Zusammenspiel.

Dies wird besonders deutlich, wenn man die Ausprägung von Fokus und Kontext in KIEL betrachtet; es gibt nur maximale und minimale Details, keine Stufen dazwischen. Dies ist unter Umständen zuviel beziehungsweise zu wenig Information. Der Übergang von minimalen zu maximalen Details und umgekehrt

erfolgt abrupt beim Wechsel des Zustandes von aktiv zu passiv. Der Betrachter muss sich die Funktionalität des ehemals aktiven Zustandes gemerkt haben, um den Kontext zu begreifen; dies macht unter Umständen ein erneutes, manuelles Ausklappen des ehemals aktiven Zustandes nötig und bedeutet damit stark erhöhten kognitiven Aufwand und niedrigere Produktivität.

Das Viewmanagement in KIELER bietet die Möglichkeit, mithilfe des Filter-, des ZoomAndScrollTo- und des CompartmentCollapse/Expand-Effekts den Detailgrad sowohl der aktiven als auch der passiven Zustände frei zu definieren. Es ist im Prinzip sogar möglich, für einzelne Zustände spezielle Verhaltensweisen in den Kombinationen zu definieren, dies bedeutet jedoch erhöhten Aufwand bei der Erstellung dieser Kombinationen. Mit Hilfe eines Chronik-Triggers und einer Aufzeichnung des Simulationsverlaufes wäre es sogar möglich, auf Basis dieser Chronik mehrstufigen Fokus und Kontext zu realisieren, also etwa den aktiven Zustand mit vollen Details darzustellen, den davor aktiven Zustand mit etwas weniger Details, den vor diesem aktiven Zustand mit wiederum weniger Details und so weiter bis minimale Details erreicht sind.

Auch beim Highlighting ließen sich mit dem KIELER Viewmanagement bessere Resultate erzeugen als mit dem Highlighting in KIEL. Wiederum ist das Problem die fest vorgegebene Ausprägung des Highlight-Effekts. Anders als in KIEL lassen sich nicht nur die Effekte den Vorlieben des Benutzers anpassen, sondern es ist auch eine unterschiedliche Hervorhebung für unterschiedliche Typen von Objekten möglich. Durch Anpassung der Kombination können beispielsweise Fehler-Zustände farblich anders hervorgehoben werden als reguläre Zustände. Eine Anpassung an Größe und Komplexität des behandelten Diagramms – etwa deutlichere Effekte bei großen Diagrammen mit hoher Objektdichte – verspricht eine Steigerung der Produktivität.

Das Layout wird in KIEL entweder automatisch nach jeder Veränderung oder durch expliziten Befehl durch den Benutzer angewandt, jeweils auf das komplette Diagramm. Das KIELER Viewmanagement kann das Layout differenzierter anstoßen, indem die entsprechende Kombination bearbeitet wird. Denkbar wäre etwa ein Layouten nach einer bestimmten Anzahl von Veränderungen im Diagramm. Weiterhin ist es in KIELER möglich, nur Teile des Diagramms zu layouten und somit eine benutzerdefinierte Struktur in das Diagramm zu bringen.

KIEL bietet aber auch Funktionalitäten, die im Viewmanagement für KIELER nicht implementiert sind und beachtenswert sind. Das Anzeigen von Objektinformationen in einem Popup bei Überfahren mit dem Mauszeiger zählt dazu. Jedoch sollte die angezeigte Information für den Betrachter von Nutzen sein, die Anzeige kryptischer Identifikatoren wie in KIEL ist nicht empfehlenswert.

Das Label-Management ist in KIELER bisher wenig beachtet worden und ist im Rahmen der Weiterentwicklung der Layoutfunktionalität sicher verbesserungswürdig. Interessante Ansätze zum Labelmanagement wurden bereits in Kapitel 3 »Visualisierung von Modellen – Stand der Entwicklung« beschrieben. Unterschiedliche Layout-Methoden für Label könnten durch das Viewmanagement situativ angestoßen werden.

6.2 KiViK

Das KIELER-Plugin KiViK, welches bereits in Abschnitt 2.6.2 vorgestellt wurde, realisiert eine einfache Form des Viewmanagements. Beim Vergleich der beiden ausgewählten Diagramme werden die Unterscheide visuell durch Highlighting hervorgehoben: Hinzugefügte Objekte werden durch Setzen der Füllfarbe auf grün gekennzeichnet, gelöschte Objekte mit roter Farbe, geänderte Objekte mit blauer Farbe.

Weiterhin ist es möglich, beim Selektieren eines Objekts in einem Diagramm das korrespondierende Objekt in einem anderen Diagramm hervorgehoben darzustellen.

Zum Thema Fokus und Kontext bietet KiViK animiertes Scrollen – der sanfte Bildlauf soll die mentale Karte des Betrachters aufrecht erhalten – und ein Fokussieren auf ein angewähltes Objekt, analog zum ZoomAndScrollTo-Effekt im KIELER Viewmanagement.

Für das Viewmanagement sollte die Funktionalität eines animierten Scrollens in Betracht gezogen werden, eventuell mit anpassbarer Scroll-Geschwindigkeit.

6.3 Frappé

Wie bereits in Kapitel 4 erwähnt, leistete Frappé – eine Implementierung von funktionaler reaktiver Programmierung (FRP) in Java – einen Beitrag zur Entwicklung des Trigger-Event-Effekt-Modells im Viewmanagement [8].

FRP ist ein deklaratives Programmiermodell für interaktive Applikationen, die so erstellten Programme werden durch *Behaviours* und *Events* beschrieben. Behaviours – also Verhalten – sind kontinuierliche und reaktive Werte. Als einfachstes Beispiel hierfür ist das konstante Verhalten anzusehen, welches unabhängig von der Zeit dasselbe Verhalten – etwa die permanente Ausgabe eines fixen Wertes – zeigt. Die Events hingegen sind Zustände, die zu diskreten Zeitpunkten auftreten, zum Beispiel eine Benutzereingabe durch Drücken der Maustaste.

Frappé modelliert dieses Verhalten in Java, indem es sich folgende, Java-eigene, dem Behaviour/Event-Muster ähnliche Eigenschaften zunutze macht:

- Properties: Es besteht die Möglichkeit, Attribute von Objekten durch Aufruf geeigneter Methoden (getter/setter) auszulesen oder zu manipulieren.
- Events: Das Listener Pattern bietet Registrierungen an einem vorgefertigten Interface, die registrierten Clients werden über Events benachrichtigt.
- Methoden: Einfache Java-Methoden können aufgerufen werden, um gewisse Funktionalitäten zu übernehmen.

Die erzeugten Java-Klassen verpackt Frappé als *Java-Beans*.

Das KIELER Viewmanagement geht einen ähnlichen Weg. Die Events gehen von Triggern aus, Kombinationen können sich an diesen Triggern registrieren, um über Events benachrichtigt zu werden. Sie erhalten bei Auftreten dieser Events einen Übergabewert mit Informationen, aufgrund derer die Kombinationen und damit die Effekte ein Verhalten zeigen, nämlich die Darstellung von grafischen Effekten auf der Zeichenfläche. Die Übergabewerte wurden vorher durch Aufruf der entsprechenden getter- und setter-Methoden entsprechend gesetzt.

6.4 Topcased

Eine interessante Parallelentwicklung mit selber Basis, selbem Hauptziel – der visuellen Modellierung, aber anderem Schwerpunkt als das KIELER Projekt stellt das *Topcased*-Projekt (Toolkit In OPen source for Critical Applications & SystEms Development) dar, welches von Combemale et al. beschrieben wird [7].

Es ist wie KIELER ein auf Eclipse basierendes, modulares und generisches *Computer Aided Software Engineering* (CASE) Tool, das sowohl *Model Driven Engineering* (MDE) anbietet als auch selbst mit dieser Technologie entwickelt wurde. Zielsetzung des Projekts ist die Unterstützung bei der Entwicklung sicherheitskritischer Systeme, daher ist die Validierung und Verifikation von Modellen das Kernfeature von *Topcased*. Aus dem Anspruch der Validierung folgt die Notwendigkeit der Simulation des modellierten Systems und das Aufzeichnen derselben. Die Verifikation fordert die Möglichkeit, das System auf Einhaltung von *correctness properties* zu überprüfen bzw. nachzuvollziehen, wieso diese nicht eingehalten werden, um dann Abhilfe zu schaffen.

In Bezug auf Viewmanagement ist die *Simulationsvisualisierung* von Interesse. Hierbei geht es um unterstützende Maßnahmen, die dem Benutzer bei der Betrachtung und Auswertung der Simulation helfen.

Combemale et al. nennen hier das Hervorheben des gegenwärtigen Zustandes des simulierten Modells als das erste und einfachste Ziel. Sie führen an, dass hierzu die grafische Notation des Modells erweitert werden muss, um die zusätzliche Information darzustellen. Farben und klassische grafische Komponenten (Fortschrittsbalken, Anzeigen usw.) erscheinen den Autoren sinnvoll einsetzbar. So sollen die aktiven und die feuerbereiten Zustände durch spezifische Farbgebung hervorgehoben werden, das heißt aktive Zustände werden als rote Rechtecke und feuerbereite Transitionen als grüne Kanten dargestellt. Dies entspricht der farblichen Hervorhebung bei der Simulation im KIEL-Projekt.

Als weiteres Hilfsmittel bei der Verdeutlichung der Simulation führen Combemale et al. die Anzeige unterschiedlicher Fenster oder Anzeigeelemente an, die gewisse Informationen ausgelagert darstellen sollen, etwa Signale in Ab-

hängigkeit zum Zeitverlauf in der Simulation. Dieses Vorgehen ähnelt der bei *Esterel Studio* – einem Modellierungs- und Simlationswerkzeug – angewandten Technik.

Der gegenwärtig implementierte Prototyp von Topcased ist in seiner Ausprägung im Bereich Viewmanagement noch sehr rudimentär, lediglich ein Highlighting scheint implementiert zu sein. Bei der Simulation sicher hilfreich wäre eine Filterung von Informationen, die Möglichkeit zum Expandieren und Kollabieren von Zuständen, Zoom und Pan et cetera. Zudem scheinen keine weiteren Entwicklungen auf dem Bereich Layout geplant zu sein. Jedoch könnte es für das KIELER-Projekt interessant sein, von den Fähigkeiten des Topcased-Projekts im dem Bereich Simulation zu profitieren.

6.5 Zusammenfassung

In diesem Abschnitt werden Vergleiche zwischen dem Viewmanagement in KIELER und dem Viewmanagement – oder Ansätzen in dieser Richtung – in verwandten oder vergleichbaren Softwareprojekten gezogen.

Es wird die Verbesserung und Weiterentwicklung von Viewmanagement in KIELER im Vergleich zu seinem Vorgängerprojekt KIEL dargestellt und die theoretische Basis in dem Bereich der funktionellen, reaktiven Sprachen beschrieben.

Schließlich wird eine unabhängige Parallelentwicklung mit selber Entwicklungsbasis und selben Grundziel präsentiert und ihre Gemeinsamkeiten und Unterschiede dargelegt und bewertet.

7 Evaluation der Lösung

In diesem Kapitel wird die in der vorliegenden Arbeit realisierte Lösung betrachtet. Es wird analysiert, in welchem Umfang die erstellten Techniken und Methoden zur Lösung des anfänglich beschriebenen Problems beitragen und welche Einsatzmöglichkeiten sich für das Viewmanagement ergeben haben.

7.1 Beitrag zur Lösung des Problems

Das in Abschnitt 1.1 adressierte Problem bei Arbeiten mit visuellen Modellierungen manifestierte sich im Kern in einer für den Betrachter zu großen Informationsdichte und mangelnder Strukturierung. Es war dem Betrachter nicht möglich, momentan relevante von irrelevanten Daten zu trennen, ohne dabei eine große kognitive Anstrengung zu vollbringen, wobei diese Anstrengung fortlaufend erbracht werden musste.

Weiterhin bestand die Gefahr, dass die für die Orientierung in einem Diagramm essentielle mentale Karte nicht oder wiederum nur unter großer kognitiver Anstrengung aufrecht gehalten werden konnte. Das Resultat aus dem so entstehenden kognitiven Overhead war die Verschwendung von geistigen Ressourcen des Betrachters und Entwicklers mit nicht zweckdienlichen und fachfremden Aufgaben – also hauptsächlich der Orientierung im Diagramm.

Der eigentliche Zweck der Betrachtung des Diagramms, dessen Weiterentwicklung, fand aufgrund der durch den erwähnten kognitiven Overhead gebundenen geistigen Ressourcen des Betrachters und Entwicklers nicht effizient statt, die Produktivität litt.

Weiterhin war durch die Ablenkung des Betrachters und Entwicklers mit fachfremden Aufgaben eine erhöhtes Fehleraufkommen anzunehmen. Dies sollte besonders im Hinblick auf die möglicherweise sicherheitskritische Rolle der entwickelten Modelle und Programme dringend vermieden werden.

Ausgehend von der Darstellung eines Statecharts ohne die im Viewmanagement realisierten Techniken und Methoden (siehe Abbildung 7.1) soll im Folgenden dargelegt werden, welche Verbesserungen diese Techniken und Methoden erreicht haben.

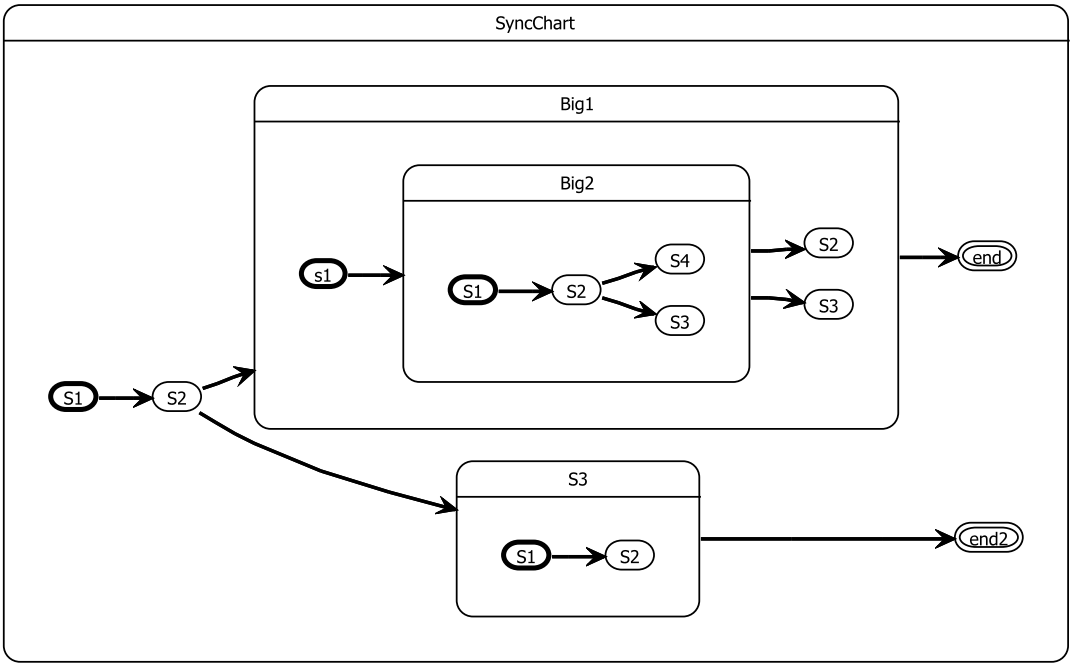

Abbildung 7.1: *Statechart ohne Effekte des Viewmanagements*

7.1.1 Highlight-Effekte

Im Viewmanagement wurden zwei Highlight-Effekte realisiert: Der normale Highlight-Effekt und der ShapeHighlight-Effekt.

Ziel der Effekte ist es, relevante Objekte aus der Masse der anderen Objekte hervorzuheben, um dem Betrachter ein einfacheres Auffinden dieser Objekte zu ermöglichen.

Der normale Highlight-Effekt zeichnet dabei zusätzliche grafische Informationen in das Diagramm, um die betroffenen Objekte hervorzuheben – standardmäßig ein rotes Rechteck (Abbildung 7.2). Farbe und Liniestärke sind veränderbar.

Der ShapeHighlight-Effekt hingegen verändert das bereits bestehende, betroffene Objekt, sodass es gegenüber den anderen hervorsticht (Abbildung 7.3). Standardmäßig wird die Linienfarbe hierbei auf rot gesetzt.

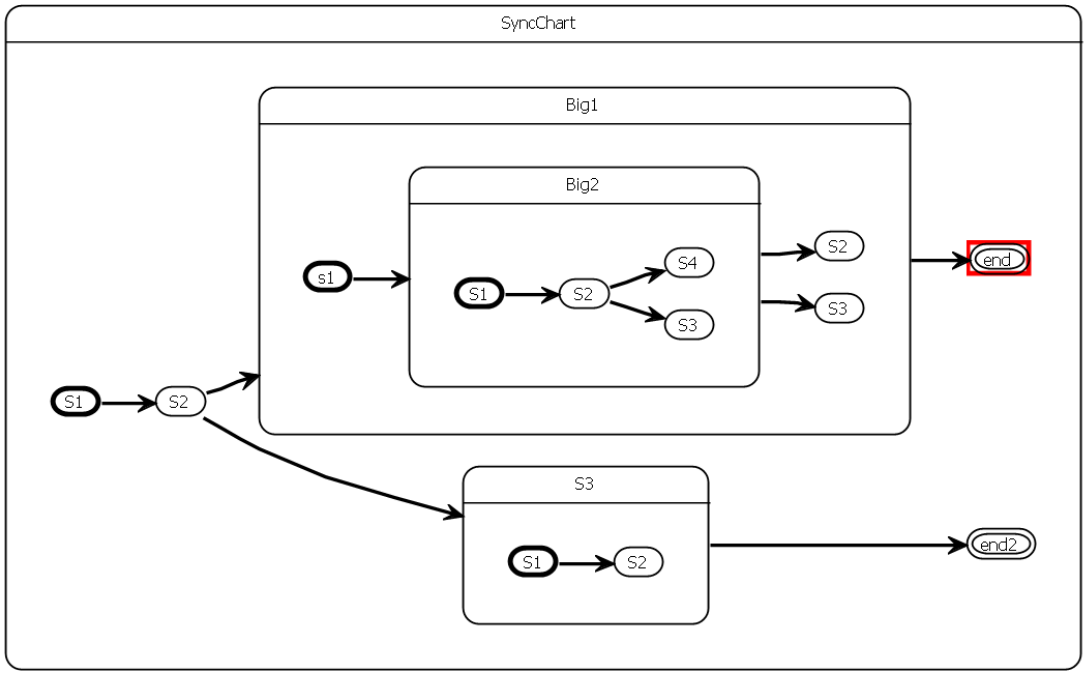

Abbildung 7.2: *Statechart mit Hervorhebung durch den Highlight-Effekt*

Durch diese Hervorhebung wird die Erkennung eines relevanten Zustandes leicht möglich, die Aufmerksamkeit des Betrachters wird unmittelbar auf die Hervorhebung gezogen. Der kognitive Aufwand beim Suchen eines so markierten Objekts ist stark herabgesetzt, die Nützlichkeit des durch diesen Effekt geleisteten Beitrages offensichtlich.

Im Bereich der Simulation wird durch diese Effekte erst eine Kennzeichnung des aktiven Zustandes – und somit faktisch die grafische Darstellung einer Simulation möglich.

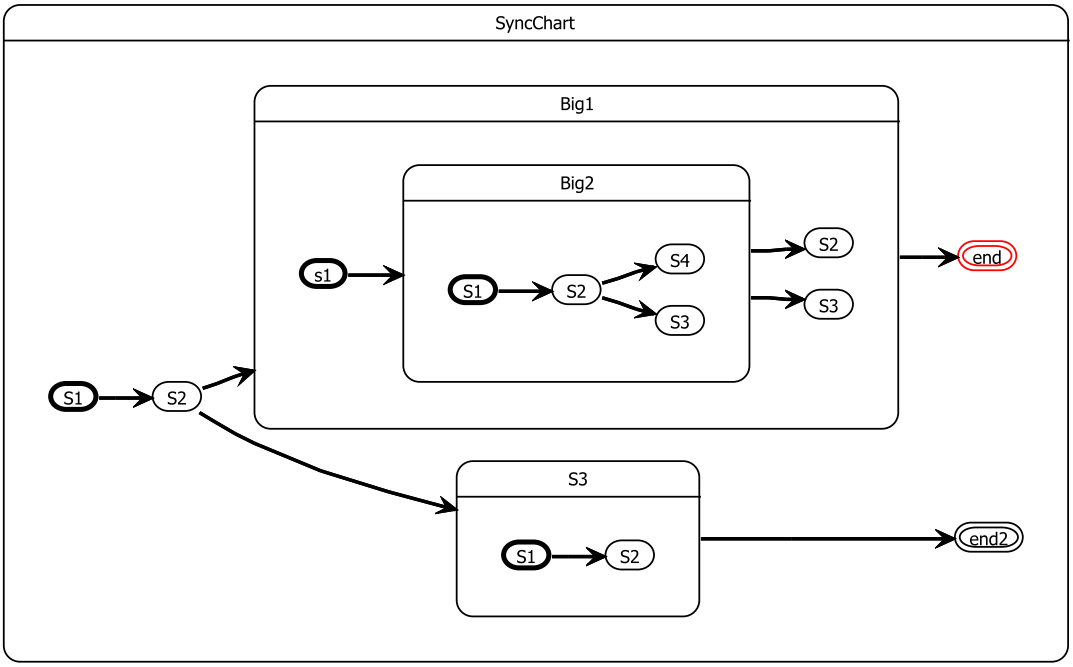

Abbildung 7.3: *Statechart mit Hervorhebung durch den ShapeHighlight-Effekt*

7.1.2 Filter-Effekt

Der Filter-Effekt wurde als weiterer, nach den Highlight-Effekten sehr viel versprechender Effekt eingeführt.

Technisch einfach in der Umsetzung verbirgt er Objekte, die im Moment als nicht wichtig erachtet werden und ermöglicht somit eine Reduktion der Informationsdichte (Abbildung 7.4) oder aber eine kompaktere Darstellung der relevanten Objekte. Letzteres erfordert jedoch ein erneutes Layouten des Diagramms, damit die durch die Filterung entstandenen Lücken gefüllt werden können. Bei großer Reduktion der Anzahl der Objekte durch Filterung kann dies zu einer starken Umorganisation des Diagramms durch das Layout führen, die Erhaltung der mentalen Karte stellt auch dann eine Herausforderung dar.

Der Nutzen des Filter-Effekts ist jedoch unbestreitbar, ist er doch eine äußerst effektive Möglichkeit der Reduzierung der dargestellten Informationen auf das Wesentliche.

Dabei ist die Aufhebung dieses Effekts ebenso einfach – eine Veränderung der

relevanten Objekte jederzeit möglich. Wurde jedoch nach der Filterung ein neues Layout auf das Diagramm angewandt, ist nach dem Sichtbarmachen der vorher versteckten Objekte ein sofortiges Layouten nötig, da sich Objekte möglicherweise überlappen oder gar verdecken.

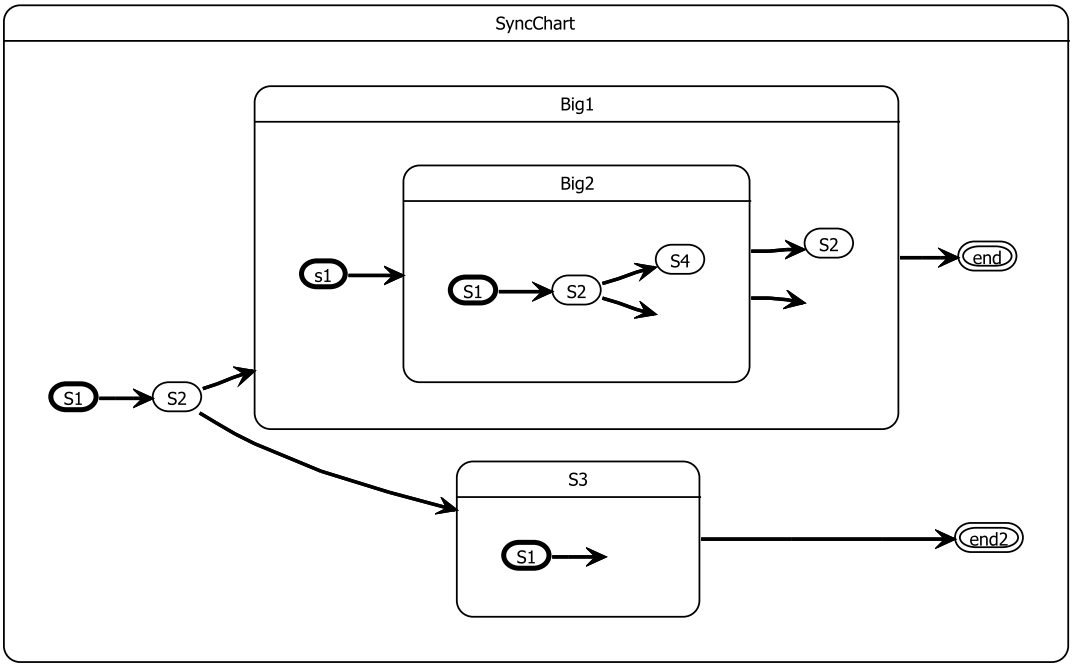

Abbildung 7.4: *Statechart mit herausgefilterten Objekten – ohne erneutes Layouten*

7.1.3 Collapse/Expand-Effekt

Der Collapse-Effekt fällt im Prinzip in dieselbe Kategorie wie der Filter-Effekt, versteckt er doch ebenso Objekte, die momentan nicht benötigt werden.

Dabei geht der Collapse-Effekt aber einen anspruchsvolleren Weg, indem er sich an den Compartments der Objekte orientiert und diese einklappt. Ein anschließendes Layouten ist quasi zwingend notwendig, da das das Compartment enthaltende Objekt ansonsten seine ursprüngliche Größe beibehält und der Nutzen des Effekts damit eher gering bleibt. Sinnvoll ist dieser Effekt einzusetzen, wenn eine Verflachung der Hierarchie in einem Diagramm gewünscht ist – die unterste Hierarchieebene wird eingeklappt (siehe Abbildung 7.5) – oder gewis-

se Informationen nicht benötigt werden, etwa die Deklaration von Signalen in einem Statechart.

Mit dem Expand-Effekt können die so versteckten Objekte sehr einfach wieder sichtbar gemacht werden.

Der Filter-Effekt kann dies nicht leisten, da er solche Bestandteile des Diagramms nicht als Objekte wahrnimmt und somit nicht verstecken kann.

Durch die Möglichkeit, diesen Effekt sehr differenziert zur Reduzierung der Informationsdichte einzusetzen, wird dessen Nutzen als sehr hoch eingeschätzt.

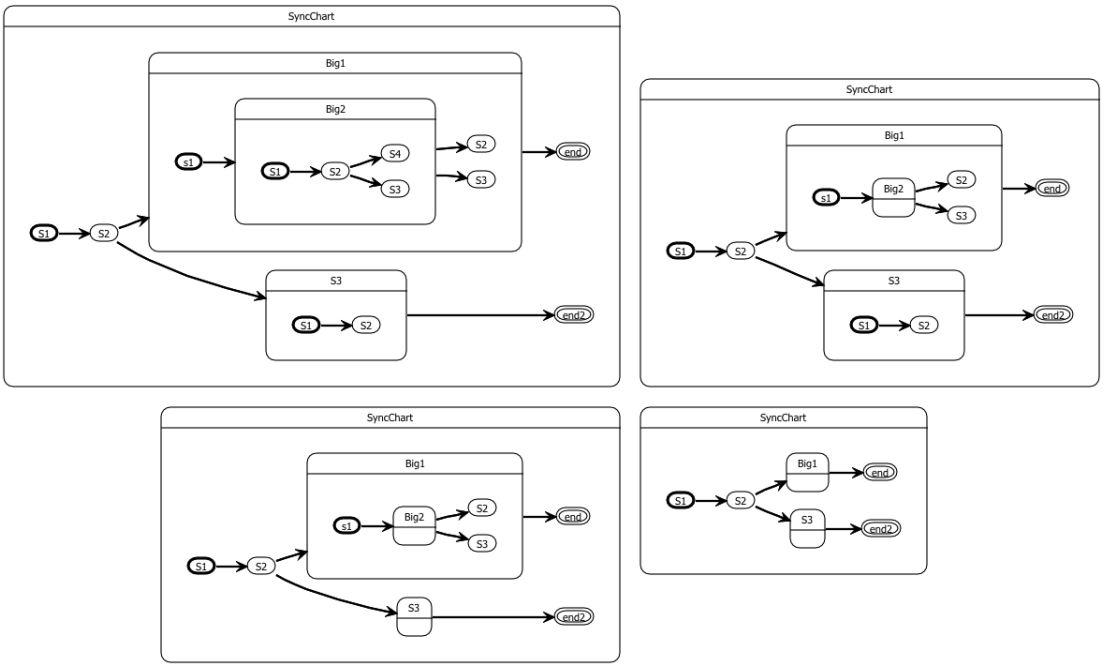

Abbildung 7.5: *Statechart mit Collapse-Effekt – Stufenweise Verflachung der Hierarchie*

7.1.4 Zoom-Effekte

Für die Navigation in Diagrammen leisten die Zoom-Effekte – ZoomToFit und ZoomAndScrollTo – wertvolle Beiträge.

Der ZoomToFit-Effekt setzt die Zoomstufe so, dass das gesamte Diagramm auf die Zeichenfläche passt und gibt dem Betrachter somit die Möglichkeit, sich zu

orientieren, eine Übersicht zu gewinnen oder Strukturen zu identifizieren (siehe Abbildung 7.6).

Der ZoomAndScrollTo-Effekt bietet die Möglichkeit, zu einem bestimmten Objekt zu navigieren und es in den Fokus zu rücken. Dazu zoomt der Effekt in die Darstellung ein, um das Objekt möglichst groß darzustellen und scrollt zu dem Objekt (siehe Abbildung 7.7). Ein frei setzbarer Offset-Wert bestimmt dabei, wie viel der Umgebung angezeigt werden soll und gibt dem Benutzer damit die Möglichkeit, die Menge der zusätzlich angezeigten Informationen zu regulieren.

Für die Navigation im Diagramm und besonders für die selbsttätige Navigation während einer Simulation ist dieser Effekt unverzichtbar und wirkungsvoll. Er entlastet den Benutzer beim Auffinden eines Objekts, da dieser nur das gewünschte Objekt spezifizieren muss. Durch Darstellung eines Kontextes kann die mentale Karte wirkungsvoll erhalten werden.

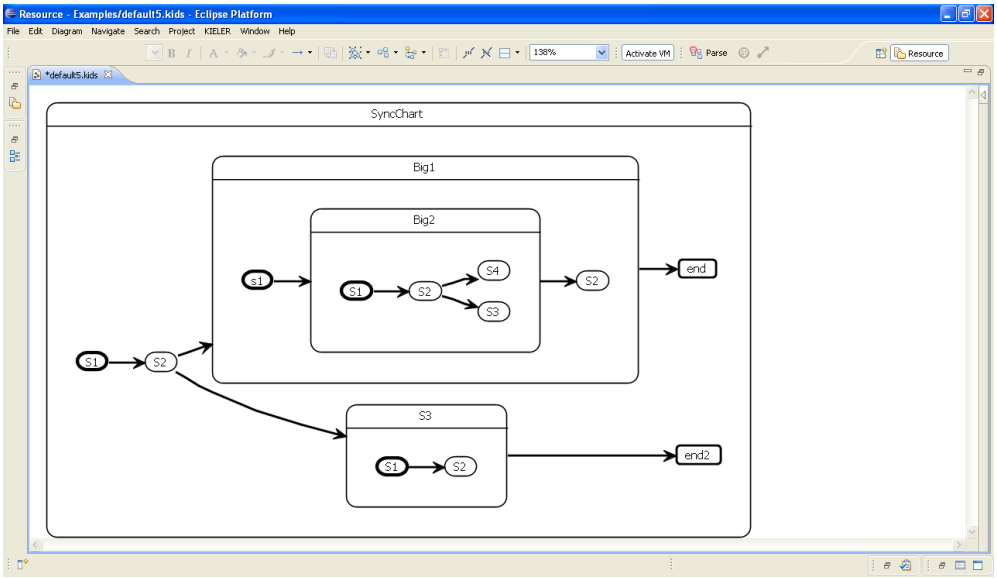

Abbildung 7.6: *KIELER-Editor nach Anwendung des ZoomToFit-Effekts auf einen Statechart*

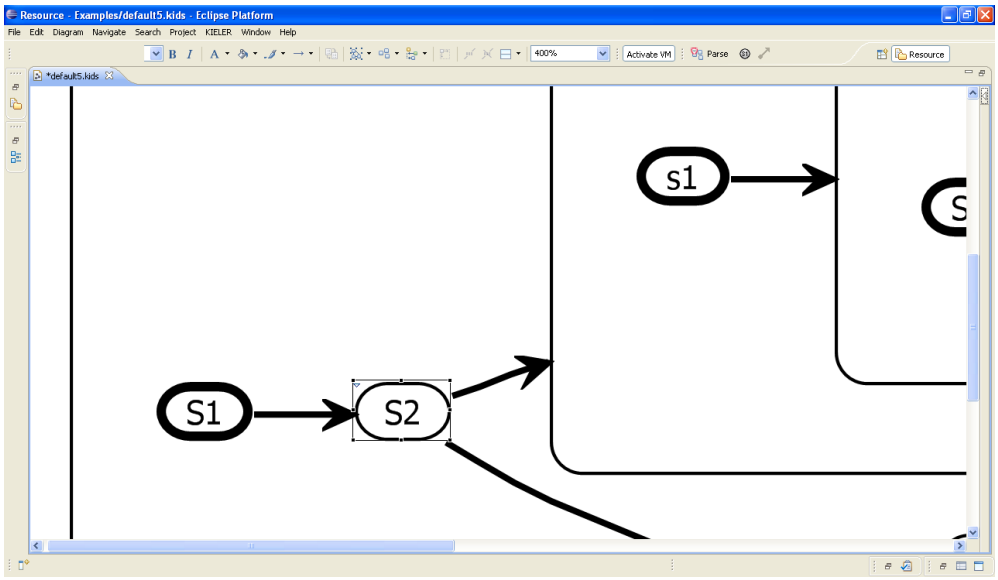

Abbildung 7.7: *KIELER-Editor nach Anwendung des ZoomAndScrollTo-Effekts auf ein Objekt in einem Statechart*

7.2 Einsatz im KIELER-Projekt

Der Einsatz in anderen Teilprojekten von KIELER ist ein weiteres, wichtiges Ziel des Viewmanagements.

Wie bereits erwähnt, ist die Darstellung einer Simulation erst durch die Möglichkeit, einen aktiven Zustände kenntlich zu machen oder Werte darzustellen, realisierbar. Diese Möglichkeiten werden durch das Viewmanagement zur Verfügung gestellt.

Es ergeben sich also zahlreiche Anknüpfungspunkte zur Verwendung des Viewmanagements in anderen Plugins, die im Folgenden aufgeführt werden:

Simulation: Im KIEM-Projekt, welches bereits in Abschnitt 2.6.4 vorgestellt wurde, spielt das Viewmanagement eine wichtige Rolle. Da das Projekt noch in der Entwicklung ist, werden nur einfache Funktionalitäten des Viewmanagements genutzt, nämlich bei Statecharts das Hervorheben aktiver Zustände durch den ShapeHighlight-Effekt. Dazu wurde ein entsprechender Trigger und eine Kombination von Christian Motika implementiert [23]. Es sind jedoch wesentlich ausgefeiltere Verhaltensweisen des Viewmanagements in Verbindung mit einer Simulation denkbar, die Skiz-

ze eines solchen Verhaltens wird in Abschnitt 8.1.1 aufgeführt.

Strukturelles Editieren: In dem von Michael Matzen momentan entwickelten Projekt KSBasE [21] wird das Viewmanagement eingesetzt, um nach jedem Editierungsschritt – etwa dem Hinzufügen eines neuen Zustandes – das Layout neu zu berechnen. Dies dient der Beibehaltung der Übersichtlichkeit und Lesbarkeit, die gerade beim Hinzufügen von Objekten sonst schnell kompromittiert wäre. Für dieses Verhalten wurde ebenfalls ein entsprechender Trigger und eine Kombination implementiert.

Textuelles Editieren: Auch im Zusammenhang mit dem Plugin für textuelles Modellieren – KITE, gegenwärtig in Entwicklung durch Özgün Bayramoglu [4] – ergeben sich mögliche Szenarien für den Einsatz des Viewmanagements. So könnten zum Beispiel Objekte, die gerade im textuellen Editor bearbeitet werden, durch das Viewmanagement hervorgehoben und fokussiert werden.

Obwohl die ersten Einsätze des Viewmanagements noch experimentell waren und großes Verbesserungspotential bieten, waren die dabei gemachten Erfahrungen positiv.

Das Viewmanagement verhielt sich wie erwartet und konnte gut in die vorhandenen Plugins eingebunden werden. Durch den Einsatz des Viewmanagements konnte eine Verbesserung der Benutzerfreundlichkeit und des Funktionsumfangs der benutzenden Plugins erreicht werden – dies bestätigen auch Aussagen der jeweiligen Entwickler und Benutzer.

7.3 Zusammenfassung

Dieses Kapitel umfasst die Evaluation der realisierten Techniken und Methoden. Es wurden die realisierten Effekte in ihrer Wirkungsweise beschrieben und verglichen, weiterhin wurden ihre Vorteile und nutzbringenden Eigenschaften hervorgehoben. Ebenso wurde der Einsatz des Viewmanagements in anderen Teilprojekten von KIELER diskutiert und bereits erfolgte Anwendungen aufgeführt.

8 Abschluss

In diesem Kapitel wird eine Übersicht über die möglichen zukünftigen Arbeiten gegeben, die bei der Erstellung dieser Arbeit aufgefallen sind und sinnvoll erscheinen. Weiterhin wird eine kurze Zusammenfassung der gesamten Arbeit und ein Fazit gegeben.

8.1 Zukünftige Arbeiten

Im Zusammenhang mit dieser Arbeit sollte eine Experimentierplattform für das Viewmanagement in KIELER geschaffen werden. Dies ist gelungen, jedoch ergeben sich aus der Natur dieser Realisation als experimentellem Bestandteil zahlreiche Gelegenheiten für Verbesserungen und Weiterentwicklungen. Die wichtigsten dieser Gelegenheiten, die bei der Erstellung des Viewmanagements aufgefallen sind, aber nicht realisiert werden konnten, sollen im Folgenden vorgestellt werden.

8.1.1 Erstellung ausgefeilter Kombinationen

Im Plugin Viewmanagement wurde eine Sammlung von Effekten, die nutzbringend erschienen, implementiert sowie dazugehörige Kombinationen mit Triggern, die zum einem dem Testen der Effekte dienten, in einigen Fällen auch – wie im Abschnitt 7.2 beschrieben – zum experimentellen Einsatz in anderen Teilprojekten von KIELER. Die hierfür erstellten Kombinationen waren jedoch zunächst nur einfach gehalten, um mit dem Zusammenspiel der Plugins und der Effekte vertraut zu werden. Dies schöpft aber bei weitem nicht die Möglichkeiten des Viewmanagements in Bezug auf die Verbesserung der Produktivität

und die Reduktion des kognitiven Aufwands bei der Arbeit mit Diagrammen aus.

Für die Simulation etwa wäre – insbesondere bei größeren Diagrammen – eine ausgefeiltere Kombination von Vorteil; ihr Ablauf soll im Folgenden grob skizziert werden:

- Anfänglich wird der ZoomToFit-Effekt einmalig ausgeführt, der so gesetzte Zoomfaktor ausgelesen.
- Ist der ausgelesene Zoomwert kleiner als 75%, wird angenommen, dass das Diagramm zu groß ist, um die Simulation ohne Zoomen lesbar darzustellen, es wird also der ZoomAndScrollTo-Effekt aktiviert, jedoch ein großer Wert für die anzuzeigende Umgebung gewählt. Der Effekt reagiert auf den Simulations-Trigger.
- Der ShapeHighlightEffekt wird aktiviert, auch er reagiert auf den Simulations-Trigger.
- Der Expand-Effekt entfaltet alle aktiven Objekte, der Collapse-Effekt faltet sie danach – unmittelbar oder nach einer Anzahl Simulationsschritte – wieder zusammen.
- Um dem Collapse- und dem Expand-Effekt zu voller Wirksamkeit zu verhelfen, wird nach jedem Simulationsschritt ein Layout berechnet und angewandt.

Als Problem bei solchen sehr umfassenden Kombinationen wird in erster Linie die richtige Abfolge der Effekte erachtet (Wann scrollen, wann Collapse/Expand, wann Layout?). Gegebenenfalls macht dies das mehrmalige Ausführen – etwa des ZoomAndScrollTo-Effekts – notwendig. Andernfalls könnte der Benutzer etwa einen Ausschnitt angezeigt bekommen, dessen Inhalt durch anschließendes Layouting verrrutscht und der Ausschnitt so nicht mehr die ursprünglich gewünschten Objekte beinhaltet. Der Entwicklungsaufwand ist auch Grund dafür, warum solche Kombinationen in der vorliegenden Arbeit nicht realisiert werden konnten.

Ein weiteres Problem könnte die Ausführungsdauer einer solchen Kombination sein. Bereits das Ausführen des Layout-Effekts allein kostet – insbesondere

bei größeren Diagrammen – einige Sekunden Zeit, der Collapse/Expand-Effekt benötigt Zeit in ähnlichem Maßstab. In einer Kombination könnten diese Effekte die Geduld des Beobachters auf die Probe stellen, um so mehr, da diese Kombination zügig während einer Simulation ausgeführt werden soll.

Die Performance einzelner Effekte ist im Einzelfall verbesserungswürdig.

8.1.2 Angrenzende Bereiche

Während der Recherche für die Kapitel 3 »Visualisierung von Modellen – Stand der Entwicklung« und 6 »Verwandte Arbeiten« entwickelten sich einige Ideen zu Themen, die das Viewmanagement nicht direkt berühren, jedoch eventuell für das KIELER-Projekt insgesamt interessant sind. Diese Ideen sollen im Folgenden kurz skizziert werden, weiterhin soll auf bereits ausgearbeitete Techniken aus dem Kapitel 3 hingewiesen werden.

- Übersichtskarte: In Eclipse ist eine Übersichtskarte standardmäßig vorhanden und kann eingeblendet werden. Diese Karte ist jedoch relativ rudimentär, es wird nur eine optisch verkleinerte Version des gesamten Diagramms angezeigt, aber weiterhin der momentan betrachtete Ausschnitt kenntlich gemacht. Eine Verbesserung dieser Übersichtskarte könnte Vorteile bei der Navigation in großen und unübersichtlichen Diagrammen bieten. Denkbar wäre etwa, die angezeigte verkleinerte Version des Diagramms durch eine abstrahierte Version zu ersetzen, auf der Strukturen besser sichtbar werden, etwa durch Reduzierung der Hierarchieebene. Weiterhin wäre ein Effekt denkbar, der das betroffene Objekt auf der Übersichtskarte deutlich sichtbar macht, ohne die Hauptansicht zu verändern. Somit könnte die Lage der Hauptansicht zu diesem Objekt deutlich gemacht werden – dies würde eine Verstärkung der mentalen Karte darstellen.

- Navigator: Bei sehr großen Diagrammen könnte sich eine Art Register der enthaltenen Objekte als nützlich erweisen. Denkbar wäre zum einen eine Baumdarstellung der Objekte um ihre Abhängigkeit zueinander darzustellen – dieses Verhalten ist bereits als *Outline* in Eclipse realisiert. Hier

könnten die Verbesserungen in der Verknüpfung dieser Outline mit der grafischen Darstellung liegen, etwa eine Kombination, die bei einfachem Klicken mit der Maus auf ein Objekt in der Outline dieses kurz hervorhebt – etwa durch dreimaliges Aufblinken –; ein Doppelklick mit der Maus fokussiert die Hauptansicht auf dieses Objekt. Weiterhin könnte die Auflistung der im Diagramm enthaltenen Objekte nach unterschiedlichen Kriterien nützlich sein. Denkbar wäre die Sortierung nach Label, Identifikator, Typ, Ziel und Quelle et cetera.

- Aus dem Bereich der bereits erforschten und beschriebenen Beiträge erscheint der Beitrag von Wong et al. [44] zum Thema Labelmanagement viel versprechend. Es sollte in Betracht gezogen werden, die dort dargestellten Lösungen in KIELER zu implementieren.

- Ebenso ist die Technik der Fillets, die von Lukka et al. [20] entwickelt wurde, interessant.

8.1.3 Kognitive Untersuchungen

Wie bereits Tory und Möller bemängeln, existieren zu vielen Entwicklungen im Bereich der Visualisierung keinerlei empirische Untersuchungen zu ihrem Nutzen oder ihrem Beitrag zur Verbesserung von Produktivität, zur Reduzierung des Suchaufwands oder der Suchzeit et cetera.

Es sollte in Betracht gezogen werden, für das Viewmanagement in KIELER entsprechende empirische Untersuchungen durchzuführen, um den Nutzen der einzelnen Effekte und ihrer Kombinationen für den jeweiligen Anwendungszweck zu quantifizieren – etwa anhand der Suchzeit in einem Diagramm oder der Fehlerhäufigkeit.

Eine Anleitung zur Durchführung einer solchen Studie liefern Ware et al. [42] – für das KIEL-Projekt wurden ähnliche Untersuchungen bereits von Jonas Völcker im Rahmen seiner Diplomarbeit durchgeführt [41].

8.2 Zusammenfassung und Fazit

Die vorliegende Arbeit hat die Entwicklung eines Viewmanagements für visuelles Modellieren zum Ziel.

Das Viewmanagement soll sich der Problematik der mangelnden Pragmatik beim Arbeiten mit visuellen Modellen annehmen und Lösungsmöglichkeiten anbieten. Die dort vorhandenen Probleme entspringen im Wesentlichen der stark angewachsenen Komplexität der modellierten Systeme. Daraus folgend ergeben sich Schwierigkeiten für den Betrachter und Entwickler. Besonders problematisch ist bei Diagrammen mit hoher Informationsdichte das Hinzufügen von Objekten, für die erst umständlich und manuell Platz geschaffen werden muss – dies bedeutet die Inanspruchnahme des Entwicklers oder Betrachters mit fachfremden Aufgaben. Weiterhin kann in sehr großen Diagrammen das Navigieren und Orientieren beim Wechseln zwischen einer großen Übersicht und einer Detailansicht eine große kognitive Anstrengung bedeuten, wenn nicht von Seiten der Entwicklungsumgebung entsprechende Hilfen angeboten werden. Auch durch diese Anstrengung wird der Betrachter von seiner eigentlichen Aufgabe – der Entwicklung des Modells – abgelenkt. Dies hat eine Beeinträchtigung seiner Produktivität zur Folge, die es zu vermeiden gilt. Ein anderer unerwünschter Effekt dieser Ablenkung kann zum Beispiel ein höheres Fehleraufkommen im entwickelten Modell sein.

Das Viewmanagement soll sich dieser Probleme annehmen und Abhilfe schaffen, indem zum Beispiel die erwähnten Hilfen bei Orientierung und Navigation angeboten werden. Im Kern besteht die Aufgabe des Viewmanagements darin, Methoden zur Regulierung der Informationsdichte anzubieten, damit der Betrachter die wesentlichen Informationen in möglichst optimaler Aufbereitung dargestellt bekommt.

Die vorliegende Arbeit behandelt nach einer Problembeschreibung und einer Zieldefiniton zunächst das Umfeld, in dem die Lösung entwickelt wird. Es werden das KIELER-Projekt mit seiner Entwicklungsbasis Eclipse, deren Bestandteilen EMF, GEF und GMF sowie die KIELER-Teilprojekten vorgestellt. Das KIELER-Projekt hat die Bereitstellung einer Umgebung zur Entwicklung von visuellen Modellen zum Ziel, wobei ein besonderer Fokus auf der Pragmatik bei der Ent-

wicklungstätigkeit liegt. Das in der vorliegenden Arbeit entwickelte Viewmanagement soll Teil dieser Entwicklungsumgebung sein.

Es wird weiterhin eine Übersicht über den Stand der Entwicklung auf dem Bereich der Visualisierung von Modellen gegeben. Hier wird herausgearbeitet, welche Methoden und Techniken in der Forschung entwickelt worden sind, die einen Beitrag zur Problemlösung leisten können. Es wird dargelegt, welche dieser Beiträge in welchem Umfang in das KIELER Viewmanagement einfließen sollen.

Anhand dieses so gewonnenen Extrakts aus der gegenwärtigen Forschung wird ein eigener Ansatz für das Viewmanagement erstellt. Es werden konkrete Ziele für die zu realisierende Lösung festgelegt, eine Struktur bestehend aus Triggern, Effekten und Kombinationen aus beiden für die zu entwickelnden Hilfsmittel definiert sowie konkrete Effekte ausgewählt, die dem Betrachter die nötige Hilfestellung bei der Navigation und Orientierung im Diagramm geben sollen. Ebenso werden Abläufe und Verfahren für die Kommunikation und Verwaltung in der zu implementierenden Lösung beschrieben.

Die konkrete Umsetzung des so beschriebenen Konzepts beinhaltet detaillierte Angaben zur Erstellung der benötigten Plugins in der Entwicklungsumgebung Eclipse, eine Definition der zu erstellenden abstrakten Klassen für Trigger, Effekte und Kombinationen sowie deren Verwaltung. Ebenfalls wird das Verhalten des Viewmanagements anhand eines Beispiels von der Auslösung des Triggers bis zur Darstellung des Effekts im Diagramm beschrieben.

Besondere Herausforderungen, die bei dieser Implementierung aufgetreten sind werden adressiert und ihre Lösungen dargelegt und schließlich die erstellten Effekte im Detail vorgestellt.

Die erstellte Lösung des Viewmanagements in KIELER wird mit bereits bestehenden Arbeiten und Softwareprojekten verglichen und aus diesem Vergleich Stärken und Schwächen der jeweiligen Betrachtungsobjekte herausgearbeitet.

Es folgt eine bewertende Betrachtung der angefertigten Lösung. Hier wird analysiert, in welchem Maße die Bestandteile des KIELER Viewmanagements zur Lösung des anfänglich adressierten Problems beitragen und der bereits erfolgte Einsatz in anderen Teilen von KIELER analysiert.

Schließlich werden mögliche zukünftige Arbeiten vorgestellt, sowohl als Verbesserung des KIELER Viewmanagements als auch in anderen Bereichen von KIELER.

Die vorliegende Arbeit hat einen Beitrag zur Lösung des beschriebenen Problems beim visuellen Modellieren geleistet. Die implementierte Experimentierplattform für das KIELER-Projekt bietet unmittelbare Hilfen bei der Navigation und Orientierung in großen Diagrammen und entlastet den Betrachter und Entwickler; die erzielte Verbesserung der Pragmatik bei der Arbeit mit visuellen Modellen verspricht eine Steigerung der Produktivität des Entwicklers. Die erzielten Erfolge sind noch stark ausbaubar, hierfür liefert das KIELER Viewmanagement die entsprechende Basis.

Literaturverzeichnis

[1] ANDRÉ, Charles: *Semantics of S.S.M. (Safe State Machine)*. I3S Laboratory - UMR 6070 University of Nice, April 2003

[2] BATTISTA, Giuseppe di ; EADES, Peter ; TAMASSIA, Roberto ; TOLLIS, Ioannis G.: Algorithms for drawing graphs: an annotated bibliography. In: *Computational Geometry* Bd. 4, 1994, S. 235–282

[3] BATTISTA, Guiseppe di ; EADES, Peter ; TAMASSIA, Roberto ; TOLLIS, Ioannis G. ; STEELE, Laura (Hrsg.): *Graph Drawing - Algorithms for the Visualization of Graphs*. Alan Apt, 1999

[4] BAYRAMOGLU Özgün: *KIELER Infrastructure for Textual Modeling*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, (In Vorbereitung)

[5] BELL, Ken: *Überprüfung Syntaktischer Robustheit von Statecharts auf der Basis von OCL*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, November 2006

[6] BRIDGEMAN, Stina ; TAMASSIA, Roberto: Difference Metrics for Interactive Orthogonal Graph Drawing Algorithms. In: *Journal of Graph Algorithms and Applications*, Springer-Verlag, 2000, S. 57–71

[7] COMBEMALE, B. ; CRÉGUT, X. ; GIACOMETTI, J.-P. ; MICHEL, P. ; PANTEL, M.: Introducing Simulation and Model Animation in the MDE Topcased Toolkit. In: *In Proceedings 4th European Congress EMBEDDED REAL TIME SOFTWARE (ERTS), Toulouse, France*, 2008

[8] COURTNEY, Antony: Frappé: Functional Reactive Programming in Java. In:

In Proceedings of Symposium on Practical Aspects of Declarative Languages. ACM, Springer-Verlag, 2001, S. 29–44

[9] DAMEROW, Julia: *Entwicklung eines MDD-Tools für eine virtuelle Ausstellung*, Technischen Fachhochschule Berlin Fachbereich VI Informatik und Medien, Diplomarbeit, August 2008

[10] DIEHL, Stephan ; GOERG, Carsten ; KERREN, Andreas: Preserving the Mental Map using Foresighted Layout. In: *In Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization VisSym'01*, Springer Verlag, 2001, S. 175–184

[11] FUHRMANN, Hauke ; HANXLEDEN, Reinhard von: On the Pragmatics of Model-Based Design / Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group. 2009 (0913). – Forschungsbericht

[12] HAREL, David ; LACHOVER, Hagi ; NAAMAD, Amnon ; PNUELI, Amir ; POLITI, Michal ; SHERMAN, Rivi ; SHTULL-TRAURING, Aharon ; TRAKHTENBROT, Mark: STATEMATE: A Working Environment for the Development of Complex Reactive Systems. In: *IEEE Transactions on Software Engineering* 16 (1990), S. 5

[13] HENNIG, Manfred ; SEEBERGER, Heiko: *Einführung in den Ëxtension Point-Mechanismus von Eclipse*. JAVASPEKTRUM, January 2008

[14] HERMAN, Ivan ; SOCIETY, Ieee C. ; MELANCËON, Guy ; MARSHALL, M. S.: Graph Visualization and Navigation in Information Visualization: A survey. In: *IEEE Transactions on Visualization and Computer Graphics* 6 (2000), January-March, Nr. 1, S. 24–43

[15] HOCHMAIR, Hartwig ; FRANK, Andrew U.: A Semantic Map as Basis for the Decision Process in the www Navigation. In: *Conference on Spatial Information Theory. D. R. Montello (Ed.). Morro Bay, California, USA*, Springer Verlag, 2001, S. 173–188

[16] JACOBS, Steffen: *Konzepte zur besseren Visualisierung grafischer Datenflussmodelle*, Christian-Albrechts Universität zu Kiel, Department of Computer

Science, Real-Time and Embedded Systems Group, Diplomarbeit, January 2007

[17] KEIM, Daniel A.: Information visualization and visual data mining. In: *IEEE Trans. Visual. Comput. Graph* 8 (2002), January-March, Nr. 1, S. 100–107

[18] KLOSS, Tobias: *Automatisches Layout von Statecharts unter Verwendung von GraphViz*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, May 2005

[19] KOSARA, Robert ; MIKSCH, Silvia ; HAUSER, Helwig: Focus and context taken literally. In: *IEEE Computer Graphics and Applications* 22 (2002), S. 22–29

[20] LUKKA, Tuomas J. ; KUJALA, Janne V. ; NIEMELÄ, Marketta: Fillets: Cues for Connections in Focus+Context Views of Graph-like Diagrams. In: *Sixth International Conference on Information Visualisation* Dept. of Mathematical Information Technology, University of Jyväskylä, Finland, 2002

[21] MATZEN, Michael: *Structure-Based Editing in Eclipse*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, (In Vorbereitung)

[22] MOORE, Bill ; DEAN, David ; GERBER, Anna ; WAGENKNECHT, Gunnar ; VANDERHEYDEN, Philippe: *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. First Edition. IBM International Technical Support Organization, February 2004

[23] MOTIKA, Christian: *KIELER Leveraging Ptolemy Semantics*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, (In Vorbereitung)

[24] OHLHOFF, André: *Consistent Refinement of Sequence Diagrams in the UML 2.0*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, November 2006

[25] PLANTE, Fredric: *Introducing the GMF*. International Business Machines Corp., January 2006

[26] PROCHNOW, Steffen: *Dynamic charts, Oberseminar*. Department of Computer Science and Applied Mathematics, Real-Time Systems and Embedded Systems Group, University of Kiel, January 2004

[27] PROCHNOW, Steffen: *Efficient Developement of Complex Statecharts*, Christian-Albrechts Universität zu Kiel, Faculty of Engineering, Diss., 2008

[28] PROCHNOW, Steffen ; HANXLEDEN, Reinhard von: Visualisierung komplexer reaktiver Systeme - Annotierte Bibliographie / Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group. 2004 (0406). – Forschungsbericht

[29] PROCHNOW, Steffen ; HANXLEDEN, Reinhard von: Statechart Development Beyond WYSIWYG. In: *Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07)*. Nashville, TN, USA, October 2007

[30] PROCHNOW, Steffen ; HANXLEDEN, Reinhard von: The Use of Complex Stateflow-Charts with KIEL - An Automotive Case Study. In: *Proceedings of 5th GI-Workshop Automotive Software Engineering (ASE'07)*. Bremen, Germany, September 2007

[31] PROCHNOW, Steffen ; SCHAEFER, Gunnar ; BELL, Ken ; HANXLEDEN, Reinhard von: Analyzing Robustness of UML State Machines. In: *Proceedings of the Workshop on Modeling and Analysis of Real-Time and Embedded Systems (MARTES'06), held in conjunction with the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006)*. Genua, Italy, October 2006

[32] PROCHNOW, Steffen ; TRAULSEN, Claus ; HANXLEDEN, Reinhard von: Synthesizing Safe State Machines from Esterel. In: *Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*. Ottawa, Canada, June 2006

[33] SCHAEFER, Gunnar: *Statechart Style Checking*, Christian-Albrechts Univer-

sität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, June 2006

[34] SCHIPPER, Arne: *Layout and Visual Comparison of Statecharts*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, December 2008

[35] SCHIPPER, Arne ; FUHRMANN, Hauke ; HANXLEDEN, Reinhard von: Visual Comparison of Graphical Models. In: *In Proceedings of the Fourth IEEE International Workshop UML and AADL, held in conjunction with the 14th International International Conference on Engineering of Complex Computer Systems (ICECCS'09)*. Potsdam, Germany, June 2009

[36] SCHMELING, Matthias: *An Eclipse-Editor for Safe State Machines*. Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Studienarbeit, September 2009

[37] SPÖNEMANN, Miro: *On the Automatic Layout of Data Flow Diagrams*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, March 2009

[38] STOREY, M.-A. D. ; FRACCHIA, F. D. ; MÜLLER, H. A.: *Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization*. 1997

[39] STOREY, Margaret-Anne D. ; FRACCHIA, F. D. ; MÜLLER, Hausi A.: *Customizing a Fisheye View Algorithm to Preserve the Mental Map*. 1999

[40] TORY, Melanie ; MÖLLER, Torsten: Human factors in visualization research. In: *IEEE Transactions on Visualization and Computer Graphics* 10 (2004), January/February, Nr. 1, S. 72–84

[41] VÖLCKER, Jonas: *A quantitative analysis of Statechart aesthetics and Statechart development methods*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, May 2008

[42] WARE, Colin ; PURCHASE, Helen ; COLPOYS, Linda ; MCGILL, Matthew:

Cognitive measurements of graph aesthetics. In: *Information Visualization* 1 (2002), S. 103–110

[43] WISCHER, Mirko: *Textuelle Darstellung und strukturbasiertes Editieren von Statecharts*, Christian-Albrechts Universität zu Kiel, Department of Computer Science, Real-Time and Embedded Systems Group, Diplomarbeit, Februar 2006

[44] WONG, Pak C. ; MACKEY, Patrick ; PERRINE, Ken ; EAGAN, James ; FOOTE, Harlan ; THOMAS, Jim: Dynamic Visualization of Graphs with Extended Labels. In: *IEEE Symposium on Information Visualization* Pacific Northwest National Laboratory and Georgia Institute of Technology, 2005

Abbildungsverzeichnis

Quelltextverzeichnis