Christian-Albrechts-Universität zu Kiel

Studienarbeit

Konzepte zur besseren Visualisierung grafischer Datenflussmodelle

cand. inform. Steffen Jacobs

21. Februar 2007

Institut für Informatik
Arbeitsgruppe Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:

Dipl.-Inf. Hauke Fuhrmann
Dipl.-Inf. Steffen H. Prochnow

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,

_____

Zusammenfassung

Die modellbasierte grafische Entwicklung eingebetteter Systeme und Regelmechanismen in CASE-Werkzeugen ist heutzutage gängige Praxis. Besonders die Semantik der Datenflusssprachen und damit Werzeuge wie SCADE Suite von Esterel Technologies, MathWorks' Matlab/Simulink und IBM Rational RoseRT haben sich für diese Anwendungen etabliert. Allerdings ist die Handhabung und Visualisierung der Modelle innerhalb der Werkzeugumgebungen häufig umständlich und schöpft nicht das gesamte Potential grafischer Sprachen aus, da oft hoher kognitiver Aufwand bei Navigation und der Objektplatzierung anfällt.

In dieser Arbeit werden daher Konzepte vorgestellt, die einen effizienteren Umgang mit Datenflussmodellen ermöglichen sollen. Insbesondere werden Ideen für die Realisierung einer semantischen Fokus-und-Kontext-Technik für Datenflussmodelle erarbeitet. Das damit verbundene Problem eines automatischen Layouts von Modellen wird ebenfalls in Grundzügen diskutiert. Neben der statischen Visualisierung von Modellen wird auch das Prinzip einer dynamischen Darstellung während der Simulation vorgestellt. Vor allem die Idee einer automatischen Fokussierung durch sogenannte *trigger events* stellt einen vielversprechenden Ansatz dar.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung und Motivation

Anwendungen in eingebetteten Systemen erreichen heutzutage eine enorme Komplexität, so dass die traditionellen Techniken der Software-Entwicklung nicht mehr ausreichen. Die modellbasierte grafische Entwicklung von Systemen mit der Fähigkeit der Entwicklung simulierbarer Prototypen ist daher ein Schlüssel zur Einhaltung der hohen Qualitäts- und Robustheitsvorgaben einerseits sowie von zeitlichen Randbedingungen an die Entwicklung andererseits. Grafische Repräsentationen entsprechen der Neigung des menschlichen Wahrnehmungsapparats, visuelle Informationsprozesse und Repräsentationsformen zu bevorzugen. Eine Annahme ist, dass sich grafische Darstellungen positiv auf die Gedächtnisleistung und die menschliche Informationsaufnahme auswirken. Daher haben sich in der Industrie visuelle Datenflusssprachen etabliert, um Regelmechanismen und datengesteuerte Anwendungen zu entwickeln. In Blockdiagrammen werden die Komponenten und der Datenaustausch eines Gesamtsystems spezifiziert und möglichst intuitiv auf der Benutzeroberfläche repräsentiert. Die Vorteile einer grafischen Entwicklung liegen in ihrer guten Les- und Wartbarkeit sowie in ihrem hohen Abstraktionsniveau. Durch die in den CASE-Werkzeugen (*Computer Aided Software Engineering*) integrierte automatische Synthese eines Modells in plattformspezifischen Code lassen sich Systeme darüber hinaus mit geringem Programmieraufwand entwerfen. Allerdings sind heutzutage auch komplexe Systeme mit mehreren hundert oder tausend Blöcken und einer Vielzahl von Datenflüssen keine Seltenheit, wodurch die Anforderungen an grafische Entwicklungswerkzeuge steigen. Zusätzlich hängt das Verständnis einer grafischen Darstellung stark von den Navigations- und Visualisierungsmöglichkeiten der Entwicklungumgebung ab, so dass Detailwissen und globale Kontextinformation zu konkurrierenden Zielen werden. Heutige Werkzeuge stoßen daher häufig an ihre ergonomischen Grenzen bei der Darstellung komplexer Datenflussdiagramme.

Diese Arbeit versucht, geeignete Konzepte zur besseren Darstellung von Datenflussmodellen vorzustellen. Dabei werden vorhandene Techniken aus unterschiedlichen Disziplinen zusammengetragen und die Anwendung für Datenflussmodelle diskutiert. Neben besseren Visualisierungs- und Navigationskonzepten werden im Sinne eines computer-gestützten Designs die Möglichkeiten zur besseren computergestützten Visualisierung diskutiert. Dazu gehört das automatische Layout eines Modells, sowie die computergestützte Bestimmung des Fokus, z. B. während einer Simulation.

Kapitel 2 stellt zunächst die in der Industrie häufig genutzten Werkzeuge MathWorks Matlab/Simulink (im nachfolgenden nur Simulink genannt), Esterel Technologies

SCADE (SCADE) und IBM Rational Rose RealTime (RoseRT) vor, zeigt die Grundelemente der drei Werkzeuge und stellt Gemeinsamkeiten und Besonderheiten heraus. Die Handhabung der Navigation innerhalb der Werkzeuge, die Möglichkeiten innerhalb einer Simulation und Probleme bei der Visualisierung werden ebenfalls diskutiert.

In Kapitel 3 werden Konzepte zur Verbesserung der Visualisierung von grafischen Datenflusssprachen vorgestellt. Schwierigkeiten bei der Implementierung werden dabei zunächst außer acht gelassen. In erster Linie wird Bezug auf Fokus-und-Kontext-Techniken genommen, die hier als geeigneter Kandidat für Datenfluss-Visualisierungen gesehen werden. Darüber hinaus wird eine mögliche Kombination von grafischer und textueller Darstellung diskutiert.

Als Basis für spätere automatische Navigations- und Fokus-und-Kontext-Techniken wird in Kapitel 4 das Layout von Datenflussmodellen diskutiert. Ebenfalls werden einige Regeln vorgestellt, die zur Bildung einer Sekundärnotation beitragen. Diese dient als Voraussetzung für automatisches Layout.

Kapitel 4.4 beschäftigt sich mit der Frage, inwiefern eine computergestützte Fokusbestimmung, d.h. eine automatische Bereitstellung einer relevanten Sicht eines Modells realisierbar ist.

Das letzte Kapitel (5) stellt Schnittstellen der behandelten Werkzeuge vor. Insbesondere die Möglichkeiten zur Integration der Techniken in die vorhandenen Werkzeuge steht hier im Vordergrund.

2 Eine Untersuchung vorhandener Werkzeuge für Datenflussmodelle

Nach Lee und Messerschmitt [25] sind Datenfluss-Modelle gerichtete Graphen, wobei jeder Knoten einen Prozess oder eine Funktion und jede Kante einen Datenstrom repräsentiert. Das Datenfluss-Prinzip besagt, dass ein Knoten seine Berechnungen nur dann durchführen kann (er *feuert*), sobald die Daten seiner eingehenden Kanten verfügbar sind. Ein Knoten ohne eingehende Kanten darf jederzeit feuern. Dieses Prinzip impliziert, dass durchaus viele Knoten gleichzeitig ihre Berechnungen durchführen dürfen. Da ein Programm-Ablauf stark von der Verfügbarkeit der Daten abhängt, bezeichnet man Datenfluss-Programme auch häufig als daten-gesteuert (*data-driven*). Ein Spezialfall von Datenfluss bildet der *synchrone* Datenfluss, bei dem die Anzahl der konsumierten und produzierten Daten pro Knoten und Aufruf *a priori* festgelegt ist.

Obwohl diese allgemeine Beschreibung gut die Datenflussdiagramme klassifiziert, haben sich unterschiedliche Dialekte entwickelt. In diesem Kapitel werden drei gängige Werkzeuge vorgestellt und syntaktische Gemeinsamkeiten und Unterschiede erläutert. Insbesondere werden mögliche Problemfälle und Spezialitäten der einzelnen Tools diskutiert. Dies dient als Basis für die spätere Entwicklung von Techniken zur Verbesserung und Erweiterung der Visualisierung und ist Grundvoraussetzung für ein automatisches Layout.

2.1 Matlab Simulink

Basierend auf dem Werkzeug *Matlab* entwickelte das Unternehmen *The Mathworks* die Entwicklungsumgebung *Simulink* [29]. Es handelt sich hierbei nicht um ein eigenständiges Programm, sondern um einen Teil der Matlab *Toolbox*, welches etliche Zusatzpakete zu Matlab enthält. Ein Datenflussmodell, bestehend aus hierarchischen Boxen, den sogenannten *Subsystems*, sowie Ports und Konnektoren zwischen diesen, kann mit Hilfe des *Real-Time-Workshops* in ausführbaren Code synthetisiert werden. Zusätzlich existiert die Möglichkeit der Simulation eines Modells innerhalb des Werkzeugs. Eine große Bibliothek bereits angelegter Operatoren und Funktionen

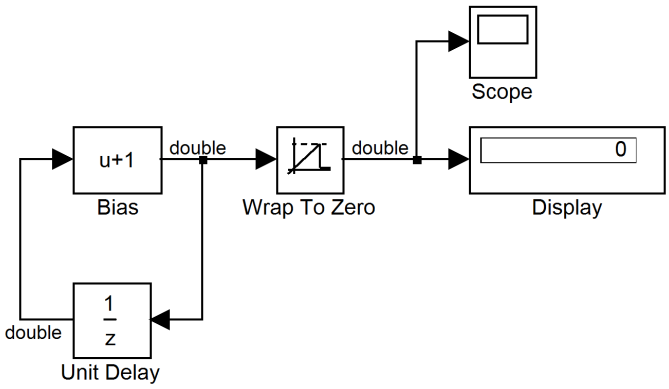


Abbildung 2.1: Einfaches Beispiel eines Datenflussmodells in *Simulink*

und die gute Erweiterbarkeit zeichnet dieses Werkzeug aus. Darüber hinaus sind viele – meist kostenpflichtige – Plug-Ins erhältlich. In Anlehnung an die physikalischen Eigenschaften von elektronischen Schaltkreisen, werden Daten über die Konnektoren („Drähte“) verschickt, sobald ein Signal anliegt. Simulink verwendet hier zwei unterschiedliche Auswertungsmodelle: Beim *diskreten* Modell geschieht die Auswertung der Elemente in festgelegten (benutzerdefinierten oder variablen) Zeitintervallen. Bei einem *stetigen* Modell geschieht dies kontinuierlich. Ein Beispiel für ein Datenflussmodell in Simulink ist in Abbildung 2.1dargestellt. Da Datenfluss in der Regel nicht geeignet ist, um das Ereignis-gesteuerte Verhalten eines Systems nachzubilden, bietet Simulink die Möglichkeit der Einbettung von Statecharts nach dem *Stateflow*-Dialekt in das Modell, sowie unterschiedliche Schnittstellen zu textuellen Sprachen wie Matlab-Script, `Java` und `C/C++`.

2.2 SCADE

Eine weitere CASE-Anwendung ist das Werkzeug *SCADE* (Safety Critical Application Developement Environment)[10]. Basierend auf der formalen, synchronen, textuellen Datenfluss-Sprache *Lustre* [**?**] bietet es die Möglichkeit der Transformation von Modellen in `C` oder Ada. Dabei wird der DO-178B-Standard [35] bis zu Level A erfüllt. Aus diesem Grund bietet sich SCADE speziell zur Entwicklung sicherheitskritischer Anwendungen von Systemen der Avionik und der Automobilindustrie an. Das Werkzeug bietet außerdem die Möglichkeit des Imports von Modellen aus Matlab/Simulink.

2.3 RoseRT

Die Entwicklungsumgebung *Rational Rose Real-Time* (RoseRT [33]), dessen Entwicklungsfirma *Rational* von *IBM* aufgekauft wurde, ist das dritte der hier vorgestellten Werkzeuge. Zur Visualisierung unterstützt es die grafische Notation der *Unified Modeling Language 2.0* (UML2.0 [40]) fast vollständig. Obwohl es sich auf den ersten Blick stark von den anderen beiden Designwerkzeugen unterscheidet wurde es in diese Arbeit aufgrund folgender relevanter Punkte aufgenommen: *UML* hat sich als Quasi-Standard zur Beschreibung von Softwaresystemen etabliert und genießt daher hohe Popularität. Darüber hinaus hat das *Capsule Structure Diagram* große Ähnlichkeit zu SCADE und Simulink und ist zur automatischen Codesynthese geeignet. In RoseRT werden grundsätzlich vier Sichten eines Modells unterschieden:

Use-Case-View: dient zur Visualisierung unterschiedlicher Anwendungsfälle des Systems; beschreibt, *was* das System macht, aber nicht *wie*.

Logical-View: Entwicklung des Systems in allen Einzelheiten; visualisiert, *wie* das System funktioniert.

Component-View: visualisiert, wie die unterschiedlichen Teile des Systems compilliert und welche Bibliotheken benötigt werden.

Deployment-View: beschreibt die Architektur der Umgebung in der das System ausgeführt wird.

Die Entwicklung findet überwiegend in der *Use-Case-View* und der *Logical-View* statt. Obwohl ein Modell aus unterschiedlichsten Diagrammtypen bestehen kann, beschränkt sich diese Arbeit auf die Darstellung von *Collaboration*-Diagrammen in RoseRT. In diesen Diagrammtypen kommunizieren *Capsules* durch Nachrichtenaustausch über Ports miteinander. Dabei werden Konnektoren benutzt, um einen Port der einen Capsule mit einem Port einer anderen Capsule zu verbinden. Die Definition von zeitgesteuerter und synchroner Kommunikation, wie es in SCADE und Simulink praktiziert wird, spielt in RoseRT nur eine untergeordnete Rolle. Echtzeitfähiges Verhalten wird dadurch nur bedingt realisierbar. Das Systemverhalten wird in der Regel durch Statecharts, Sequenzdiagramme und/oder Code gesteuert. Da der verwendete Statechart-Dialekt kein Konstrukt zur Modellierung eines Echtzeitsystems bietet und die Sequenzdiagramme (im Gegensatz zum UML2.0-Standard) keine Angabe von Zeitintervallen erlauben, ist der Einsatz für Systeme mit zeitrelevantem Verhalten schwierig. Als einziges Konstrukt bietet sich hierfür ein Timing-Port an, dessen *timeout*-Events in den Diagrammen abgefragt werden können. Ein einfaches Beispiel der Realisierung eines *Ping-Pong*-Spiels wird in Abbildung 2.2 gezeigt. In jedem der beiden dargestellten Capsules wird ein Statechart-Diagramm zur Realisierung des Verhaltens eingesetzt. Die Nachrichtengenerierung und der Austausch finden nach dem folgenden Prinzip statt:

> *„Sobald die Nachricht* ping *(bzw.* pong*) empfangen wird sende* pong *(bzw.* ping*) und warte eine Sekunde.“*

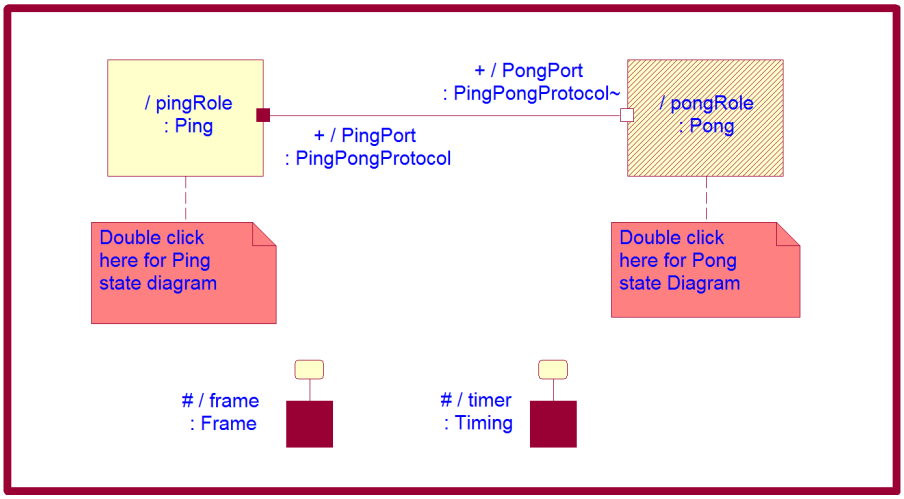


Abbildung 2.2: Realisierung einer *Ping-Pong*-Simulation in RoseRT [33]

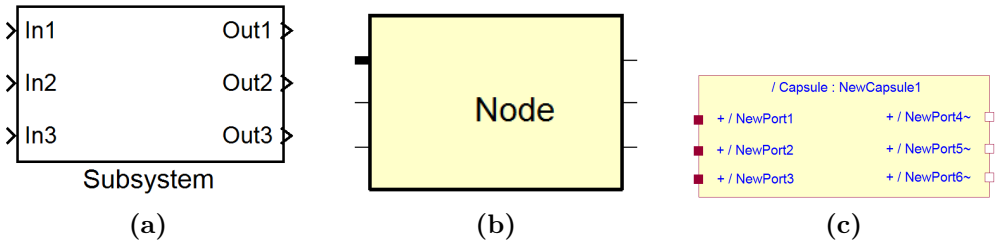


Abbildung 2.3: Komponenten zur Kapselung von Teilsystemen in Simulink (a), SCADE (b) und RoseRT (c)

2.4 Visualisierung der Elemente

Alle Werkzeuge basieren auf den Grundelementen von Datenflusssprachen: die hierarchische Kapselung in grafischen Boxen (Capsules, Subsystems, Nodes) und die Darstellung des Datenflusses. Diese beiden Elemente werden auf unterschiedliche Weise in den Sprachen visualisiert.

2.4.1 Box

Eine Box beschreibt ein abgeschlossenes Teilsystem eines Modells. Sie ist eine hierarchische Komponente, d.h. ihr inneres Verhalten kann rekursiv ebenso durch ein Modell beschrieben werden. In Abbildung 2.3 wird gezeigt, wie die Boxen im jeweiligen Werkzeug dargestellt werden. Grafisch besteht jede Box aus einer rechteckigen Begrenzung (der sog. *Bounding-Box*), die zur Abgrenzung des Objekts und der Interaktion durch den Benutzer dient. Dadurch wird das Element selektier-, platzier- und

	RoseRT	SCADE	Simulink
Box			
Skalierung	+	+	+
90 Grad Drehung	−	+	+
stufenlose Drehung	−	−	−
Spiegelung	−	+	−
Veränderung in Form, Farbe	−	+	+
Vektorgrafik als Symbol	−	+	+
Rastergrafik als Symbol	−	+	+
Label			
vorhanden	+	+	+
Position veränderbar	−	+	+
benutzerdefinierter Name	+	+	+
benutzerdefinierte Schriftart	−	+	+
Anzeige der Kardinalität	+	+	−

Tabelle 2.1: Gemeinsamkeiten/Unterschiede der Box-Visualisierung

skalierbar. Zusätzlich zur Begrenzung, enthält jede Box ein grafisches Symbol und ein Label. Veränderungen am Erscheinungsbild erlauben die Werkzeuge in unterschiedlichen Ausprägungen. Den größten Komfort bietet SCADE, welches in einem speziellen Editier-Modus fast jede beliebige Form zulässt. Etwas weniger Komfort bietet Simulink. Mit Hilfe einer sogenannten *Maske*, die für jede Box definiert werden kann, können Pixelgrafiken, Vektorlinien und/oder Texte auf das Element platziert werden. Auch hier kann – mit höherem Aufwand – fast jedes beliebige Objekt gezeichnet werden. Lediglich RoseRT bietet keine dieser Funktionen. Die Frage nach der Notwendigkeit solcher Möglichkeiten bleibt hier vorerst unbeantwortet. Um die Anordnung der Elemente auf der Arbeitsfläche zu erleichtern, bieten die Werkzeuge zusätzlich die Möglichkeit, Boxen zu drehen und zu spiegeln. So lassen sich Datenflussmodelle ohne unnötige Kantenkreuzungen modellieren.

Die Beschriftung einer Box – das Boxlabel – kann beliebig platziert werden (SCADE), richtet sich an Ankerpunkten oberhalb bzw. unterhalb der Box aus (Simulink) oder ist gar nicht manuell positionierbar (RoseRT). Die Umsetzung der einzelnen Box-Eigenschaften innerhalb der Werkzeuge ist in Tabelle 2.1 noch einmal zusammengefasst.

Port Der Datenaustausch der Teilsysteme mit Komponenten in ihrer Umgebung erfolgt durch definierte Schnittstellen – den *Ports*. Datenleitungen zwischen den Ports ordnen jedem Eingangs- einen Ausgangsport zu. Andere Verbindungen (z. B. Eingangs- mit Eingangsport) sind nicht zugelassen. Während bei SCADE und Simulink jedem Port nur ein Signal zugeordnet wird und die Trennung von Eingangs-

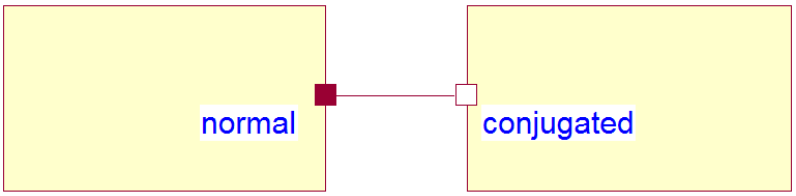


Abbildung 2.4: *normal* und *conjugated port* in IBM Rational Rose RealTime

und Ausgangsport strikt ist, bildet RoseRT hier die Ausnahme. Jeder Port kann – definiert durch spezielle Protokolle – eine beliebige Anzahl unterschiedlicher Daten senden und empfangen. Dieses Konstrukt erlaubt einen flexibleren Umgang mit Daten, erschwert aber die Lesbarkeit des Modells, da nicht alle vorhandenen Datenflüsse visualisiert werden. RoseRT unterscheidet zwei unterschiedliche Arten von Ports:

public port Schnittstelle zur Kommunikation zwischen Boxinhalt und -umgebung.

private port Nur sichtbar innerhalb der Box. Verwendung z. B. als Zeitgeber durch einen Timing-Port

Darüber hinaus gibt es normale und konjugierte (*conjugated*) Ports, die sich auch visuell unterscheiden (siehe Abbildung 2.4). Ein konjugierter Port übernimmt die Signale eines verbundenen normalen Ports mit vertauschten Rollen für Eingangs- und Ausgangssignale. Auf der gleichen Hierarchie-Stufe können nur normale mit konjugierten Ports verbunden werden, um das Senden und Empfangen von Nachrichten zu ermöglichen.

Ports können in SCADE und RoseRT durch den Benutzer positioniert werden. Während bei SCADE Ports beliebig (auch getrennt von der Box) platziert werden dürfen, wodurch mehr oder weniger sinnvolle Boxen entstehen können, beschränkt sich die Platzierung bei RoseRT auf das Innere (private port) bzw. auf den Rand der Box (public port). Im Gegensatz dazu sind bei Simulink die Port-Positionen festgeschrieben: Ein- und Ausgangsports liegen sich immer gegenüber. Mehrere Ports des gleichen Typs werden automatisch auf einer Boxseite gleichverteilt dargestellt.

Portlabel In allen drei Werkzeugen besitzt ein Port eine eindeutige Bezeichnung – das *Portlabel*. Im Gegensatz zu SCADE und Simulink kann das Label in RoseRT beliebig positioniert werden. Die Zugehörigkeit von Port und Label ist somit vom Betrachter nicht immer eindeutig festzustellen (siehe Abbildung 2.5). Der Leser muss darauf hoffen, dass bei der Entwicklung des Modells auf eine „sinnvolle“ Platzierung geachtet wurde. Auf der anderen Seite werden die Portlabels in Simulink und SCADE immer direkt links bzw. rechts des Ports platziert. Die Platzierung von Ports und Portlabel wird ausführlicher in Kapitel 4 erörtert. Das Ein- bzw. Ausblenden der Portlabel ist bei beiden Werkzeugen erlaubt während eine Änderung des Labels

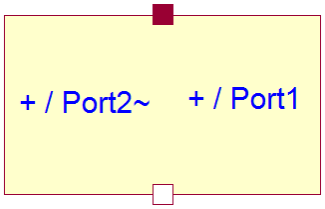


Abbildung 2.5: Die Zuordnung von Label und Port ist in RoseRT nicht immer eindeutig möglich.

	RoseRT	SCADE	Simulink
Port			
Position veränderbar	+	+	−
Aussehen veränderbar	−	−	−
Unterscheidung von In-/Output	−	+	+
Label			
Position veränderbar	+	−	−
benutzerdefinierter Name	+	+	+
benutzerdefinierter Font	−	−	+

Tabelle 2.2: Gemeinsamkeiten/Unterschiede der Port-Visualisierung

nur in Simulink und RoseRT beliebig möglich ist, wobei darauf geachtet werden muss, dass es sich um gültige Bezeichner der Zielsprache handelt, in die das Modell transformiert wird. Die Portbezeichnung von SCADE entspricht der Bezeichnung des Ausgangselements im Inneren der Box, ist also nicht unabhängig davon veränderbar. Die Eigenschaften der Port-Visualisierung sind in Tabelle 2.2 zusammengefasst.

Spezielle Ports In Simulink sowie in SCADE wird ein weiterer Port-Typ bei der Darstellung unterschieden. Ein Port, der nicht direkt dem Datenfluss des Modells zugeordnet werden kann, sondern eher die Rolle eines Parameters einnimmt, also das Verhalten der Box beeinflusst, wird orthogonal zu den Ein- bzw. Ausgangsports positioniert. In Simulink zählen dazu Boxen, in denen ein *Trigger*- bzw. ein *Enable*-Element platziert ist (s. Abbildung 2.6). So wird beispielsweise bei einem *enabled Subsystem* die Berechnung innerhalb der Box nur ausgeführt, wenn am oberen Eingangsport ein boolesches `true` anliegt, ansonsten wird der vorherige Wert ausgegeben. Ein ähnliches Verhalten kann in SCADE mit einem speziellen Boxtyp erreicht werden. Zusätzlich existieren in SCADE sogenannte *Hidden Ports*. Diese dienen als reine Parameter und bieten zwei Alternativen: zum einen kann an einen Hidden Port ein Operator über eine Datenflussleitung angeschlossen werden. Zum Anderen (falls an dem Hidden Port kein Element anliegt) muss durch Aufruf eines Kontextmenüeintrags der Box der Parameterwert manuell eingegeben werden. Im

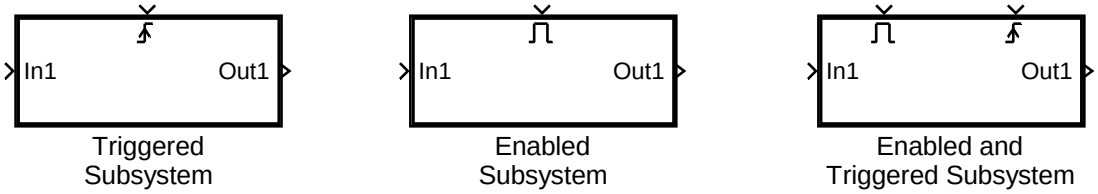


Abbildung 2.6: Einbettung von *Enabled/Triggered*-Komponenten in Subsystems

zweiten Fall bleibt der Hidden Port unbelegt.

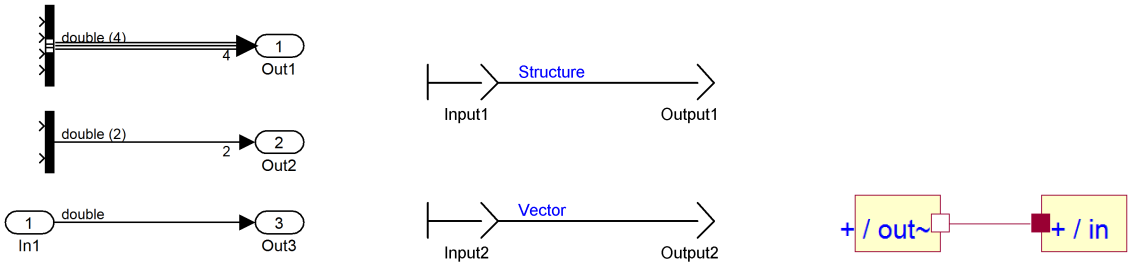
Die unterschiedlichen Arten der Portdarstellung und Positionierung in den Werkzeugen zeigt, dass die Berücksichtigung aller Spezialfälle in einem automatischen Layout nicht trivial ist. Selbst das Hinzufügen von nur einem weiteren Port kann in einem ungünstigen Fall zu einer kompletten Umordnung aller anderen Objekte führen, was in der Regel nicht gewünscht ist.

2.4.2 Datenfluss

In den Werkzeugen wird der Datenaustausch zwischen Boxen durch Leitungen zwischen den Ports visualisiert. Das synchrone Datenflussmodell wie es in Simulink und SCADE genutzt wird, erzeugt kontinuierlich Daten. D.h. zu jedem (diskreten oder stetigen) Zeitschritt werden Daten produziert und konsumiert, sofern Datenquellen im Modell existieren: es entsteht ein regelmäßiger Datenstrom. Dabei ist zu beachten, dass Datenströme nur in eine Richtung fließen können: von einem Ausgangs- zu einem Eingangsport (*Inport* und *Outport*). Durch die Analogie zu einem elektrischen Stromkreis ergeben sich weitere intuitive Visualisierungen in Simulink und SCADE. So ist es möglich, ein Ausgangssignal auf einfache Weise zu duplizieren und somit an n weitere Boxen zu senden ($1 : n$-Verbindung).

Auf der anderen Seite verwendet RoseRT eine Ereignis-basierte Datengenerierung. D.h. ein Signal wird nur über eine Datenleitung geschickt, wenn das zugehörige Ereignis (innerhalb eines Statecharts oder Sequenzdiagramms) aufgetreten ist. Die Flussrichtung ist in diesem Fall nicht festgelegt. Jeder Port kann – wie oben beschrieben – über die gleiche Leitung Signale senden und empfangen. Im Gegensatz zu den beiden anderen Werkzeugen besteht immer nur eine $1 : 1$-Verbindung zwischen den Ports. D.h. ein Port ist mit genau einem anderen Port verbunden.

Obwohl sich die Semantik der Konnektoren von RoseRT zu den anderen Werkzeugen unterscheidet (s. o.), gibt es bei der Visualisierung der Datenfluss-Elemente kaum Unterschiede. In SCADE sowie in RoseRT werden sämtliche Verbindungen durch



(a) Simulink **(b)** SCADE **(c)** RoseRT

Abbildung 2.7: Visualisierung von Datenfluss

	RoseRT	SCADE	Simulink
Verbindungen			
Orthogonal	+	+	+
Linear	+	−	+
Splines	+	−	−
Datentypabhängige Darstellung	−	−	+
Label			
automatische Beschriftung	+	+	−
benutzerdef. Beschriftung	+	+	+
Position veränderbar	+	−	+

Tabelle 2.3: Gemeinsamkeiten/Unterschiede der Datenfluss-Visualisierung

eine einfache Linie dargestellt, unabhängig davon, ob es sich um einen zusammengesetzten oder einfachen Datentyp handelt. Beide Werkzeuge bieten die Möglichkeit, neue Datentypen zu definieren. Simulink unterstützt dagegen nur eine kleine Auswahl: es gibt Skalare (einfache Datentypen), Vektoren und Busse. Die Repräsentation von Skalaren ist eine einfache orthogonale Linie zwischen den Ports. Diese Darstellung ist analog zu der von RoseRT und SCADE. Ebenso verhält sich Simulink bei Vektoren, die aus einfachen Datentypen (Skalare) zusammengesetzt sind. Soll ein Vektor von Vektoren gesendet werden, wechselt Simulink die Darstellung wie in Abbildung 2.7a zu sehen.

Während SCADE ausschließlich orthogonale Verbindungen zulässt und damit nahe an den Standards von Platinenlayouts bleibt, lässt Simulink auch lineare Verbindungen, die in beliebigem Winkel zueinander stehen können, zu. In der Regel wird aber diese Darstellung von Entwicklern vermieden, da sich in einem orthogonalen Modell ein manuelles Layout erheblich leichter realisieren lässt. RoseRT bietet darüber hinaus sogar die Verwendung von Splines als Verbindungen zwischen den Ports. Die Eigenschaften bei der Datenfluss-Darstellung der einzelnen Werkzeuge ist noch einmal in Tabelle 2.3 zusammengefasst.

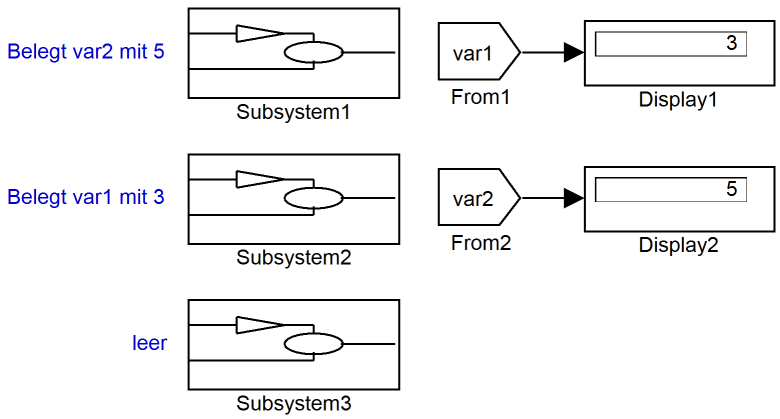


Abbildung 2.8: Problem bei Verwendung globaler Variablen durch GOTO- und FROM-Boxen in Simulink

Unsichtbarer Datenfluss SCADE bietet die Möglichkeit durch Definition lokaler Variablen, Daten zwischen Boxen innerhalb der gleichen Hierarchie zu versenden, ohne dass eine sichtbare Verbindung besteht. Dadurch werden unnötige Kantenkreuzungen, z. B. beim Verbinden weit auseinanderliegender Boxen vermieden und die Übersichtlichkeit wird nicht zerstört. Andererseits kann argumentiert werden, dass durch die fehlende visuelle Verbindung erhöhte kognitive Arbeit aufgebracht werden muss, um den semantischen Zusammenhang herzustellen.

Ein ähnliches Prinzip wird in Simulink durch sogenannte GOTO- und FROM-Boxen realisiert. Hinzu kommt, dass die Möglichkeit besteht, globale Variablen zu definieren, sowie Variablen, deren Gültigkeitsbereich beliebig definierbar ist (durch einen *Goto Tag Visibility Block*). D.h. eine GOTO- und die zugehörige FROM-Box müssen nicht mehr zwangsläufig – wie bei den lokalen Variablen in SCADE – in der gleichen Hierarchie existieren, sondern könnten unter Umständen über mehrere Hierarchien hinweg kommunizieren. Dem erhöhten Freiheitsgrad in der Gestaltung stehen einige Probleme bei der Analyse und dem Verständnis des Modells gegenüber. So ist es in Abbildung 2.8 nicht ersichtlich, in welchem Subsystem die Variablen var1 und var2 mit Werten belegt werden. Es ist nicht einmal ersichtlich, ob beide Subsysteme überhaupt Daten generieren, da sich die Darstellung von Subsystem1 und Subsystem2 nicht von der Visualisierung der leeren Box Subsystem3 unterscheidet. Dies hat zur Folge, dass der Betrachter die zugehörige Datenquelle suchen muss, indem er „auf gut Glück“ durch die Hierarchien der Subsysteme navigiert. Eine Möglichkeit zur Lösung dieses Teilproblems wäre die vollständige Vermeidung globaler Variablen, wie es in Kapitel 4.1 bei der Angabe von Ästhetikkriterien diskutiert wird. Weniger radikale Ansätze, die Probleme mit unsichtbarem Datenfluss abschwächen können, werden insbesondere in den Abschnitten 3.3.1 und 4.4.3 vorgestellt.

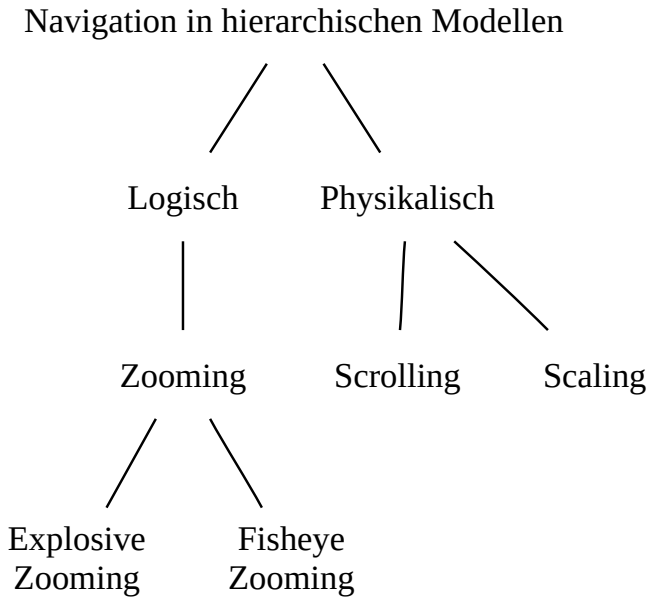


Abbildung 2.9: Typen der Navigation [3]

2.5 Navigation

Komplexe Modelle haben einen großen Platzbedarf und sind in der Regel in allen ihren Details schwer verständlich. Ein Modellierungswerkzeug sollte dem Benutzer die Möglichkeit der Navigation innerhalb des Modells bieten, um einerseits die Beschränktheit des Anzeigegerätes auszugleichen, aber auch um das Verständnis durch geeignete abstrahierte Sichten für den Leser zu erleichtern. Berner [3] unterscheidet zwei Arten von Navigation: Die *physikalische Navigation* und die *logische Navigation*. Physikalische Navigation wird dann notwendig, wenn die Arbeitsfläche (i.d.R. der Bildschirm) nicht mehr ausreicht, um das komplette Modell darzustellen (siehe auch Abbildung 2.9). Dann ist es erforderlich durch Größenänderung des Modells die Sicht zu ändern (skalieren) oder lediglich nur einen Ausschnitt der Modells darzustellen. Beide Möglichkeiten werden von allen drei Werkzeugen unterstützt. Um im zweiten Fall auch nicht dargestellte Bereiche des Modells sichtbar zu machen, bieten die Werkzeuge die standardisierte Ein-Fenster-Navigation durch Rollbalken (*scroll bars*) an. Diese Anpassung der Sicht findet auf der *Darstellungsebene* statt. Dadurch wird auch die Erstellung großflächiger Modelle möglich, ohne an die Beschränktheit der Arbeitsfläche gebunden zu sein.

Nach Brooks Jr. [4] ist die Navigation ein nicht-essentieller Teil einer Benutzeraufgabe. Ein Benutzer navigiert, um das Werkzeug zu veranlassen die gerade benötigte Sicht darzustellen. Das ist *nicht* die eigentliche Aufgabe, die der Benutzer ausführen will, sondern vielmehr eine notwendige Vorbereitung zur Durchführung dieser Aufgabe. Hierbei sind zwei grundlegende Aspekte der Navigation zu unterscheiden:

- kognitiver Aspekt
- mechanischer Aspekt

Während der kognitive Aspekt den mentalen Aufwand des Benutzers bezeichnet, der notwendig ist, um den aktuellen Fokus zu bestimmen und ihn zu ändern, bezeichnet der mechanische Aspekt den rein motorischen Aufwand – z.B. das Bewegen und Klicken mit der Maus – der aufgebracht werden muss. Der kognitive, nicht-mechanische Aufwand für Navigationsaktivitäten wird als *kognitiver Ballast (cognitve overhead)* bezeichnet [2]. Das Ziel ist es, den kognitiven Ballast zu minimieren. Im – nicht realisierbaren – Idealfall ist die Bestimmung des Fokusses frei von kognitivem Ballast. Das Werkzeug wäre in diesem Fall in der Lage den vom Benutzer gewünschten neuen Fokus grundsätzlich vorherzusagen bzw. zu raten.

Die hier diskutierten Werkzeuge verwenden zur Visualisierung der Darstellungsebene ein 1-Fenster-Konzept. Zusätzlich zur Rollbalken-Navigation lassen sich die Modelle skalieren (im Falle von Simulink ausschließlich durch Tastaturkommandos), um sie der Darstellungsfläche anzupassen. Allerdings muss bei diesem Konzept der Benutzer oft mit hohem kognitiven Aufwand navigieren, um sich immer wieder neu zu orientieren. Eine Lösung für dieses Problem kann ein Zwei-Fenster-Konzept sein, welches in Abschnitt 3.2 diskutiert wird. Im Gegensatz zur physikalischen Navigation verändert die logische Navigation den Informationsgehalt bezüglich der Hierarchie eines Modells. Betrachtet man hierarchische Konstrukte muss die Möglichkeit gegeben sein durch Änderung der *Darstellungstiefe* (DT, die Anzahl der Hierarchien, die ohne Persektivwechsel im Detail dargestellt werden) oder der *Darstellungsperspektive* (das perspektivische Wechseln der Hierarchieebene) die hierarchische Abstraktion aufzulösen (Berner [3]). Die Änderung der Darstellungsperspektive wird von allen drei Werkzeugen nur durch *explosives Zoomen (explosive zooming)* unterstützt. Bei diesem Verfahren wechselt der Fokus eines Betrachters in ein hierarchisches Element der Sprache (Box), wobei die bestehende Sicht komplett ersetzt wird; d.h. alle umgebenden Elemente werden ausgeblendet. Das Verwenden von explosivem Zoomen unterstellt, dass jedes Detail eines Modells gleich wichtig ist und entweder komplett oder gar nicht dargestellt wird. Dies hat allerdings zur Folge, dass der globale Kontext verloren geht, bzw. erhöhter kognitiver Aufwand nötig ist, um den Kontext zu bewahren.

2.6 Simulation

Eine Simulation dient in erster Linie dazu, das System (bzw. Modell) in einem experimentellen Umfeld zu testen und Erkenntnisse über das Verhalten zu gewinnen. In diesem Zusammenhang muss ein Simulationswerkzeug die Anforderung erfüllen, dem Entwickler möglichst alle nötigen Informationen übersichtlich zu präsentieren.

Dabei wird das Modell in einen realen (zeitlichen) Kontext eingebettet, und bezüglich seines Verhaltens im Laufe der Zeit bei gegebenen Parametern analysiert. Eine Simulationsumgebung sollte mindestens folgende drei Punkte unterstützen:

Datenvisualisierung: Ein Datenfluss-Modell visualisiert, wie Daten zwischen Teilsystemen ausgetauscht werden. Erst durch eine Instanziierung eines Modells in einer Simulation kann konkret dargestellt werden, welche Daten zu welchem Zeitpunkt ausgetauscht werden. Die Datenvisualisierung ist eine Kernaufgabe der Simulation. Das hohe Datenvolumen bei großen Modellen erfordert daher mächtige Visualisierungstechniken.

Zeitliche Kontrolle: Da das Erfassen sämtlicher Informationen eines simulierten Systems in Echtzeit nur schwer möglich ist, erfordert eine Simulationsumgebung die Möglichkeit der Kontrolle über die Zeit. Auf diese Weise lassen sich Simulationsgeschwindigkeiten variieren.

Dateninjektion: Um ein reaktives System zu testen, ist es wichtig, in einer experimentellen Umgebung Eingaben und Parameter ändern zu können. Insbesondere die Injektion von fehlerhaften Werten zeigt, wie robust das System tatsächlich ist.

Jedes der Werkzeuge bietet die Möglichkeit zur Simulation eines Modells, wobei sich die Umgebungen stark voneinander unterscheiden. Simulink bietet den geringsten Handlungsspielraum während der Simulation. Im Rahmen einer benutzerdefinierten Ausführungsdauer wird die Berechnung der Simulation schnellstmöglich durchgeführt. Bei ausreichend großen Modellen und einer entsprechend langsamen Berechnung ist es möglich, die Simulation zu pausieren, um Daten zu einem bestimmten Zeitpunkt zu betrachten. Dann ist auch eine Veränderung der Daten und Parameter möglich. Zur Datenvisualisierung bietet Simulink spezielle Elemente aus der Bibliothek an. Sogenannte *Scopes* werden – analog zu Messinstrumenten aus der Elektrotechnik – mit Konnektoren verbunden, so dass nach Abschluss bzw. während der Simulationsberechnung der Datenflusswert über die Zeit dargestellt wird (siehe Abbildung 2.10). Dieser Graph wird in einem separaten Fenster angezeigt, wobei ein Scope-Fenster auch mehrere Scopes beinhalten kann. Mit Scopes bietet Simulink ein Konstrukt zur Datenvisualisierung an, dass zwar intuitiv in das Modell integrierbar ist, aber hohe kognitive Anstrengung des Benutzers erfordert, um eine Verbindung zwischen eigentlichem Inhalt und der Visualisierung eines Konnektors herzustellen. Darüber hinaus wird das Modell durch Elemente erweitert (und verbunden), die eigentlich keine echten Bestandteile des Modells darstellen. Dies ist abträglich für das Verständnis und die Übersicht des Modells. Allerdings ist dieses Problem in Simulink bekannt, so dass mit den sogenannten *Floating Scopes* auch Daten angezeigt werden können, ohne dass ein solches Scope explizit mit dem Modell verbunden werden muss.

Nicht zuletzt verursacht die Darstellung aller Scopes in separate Fenster zusätzlich zum erhöhten kognitiven Ballast einen hohen mechanischen Aufwand während

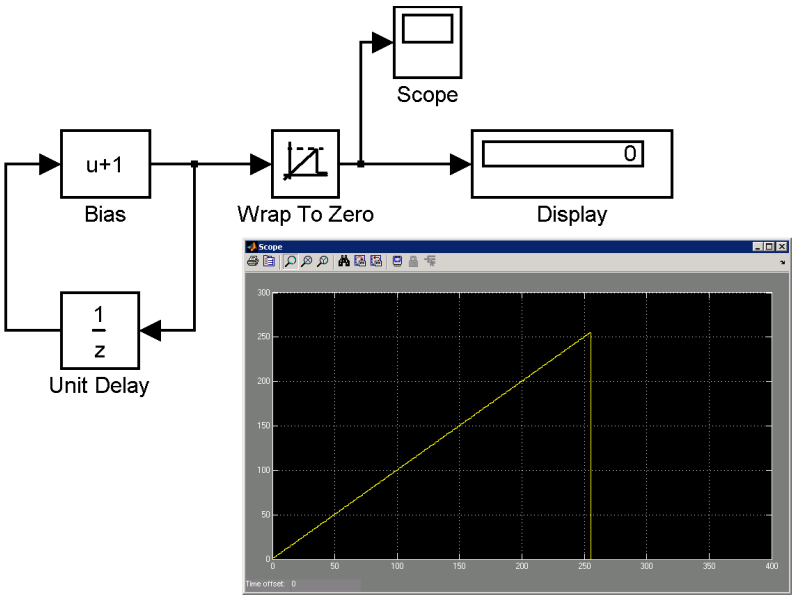
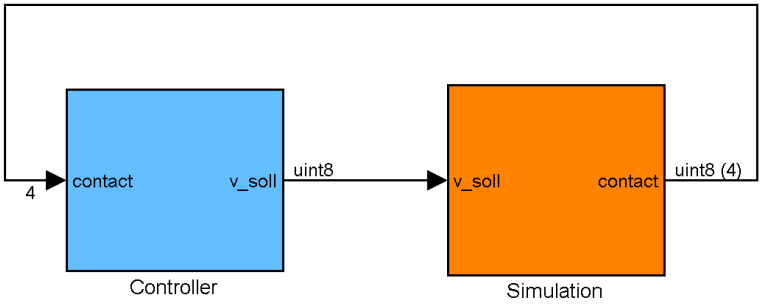


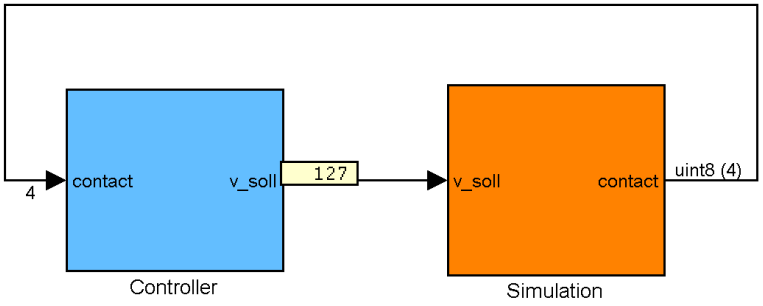
Abbildung 2.10: Datenvisualisierung in Simulink durch Scopes.

der Navigation. Eine Alternative wäre es, die Daten an jedem Konnektor direkt anzuzeigen, um so die Zugehörigkeit zwischen einer Datenleitung im Modell und den gesendeten Werten darzustellen. Durch die sogenannte *Show-When-Hovering*-Funktion ist es in Simulink möglich, alle Ausgangssignale einer Box anzeigen zu lassen, sobald der Mauszeiger über dem entsprechenden Objekt „schwebt“ (*hover*, siehe Abbildung 2.11). Dieses Hilfsmittel steht sowohl im Editiermodus als auch während der Simulationsberechnung zur Verfügung, so dass es als Ersatz für Scopes dienen kann. Allerdings zeigt das Beispiel in Abbildung 2.12 deutlich die Grenzen dieser Visualisierungstechnik: Größere Datenmengen (in Vektoren, Arrays oder Bussen) sind nur beschränkt bzw. gar nicht anschaulich darstellbar, da sie mehr Platz benötigen als die Arbeitsfläche zur Verfügung stellt oder Überdeckungsanomalien auftreten, so dass der Benutzer keine sinnvollen Informationen erhält und die Darstellung ihn verwirrt. Neben diesem Problem erschwert das Fehlen einer Einzelschrittausführung die konstruktive Analyse der Simulationsdaten.

Anders als in Simulink erhält die Arbeitsfläche in SCADE in einer eigenen Simulationsumgebung zwei zusätzliche Fenster zur Datenbetrachtung (siehe Abbildung 2.13). Während im linken Fenster für ausgewählte Input- und Output-Boxen ein Graph über die Zeit erstellt wird (analog zu den Scopes in Simulink), können im rechten Fenster die aktuellen Werte der hinzugefügten Daten abgelesen werden. Auf diese Weise werden sämtliche Informationen auf der gleichen Arbeitsfläche dargestellt, ohne dass zusätzlicher kognitiver Aufwand für die Navigation nötig ist. Nachteilig in SCADE ist das Fehlen einer Navigation innerhalb der Graph-Sicht. So ist es nur möglich, entweder den Graphen über die gesamte Zeit vollständig darzustellen oder jeweils nur einen Teil. In SCADE wurde die zweite Möglichkeit gewählt, wodurch

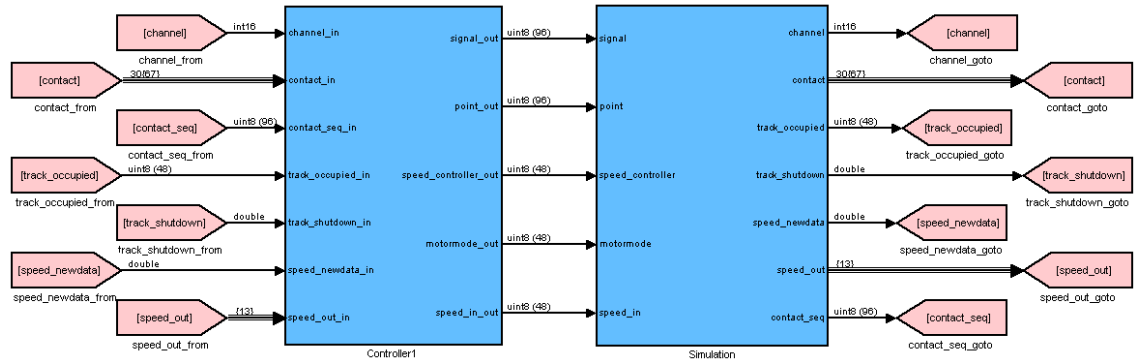


(a)

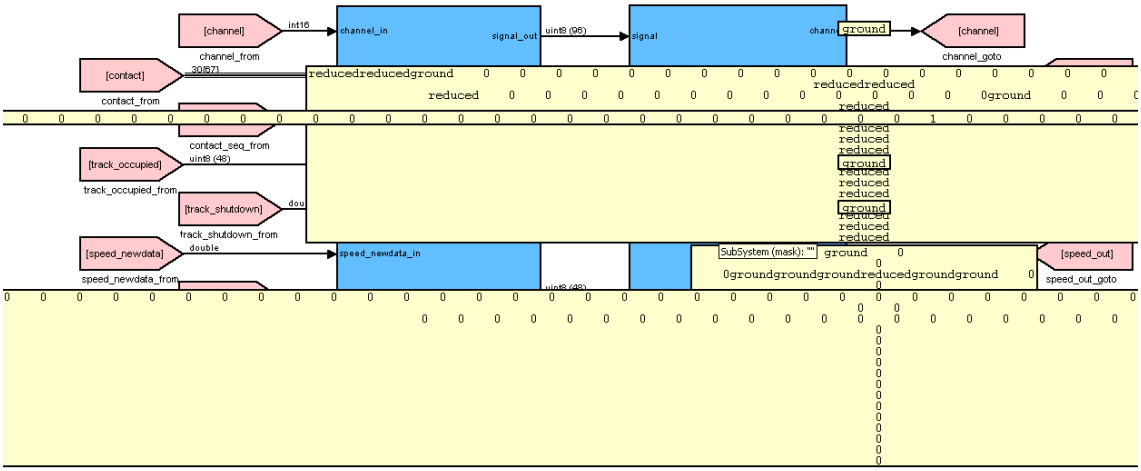


(b)

Abbildung 2.11: Direkte Anzeige der Daten im Modell mit aktivierter *Show-When-Hovering*-Funktion in Simulink ohne Fokussierung (a) und mit Fokus auf der Controller-Box (b)



(a)



(b)

Abbildung 2.12: Anzeige der Daten direkt im Modell mit aktivierter *Show-When-Hovering*-Funktion; (a) ohne aktuellem Fokus, (b) mit Fokus auf dem Simulationsblock

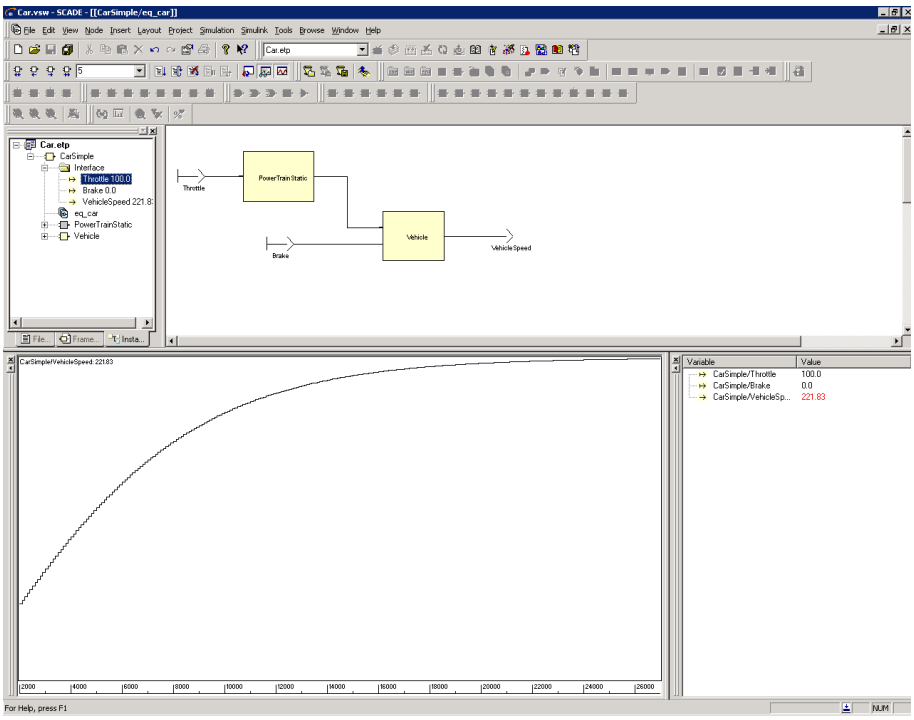


Abbildung 2.13: Simulationsumgebung in SCADE

schon nach relativ kurzer Simulationszeit der linke Teil abgeschnitten wird, wie in Abbildung 2.13 zu sehen ist. Aber auch eine Darstellung des kompletten Graphen über die gesamte Zeit wäre nur durch Stauchen der Grafik möglich, so dass bei längerer Simulation Detailinformationen verloren gehen würden. Ähnlich wie bei der *Show-When-Hovering*-Funktion in Simulink werden die Daten in SCADE auch direkt an den Output-Ports der Boxen angezeigt. Dabei ergeben sich bei komplexen Datenstrukturen allerdings auch die gleichen Probleme.

RoseRT besitzt ebenfalls wie SCADE eine eigene Simulationsumgebung, in der sowohl eine Einzelschritt- als auch eine sofortige Simulation durchführbar ist. Die Navigation innerhalb des Modells ist weiterhin möglich, allerdings werden in diesen Sichten keine Simulations-Aktivitäten dargestellt. Stattdessen exisitieren in der Simulationsumgebung für jedes Box-Element zwei zusätzliche Kontrollsichten: *State Monitor* und *Structure Monitor*, die auf die Daten der Simulation zurückgreifen und aktuelle Zustände der Statecharts sowie Datenflüsse zeigen (siehe Abbildung 2.14). Soll der Nachrichtenaustausch der Protokolle zwischen den Capsules angezeigt werden, muss einer der beteiligten Ports ausgewählt und ihm eine sogenannte *Probe* zugeordnet werden. Diese „horcht“ die gesendeten und empfangenen Nachrichten ab und listet sie in einem separaten Fenster auf.

Tabelle 2.4 zeigt zusammengefasst alle wichtigen Funktionen zur Visualisierung während der Simulation und welche Werkzeuge sie unterstützen. Auffällig ist, dass keines der Werkzeuge sämtliche Funktionen zur Verfügung stellt.

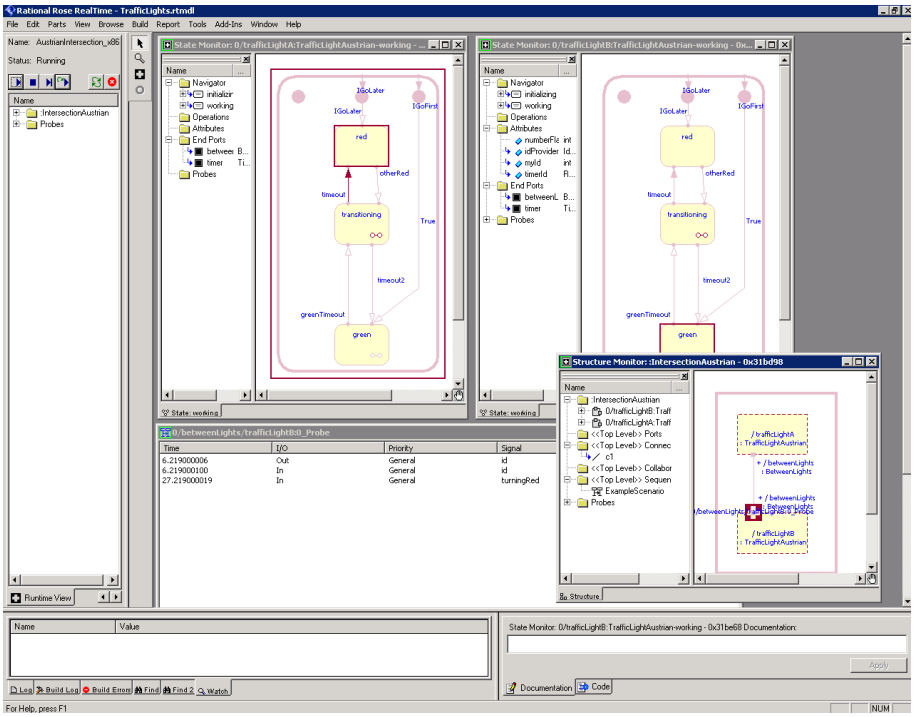


Abbildung 2.14: Simulationsumgebung in RoseRT: Beispiel einer Ampelsimulation (Quelle: [33])

	RoseRT	SCADE	Simulink
Inputs/Outputs veränderbar	+	+	+
Fehlerinjektion	+	−	+
Scopes	−	+	+
Einzelschrittsimulation	+	+	−
modifizierbare Schrittweite	−	+	−

Tabelle 2.4: Funktionsumfang während der Simulation

2.7 Zusammenfassung und Diskussion

Ein Datenflussmodell besteht aus Boxen mit Ports als Schnittstellen und Linien als Konnektoren zwischen Ports unterschiedlicher Boxen. Zusätzlich ermöglicht das hierarchische Konzept die Kapselung von Teilsystemen in Boxen. Die hier betrachteten Werkzeuge Simulink, SCADE und RoseRT unterstützen diese Syntax und unterscheiden sich nur in einigen Details der Visualisierung. Wichtig für das Verständnis eines grafischen Modells ist die Fähigkeit des CASE-Werkzeugs den Betrachter dabei zu unterstützen. Dazu gehört die Informationsdarstellung und Simulation, sowie die Navigation als Interaktionsschnittstelle für den Benutzer. Bei der Navigation innerhalb eines Modells halten sich alle Werkzeuge in der Regel an konservative Techniken wie Fenster- und *Explosive-Zoom*-Navigation, Rollbalken und Tree-Browser. Diese Techniken zusammen mit dem Fehlen eines Übersichtsfensters verursachen allerdings bei größeren Modellen einen Kontextverlust und hohen kognitiven Aufwand bei der Navigation: Erfahrungen zeigen, dass ein Benutzer häufig aus tieferen Hierarchien zurücknavigiert, sich einen Überblick verschafft, um dann wieder die Sicht auf den detaillierten Bereich zu fokussieren. Eine gleichzeitige Betrachtung von Detail und Gesamtsicht ist nur mit Mühe (z.B. durch Verschieben und Skalieren von Fenstern bei jedem Fokuswechsel) möglich.

Bei der Untersuchung der Simulation fällt vor allem in Simulink das Fehlen vieler benötigter Features auf. Es besitzt keine eigene Simulationsumgebung, daher ist es nicht möglich eine Einzelschrittsimulation durchzuführen. Bei der Visualisierung der berechneten Daten bilden Scopes in Simulink und SCADE die zentralen Komponenten. Das Verstehen eines Scope-Graphen und die Assoziation zu einem Datum erfordert allerdings häufig höheren kognitiven und mechanischen Aufwand als nötig: es besteht keine direkte visuelle Verbindung zwischen der Graph-Anzeige und dem Modell. Ein zusätzlicher Nachteil der Simulink-Scopes ist, dass sie als eigenes Element dem Modell hinzugefügt werden müssen. Dadurch kann das Modell unnötig groß und unübersichtlich werden. Im Allgemeinen kann man sagen, dass Scopes gut dazu geeignet sind, den Datenfluss von geringen Datenmengen zu visualisieren, vorausgesetzt es handelt sich um einfache Datentypen.

Eine andere Technik zur Datenvisualisierung in Simulink (in SCADE ist es ähnlich) – die *Show-When-Hovering*-Funktion – kann aber auch keine Alternative sein, da bei größeren Datenstrukturen auch hier keine sinnvolle Darstellung mehr möglich ist. In RoseRT werden die Daten eines Datenflusses während der Simulation zunächst nicht angezeigt. In Monitor-Fenstern, in denen das Modell analog zur Editier-Sicht angezeigt werden kann, lassen sich allerdings sogenannte Probes erstellen und manuell mit den Ports verbinden. Eine grafische Visualisierung der „abgehörten“ Datenflüsse ist nicht möglich, diese werden lediglich in einer Tabelle mit ihrem aktuellen Wert angezeigt.

Diese Aspekte zeigen, dass konservative Visualisierungstechniken nicht ausreichen, um die komplexen Abläufe innerhalb eines Modells geeignet darzustellen. Die Mög-

lichkeiten eines CASE-Werkzeugs zur Analyse und Fehlerbehebung, sowie für die Visualisierung und Navigation sollten vielfältiger und leichter zugänglich sein. Neben einem Effizienzgewinn bei der Entwicklung und Einarbeitung erhält man dadurch auch bessere Ergebnisse bei der Dokumentation und Präsentation von Systemen. Das nachfolgende Kapitel zeigt, welche Konzepte zu einer Verbesserung der Datenflussvisualisierung in den Werkzeugen führen können.

3 Visualisierungskonzepte

Das vorherige Kapitel hat einige Schwächen der Werkzeuge bei Visualisierung und Navigation von komplexen Modellen, sowie bei den Analysemöglichkeiten von Daten während der Simulation diskutiert. Heutige Standardkonzepte für ergonomisches Arbeiten reichen häufig nicht mehr aus, um alle relevanten Informationen geeignet auf der Arbeitsfläche darzustellen. Durch neue Formen der Visualisierung von Datenfluss könnte das Arbeiten leichter, ein Modell robuster und nicht zuletzt die Produktivität der Entwicklung gesteigert werden. Dieses Kapitel soll eine Sammlung von Konzepten vorstellen, die zur besseren Darstellung von Datenflussmodellen geeignet sind. Dabei wird auf die Schwierigkeit bei der Realisierung der Techniken nur in geringem Maße Bezug genommen, um den Rahmen dieser Arbeit nicht zu sprengen. Zunächst werden einige Kriterien definiert, die eine gute Visualisierung charakterisieren.

3.1 Basiskriterien der Visualisierung

Nach Stasko u. a. [39] beinhaltet die allgemeine Software-Visualisierung den Einsatz von Techniken aus den Bereichen der Typographie, des Grafikdesigns, der Animation und der Kinematographie zusammen mit moderner Mensch-Maschine-Interaktionstechnologie, so dass der Mensch die Software sowohl verstehen als auch effektiv nutzen kann. Es werden u. a. Hauptkriterien genannt, die die Form, Interaktion und Effektivität von Software-Visualisierung beinhalten. Die wichtigsten Kriterien für eine gute Visualisierung werden im Folgenden aufgelistet. Diese werden bei den darauffolgenden Konzeptvorschlägen wieder aufgegriffen.

Ergonomie: Eine Visualisierung sollte sich in einem sinnvollen Rahmen an der natürlichen Wahrnehmung des Menschen orientieren. Unnatürliche und somit in der Regel unintuitive Darstellungen erhöhen den kognitiven Aufwand für den Betrachter, um relevante Informationen erfassen zu können. Die Ergonomie muss sich dem Menschen anpassen, nicht umgekehrt.

Multimediale Darstellung: Durch die Verwendung unterschiedlicher Farben, akustischer Signale und Animationen kann das menschliche Auge bei der Aufnahme der visuellen Informationen geführt und unterstützt werden. Die Nutzung aktueller Techniken wie die von Efroni u. a. [9] vorgeschlagene *state-of-the-art*

animation und *reactive animation*, die die dynamische, interaktive und grafische Visualisierung komplexer reaktive Systeme befürwortet, sollte berücksichtigt werden.

Zusätzliche Dimensionen: Die Hinzunahme von Tiefe in der räumlichen Darstellung vergrößert den zur Verfügung stehenden Platz für Informationen. Darüber hinaus orientiert sich das menschliche Auge im dreidimensionalen Raum intuitiv. In der Regel kann es außerdem sinnvoll sein, die zu visualisierenden Informationen in einem zeitlichen Kontext zu betrachten.

Zeitliche Kontrolle: Zeitliche Einbettung erfordert die Möglichkeit der Kontrolle durch den Betrachter. Das natürliche, stetige Fortschreiten der Zeit ist eine Beschränkung, die in einer virtuellen Umgebung aufgehoben werden sollte.

Kontinuität: Die menschliche Wahrnehmung ist an natürliche und stetige Bewegungen gewöhnt. Daher ist es wichtig auch in der Software-Visualisierung diesem Grundsatz zu folgen. Jede Änderung der Darstellung (Perspektivwechsel, Objektanpassung, Animationen, ...) wird durch eine kontinuierliche Transformation (*Morphing*) leichter nachvollziehbar.

Abstraktionskontrolle: Der Betrachter muss die Möglichkeit haben, den Detailgrad einer Darstellung anpassen zu können. Informationen sollten dabei möglichst intuitiv aus- und einblendbar sein.

Automatisierung: Um die kognitive und mechanische Belastung für den Betrachter gering zu halten, sollte eine Visualisierung Aktionen zur Navigation, Anpassung und Abstraktion wenn möglich automatisieren.

Aufzeichnung: Gerade bei hohen Freiheitsgraden bei Perspektivwechseln (z. B. 3D-Raum) hilft die Möglichkeit zur Aufnahme/Wiedergabe von Teilen der Visualisierung.

3.2 Fenster-Konzepte

Fenstermanagement-Systeme gehören seit Jahren zum Standard in der ergonomischen Visualisierung von Software. Auch bei der Entwicklung von Datenflussmodellen wird diese Technik in den Werkzeugen eingesetzt. Dabei werden einzelne Sichten des Modells in separate Fenster ausgelagert, wobei diese auch in dem Hauptfenster der Anwendung integriert sein können und so die gesamte Arbeitsfläche in Bereiche unterteilen. Die Hauptansicht auf das Modell nimmt hier in der Regel den größten Platz ein. Bei der Vorstellung einiger Fensterkonzepte wird im Folgenden nur die Darstellung unterschiedlicher Sichten des Modells in separaten Fenstern betrachtet.

3.2.1 Ein-Fenster-Konzept

Bei einem Ein-Fenster-Konzept wird die gesamte Arbeitsfläche des Werkzeugs in einem Fenster dargestellt. Die Visualisierung und Navigation findet zum größten Teil in diesem Bereich statt (von der Navigation in den Menüs abgesehen). Die Darstellung des Modells in nur einem Fenster hat den Vorteil, dass der mechanische Aufwand zur Neuorientierung des Betrachters zu jedem Zeitpunkt und bei jedem Sichtenwechsel, z. B. bei Navigation durch die Hierarchien, relativ gering bleibt. Allerdings beschränken sich bei nur einem Fenster die Möglichkeiten zur Tiefen-Navigation auf *Explosive Zoom* oder *Full Zoom*. Während im ersten Fall der globale Kontext schnell verloren geht, da die umgebenden Objekte ausgeblendet werden, verändert der zweite Ansatz nicht die Perspektive des Betrachters, sondern erhöht die Darstellungstiefe des Modells so weit, dass sämtliche Hierarchien eingeblendet werden. Dabei bleibt zwar der Kontext erhalten, aber Detailinformationen gehen schon bei relativ kleinen hierarchischen Modellen verloren. In der Regel nutzen heutige Werkzeuge einen *Explosive Zoom*, da das Vermitteln von Detailwissen als relevanter bewertet wird, als die Visualisierung des globalen Kontexts. Einen Mittelweg dieser beiden extremen Ansätze gehen die Fokus-und-Kontext-Konzepte in Abschnitt 3.3.

3.2.2 Multi-Fenster-Konzept

Unter einem Multi-Fenster-Konzept versteht man die Betrachtung eines Modells aus mehreren fixen Blickwinkeln. Ein Spezialfall ist ein Zwei-Fenster-Konzept, welches in der Regel aus einer Arbeitssicht, die Detailinformationen des Modells vermittelt, und einer Übersicht (auch: Totalsicht) besteht. Abbildung 3.1 zeigt ein Beispiel einer Anwendung mit Arbeits- und Übersichtsfenster.Beide Fenster beanspruchen Bereiche der Arbeitsfläche, wobei die Arbeitssicht den größeren Platz einnimmt. Die Übersicht verhilft zur besseren Orientierung in der Darstellungsebene und verringert so den kognitiven Ballast bei der Navigation. Ein Nachteil dieser Technik ist der Platzbedarf des Übersichtfensters, wodurch in der Arbeitssicht Darstellungsfläche verloren geht. Eine Alternative sind die von Cox u. a. [7] vorgestellten *overview layer* (siehe Abbildung 3.2),bei denen Sichten mit unterschiedlichem Detailgrad halb-transparent überlagert werden. Es entsteht eine verzerrungsfreie Darstellung von Detail und Kontext. Ein Problem dabei bilden allerdings Unstetigkeiten der Detailsicht bezüglich der (überlagerten) Totalsicht. Es ist aber wichtig, dass der Betrachter ein Objekt in der einen und in der anderen Sicht als identisch identifizieren kann. Dieser Zusammenhang zwischen den Sichten könnte z. B. durch Einblendung des Mauszeigers im Arbeitsfenster und in der Totalsicht geschehen (siehe auch Berner [3]).

Die in Kapitel 2 vorgestellten Werkzeuge nutzen Varianten der Multifenster-Technik als Option zusammen mit einer Full-Zoom-Variante, wobei in keiner der Anwendungen eine Übersicht integriert ist. Stattdessen wird bei der Tiefennavigation jeder

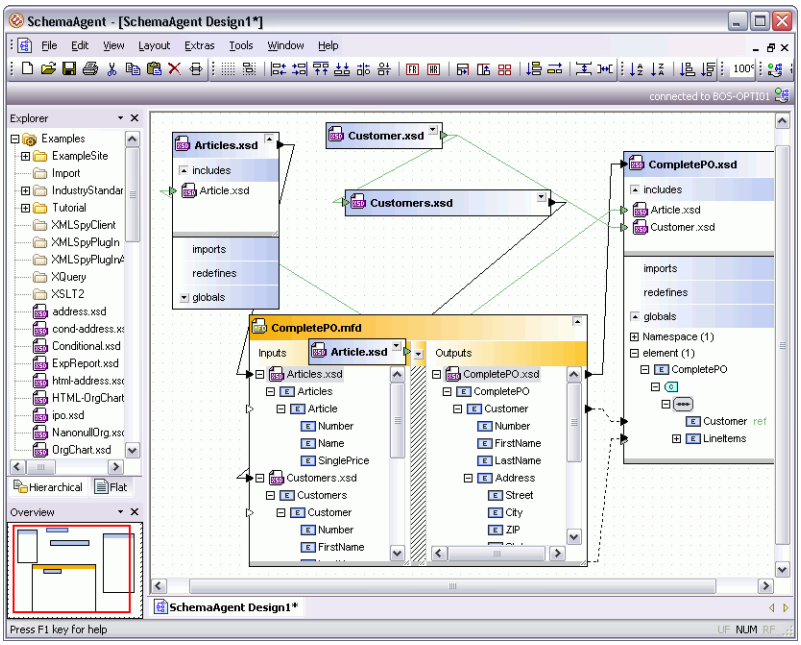


Abbildung 3.1: Design-Werkzeug mit Arbeits- und Übersichtsfenster (kleiner Bereich links unten). Der Rahmen in der Übersicht markiert den im Arbeitsfenster vergrößerten Bereich [1]

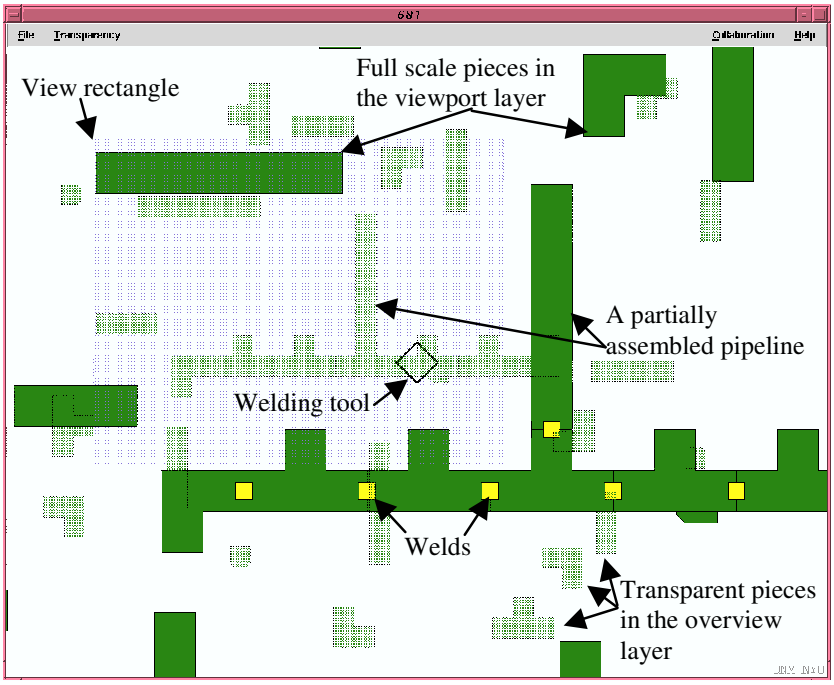


Abbildung 3.2: Übersicht und Fokus in einer gemeinsamen Sicht durch *overview layer* [7]

Perspektivwechsel in einem separaten Fenster dargestellt, so dass mehrere unterschiedliche Detailsichten parallel betrachtet werden können. Was einerseits den Informationsgehalt erhöhen und zur Kontexterhaltung beitragen soll, kann schnell zu Problemen bei der Neuorientierung und somit zu hohem Navigationsaufwand (kognitiv und mechanisch) führen. Zusätzlich überdeckt jedes neue Fenster in der Regel den größten Teil der Arbeitsfläche, so dass nur ein geringer Vorteil gegenüber einer Full-Zoom-Variante mit Ein-Fenster-Navigation besteht.

3.3 Fokus-und-Kontext-Konzepte

Konventionelle Zooming-Techniken, die die alte Sicht durch eine neue ersetzen, haben den Verlust des Kontexts zur Folge. Full-Zoom-Ansätze, bei denen die Darstellungstiefe der Hierarchien maximal ist, bewahren diesen, bieten aber keine Möglichkeit der Abstraktion und führen zu großen und komplexen Darstellungen. Jede Visualisierung hat zum Ziel, viele Informationen auf einer begrenzten Fläche geeignet und übersichtlich darzustellen. Dabei ist es wichtig, den Zugang zu Detailwissen und Übersichtswissen möglichst einfach zu gestalten. Generell gibt es zwei Grundprobleme bei der Präsentation von Daten in einem beschränkten Raum:

- das Platzproblem
- die Informationsüberladung

Das zweite Problem entsteht dadurch, dass mehr Informationen visualisiert werden als die Aufnahmefähigkeit des Betrachters zulässt. Die Beschränktheit der Darstellungsfläche ist der Grund dafür, dass die Fokussierung auf Details und der globale Kontext konkurrierende Ziele sind. Es muss daher ein Weg gefunden werden, Information im Kontext kompakter darzustellen, als in der Fokus-Sicht. Card u. a. [5] definieren Herangehensweisen zur Informationsreduktion:

- Herausfiltern der am wenigsten relevanten Informationen (*Filtering*)
- Abstraktion auf wenige Komponenten (*Selective Aggregation*)
- Hervorheben der wichtigsten Informationen durch *Highlighting*
- Verzerren/Verformen der grafischen Repräsentation (*Distortion*)

Nachfolgend werden einige bekannte Fokus-und-Kontext-Konzepte vorgestellt, die diese Punkte bereits teilweise umsetzen. Der letzte Unterabschnitt beschreibt ein semantisches Fokus-und-Kontext-Konzept, bei dem versucht wird, die Informationsselektion auf oben genannte Weisen zu realisieren, ohne die Nachteile der konventionellen Techniken zu übernehmen.

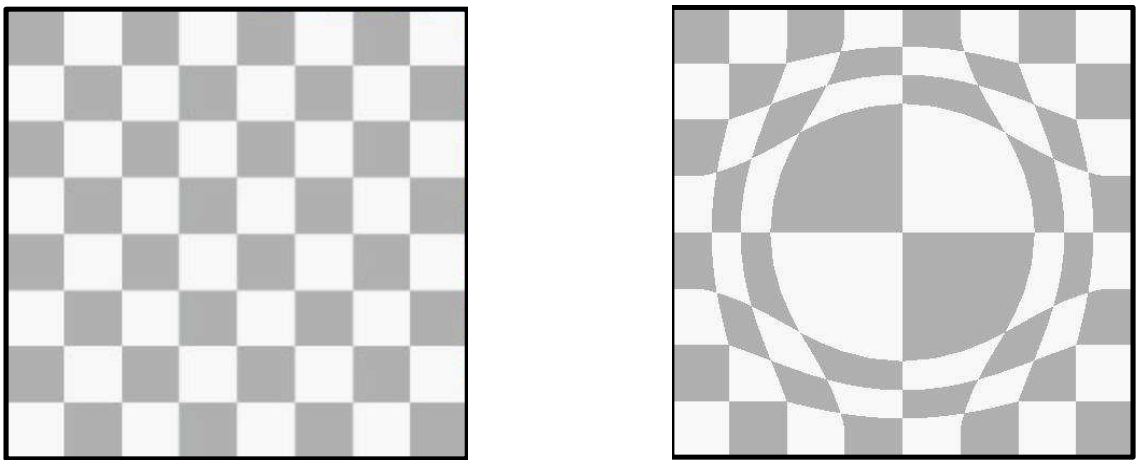


Abbildung 3.3: Verzerrung durch Fisheye-Transformation (Keahey [20])

3.3.1 Optisches Fisheye-View

Optische *Fisheye-View*-Techniken suchen in erster Linie eine Lösung für das Platzproblem, indem der fokussierte Bereich vergrößert und der umgebende Kontext verkleinert dargestellt wird. Die ursprüngliche Definition des Fisheye-Views von Furnas [12] ist eher eine Technik zur Lösung des zweiten Problems der Informationsüberladung und zählt nicht zu den Fisheye-Techniken im Sinne einer geometrischen Verzerrung. Ein Beispiel eines optischen Fisheye-Views ist in Abbildung 3.3 zu sehen. Die dabei entstehende Verzerrung erinnert an den Blick durch eine Fischaugenlinse. Gerade bei der Visualisierung von Kartenmaterial eignen sich optische Fisheye-Techniken sehr gut. Allerdings kann das menschliche Auge bei einer unbekannten Struktur nicht ohne weiteres erkennnen, ob es sich um die Fisheye- oder eine normale Sicht handelt. Hier ist es notwendig dem Auge einen Hinweis zu geben, wie es die aktuelle Sicht interpretieren muss. Sinnvoll ist hier der Einsatz von Schattierungen (*Shadings*) oder Gitterlinien (*Grids*), die den verzerrten Bereich hervorheben. Da ein realistischer Schattenwurf in der Regel sehr rechenintensive Transformationen und Lichtberechnungen erfordern, ist eine Alternative das Hervorheben des Fokus durch sogenannte *Fogs*, wie in Abbildung 3.4 zu sehen. In den letzten Jahren haben sich unterschiedliche ein- und zweidimensionale Fokus-und-Kontext-Techniken entwickelt, die auf ähnliche Weise das Fisheye-Konzept umsetzen. Häufig unterscheiden sie sich nur in ihrer Vergrößerungs- oder Transformationsfunktion. Die erste kontrolliert dabei den Grad der Vergrößerung im Fokus, und letztere den Grad der Änderung der Vergrößerung bezüglich des Abstandes zum Fokus (siehe Abbildung 3.5). Um einen Einblick zu bekommen, werden die wichtigsten Fisheye-Konzepte hier kurz vorgestellt.

Polyfocal Display: Eine unstetige Transformationsfunktion teilt das Display in mehrere Detailstufen auf. Der Fokus befindet sich im Zentrum der Ansicht, der von

Abbildung 3.4: Hervorhebung des verzerrten Bereichs durch *Fogs* (Keahey [20])

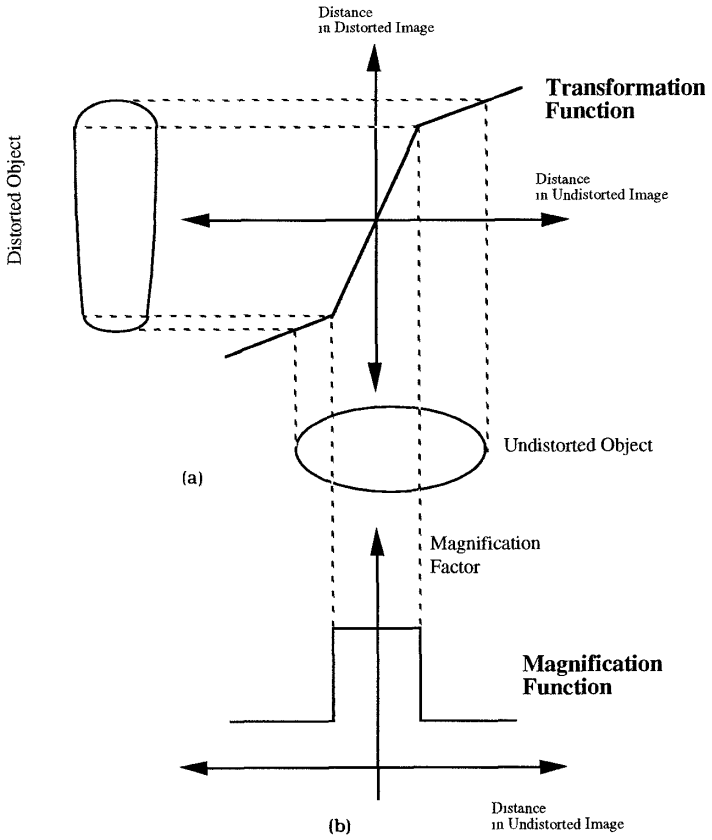


Abbildung 3.5: (a) Die eindimensionale Transformation eines Objekts durch eine Transformationsfunktion; (b) die zugehörige Vergrößerungsfunktion (Leung und Apperley [26])

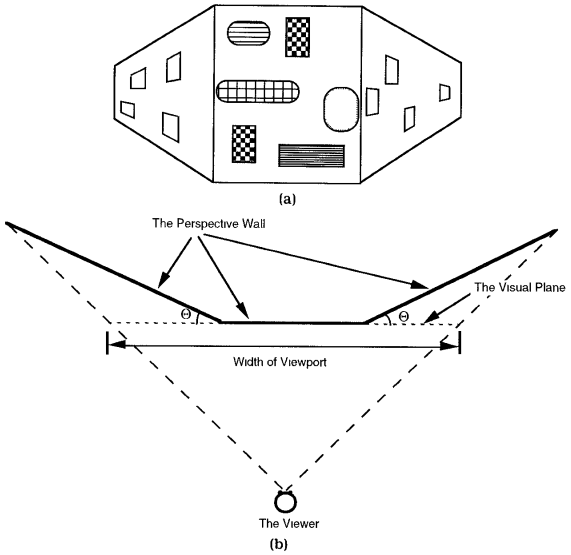


Abbildung 3.6: Funktionsweise einer Perspective wall [28]

Kontextfenstern eingerahmt wird. Die zugehörige Transformationsfunktion hat eine Treppenstufenform, so dass eine Erhöhung der Sprungstellenanzahl zu einer annähernd stetigen Sicht führt.

Bifocal Display: Ähnlich wie bei dem *Polyfocal Display* wird der Bildschirm diesmal in drei feste Bereiche eingeteilt, wobei im mittleren Bereich der Fokus liegt, der links und rechts vom Kontextinhalt umgeben wird. Diese eindimensionale Variante wurde von Spence und Apperley [38] entwickelt.

Perspective Wall: Von Mackinlay u. a. [28] kommt eine Variante des eindimensionalen *Bifocal Display* im dreidimensionalen Raum. Dabei wird, wie in Abbildung 3.6 zu sehen, die darzustellende Struktur auf eine Wand projiziert. Im Zentrum befindet sich der Fokus in einer normalen Ansicht, während die Bereiche links und rechts davon perspektivisch verzerrt dargestellt werden. Nachteil dieser Technik ist allerdings der ungenutzte Raum in den Ecken der Darstellungsfläche.

Document Lens: Hierbei handelt es sich um eine Variante der *Perspective Wall* bzw. des zweidimensionalen *Bifocal Display*. Diese Technik erweitert die Perspective Wall um Wände oberhalb und unterhalb des Fokus. Dadurch wird die ungenutzte Fläche reduziert.

Fisheye-Visualisierungen realisieren also Detail- und Überblickwissen in einer gemeinsamen Sicht und sind damit kontexterhaltend. Allerdings bestehen Datenflussmodelle typischerweise aus einfachen geometrischen Elementen, geraden Linien und rechten Winkeln. Diese Eigenschaften gehen bei einer optischen Fisheye-Transformation in der Regel verloren. Zusätzlich enthalten die Modelle viel Text, der in den Grenzbereichen des Fokusses oder bei starker Vergrößerung verzerrt wird und

dadurch nur schlecht oder gar nicht lesbar ist. Ein weiterer Nachteil von optischen Fisheye-Views ist die mangelhafte Einsatzmöglichkeit bei Fokussierung eines bereits im Fokus befindlichen Bereichs (*Fokus-auf-Fokus*), wie es in Datenflussmodellen und anderen hierarchischen Strukturen häufig notwendig ist. Hierbei treten starke Verzerrungen und Anomalien auf, die sogar zu einer Verkleinerung eines eigentlich im Fokus befindlichen Objekts führen können.

3.3.2 Grafisches Fisheye-View

Eine Lösung zur Vermeidung der Verzerrungen im optischen Fisheye-View kommt von Sarkar und Brown [36]. Die Technik wurde bei der Visualisierung topologischer Netzwerkze eingesetzt. Abbildung 3.7 zeigt ein Beispiel eines grafischen Fisheye-Views. Dabei werden nur die Knotenkoordinaten durch die Transformationsfunktion verändert und somit die typischen Verzerrungen an den Knoten und Kanten vermieden. Die Berechnung der Knotengröße richtet sich nach dem Abstand zum Fokus und einem vorher für jeden Knoten festgelegten Gewicht (*a priori importance (API)*).

3.3.3 Andere Projektionstechniken

Neben der optischen und grafischen Fisheye-Transformation gibt es noch weitere Techniken, die eine Struktur in der Regel auf eine höherdimensionale Fläche oder einen Körper projiziert, um so den zur Verfügung stehenden Raum zu vergrößern. Als Beispiele seien hier der Hyperbolische Browser (oder *hyperbolic viewer*) von Lamping und Rao [23] und die *Magic-Eye-View* von Kreuseler u. a. [22] als hyperbolische und dreidimensionale Projektionstechniken genannt. Der Hyperbolische Browser ist ein Navigations- und Visualisierungsinstrument, welches gut dazu geeignet ist große hierarchische Strukturen darzustellen. Dabei wird die zu visualisierende Struktur auf eine hyperbolische Fläche projiziert und in den euklidischen Raum abgebildet. Da in der hyperbolischen Ebene der Umfang eines Kreises exponentiell mit seinem Radius wächst, steht mit zunehmendem Abstand zum Kreismittelpunkt exponential mehr Platz zur Verfügung. Dadurch ist eine günstige Anordnung der Hierarchie möglich. Zur Änderung des Fokusses wird der zu fokussierende Bereich mit der Maus angeklickt. Abbildung 3.8 zeigt ein Implementierungsbeispiel eines Hyperbolischen Browsers. Diese Technik hat den Vorteil, dass die Darstellung vollständig konstant ist und somit eine Sichtänderung keine Unstetigkeiten verursacht.

Ähnlich funktioniert auch die Magic-Eye-View, bei der die Informationsstruktur auf einer Halbkugel abgebildet wird. Allerdings dienen diese Techniken allein der Visualisierung von Hierarchien. Sie sind nicht geeignet, Datenflussinformationen zu vermitteln. Darüber hinaus gibt es für flache Hierarchien, denen man in Datenflussmodellen häufiger begegnet, effizientere Darstellungen.

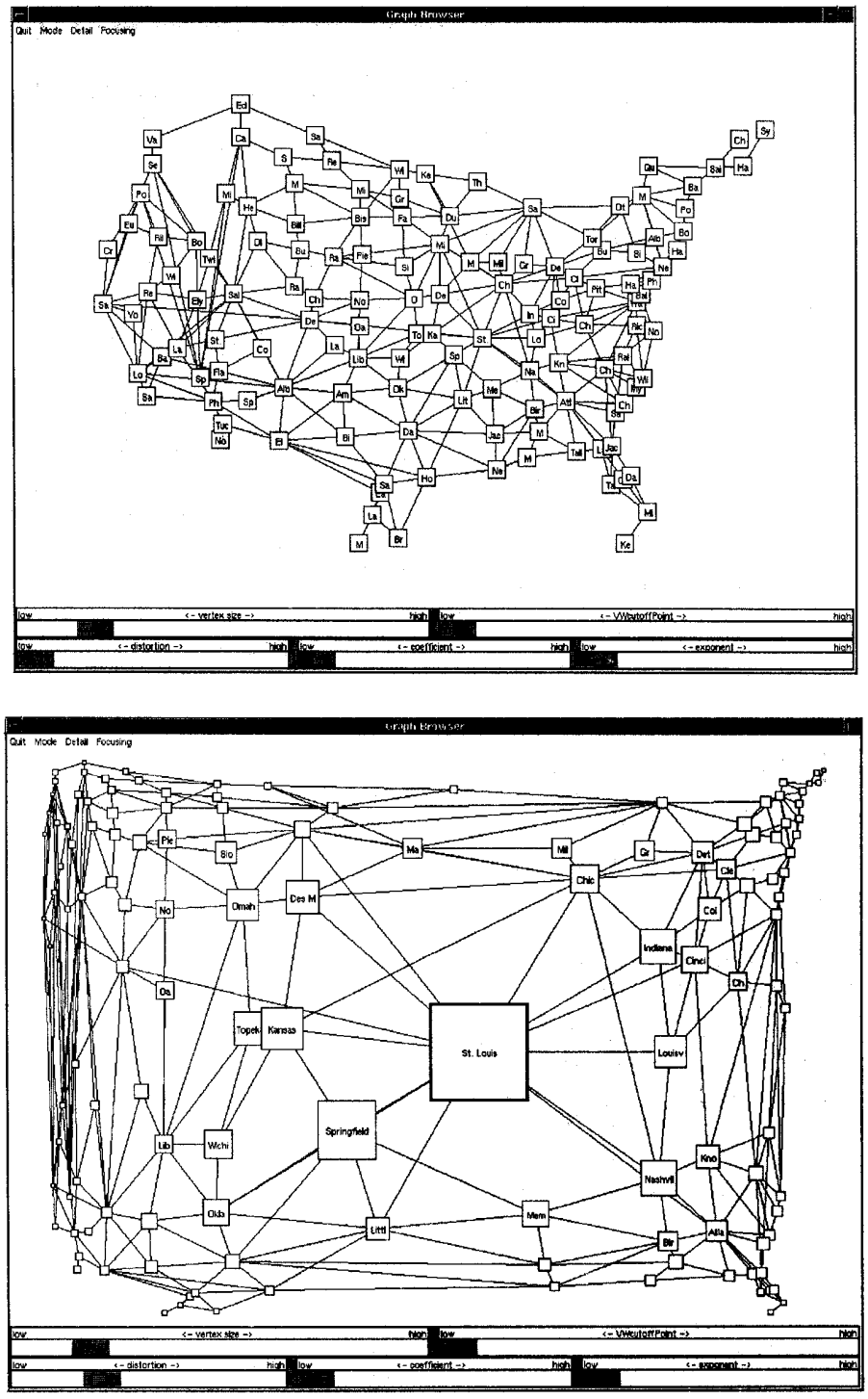
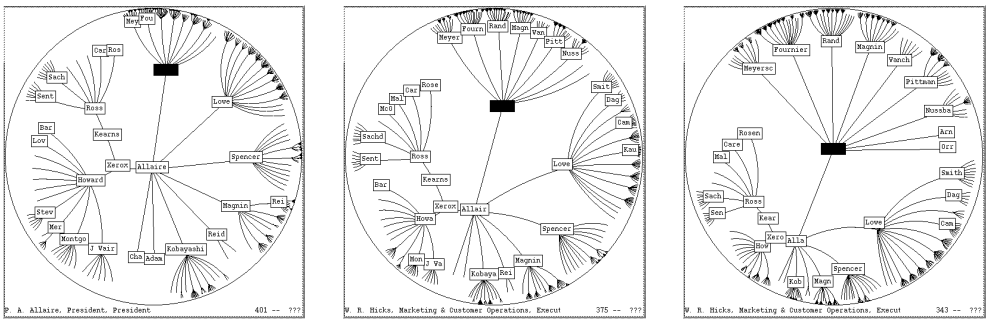


Abbildung 3.7: Grafischer Fisheye-View von Sarkar und Brown [36]



(a) (b) (c)

Abbildung 3.8: Beim hyperbolischen Browser löst ein Klick auf den hier geschwärzten Knoten eine Animation (von (a) nach (c)) aus, die diesen Knoten in das Zentrum der Darstellung platziert [23]

3.3.4 Semantisches Fokus-und-Kontext

Wie in der Einführung zu diesem Kapitel bereits gesagt, besteht das Prinzip der Informationsreduktion bei Fokus-und-Kontext nicht nur aus grafischer Verzerrung, wie es die bereits vorgestellten Techniken einsetzen, sondern auch aus Informationsselektion und Abstraktion. Ein intuitiver Ansatz zum Anpassen des Abstraktionsgrades in Datenflussmodellen wäre, den Unterbaum einer hierarchischen Komponente nicht nur abstrahiert als Box darzustellen, sondern bei Fokussierung die nicht sichtbaren Informationen einzublenden. Unter Verwendung eines grafischen Fisheye-Views nach Sarkar und Brown [36] steht der fokussierten Box mehr Platz zur Verfügung, so dass der Inhalt detaillierter angezeigt werden kann. Somit wird – verzerrungsfrei – das Detailwissen in den globalen Kontext eingebettet (oft auch *in-situ*-Vergrößerung genannt). Durch eine geeignete animierte und kontinuierliche Übergangstranformation von der Abstraktionssicht in die Fokussicht entsteht so der Eindruck einer „aufklappenden“ Box.

Zusätzlich kann – unter Beibehaltung der topologischen Eigenschaften – die Visualisierung der Umgebung durch Skalierung und Informationsselektion detailärmer gestaltet werden, um die verfügbare Fläche für die Fokussicht noch weiter zu vergrößern (Verkleinerung der nicht fokussierten Boxen, Ausblenden von Informationen). Dieses Prinzip der auf- und zuklappenden Boxen erlaubt auch – ohne auf die technischen Details einzugehen – polyfokale Sichten. So ist es möglich, ohne weiteren navigatorischen Aufwand, Detailinformationen mehrerer Teilsysteme parallel zu vergleichen, bzw. den Datenaustausch zwischen diesen genauer zu analysieren. Abbildung 3.9 zeigt in einer ersten Konzeptzeichnung, wie semantisches Fokus-und-Kontext realisiert werden könnte. Bei der Selektion von Informationen stellt sich die Frage, welche Daten (Box, Boxlabel, Ports, Portlabel, Datentypen, Werte, Kommentare) in einer Fokussicht angezeigt werden sollen bzw. welche Informationen zur Wahrung des Kontexts relevant sind. Notwendig ist hierfür eine genaue Ab-

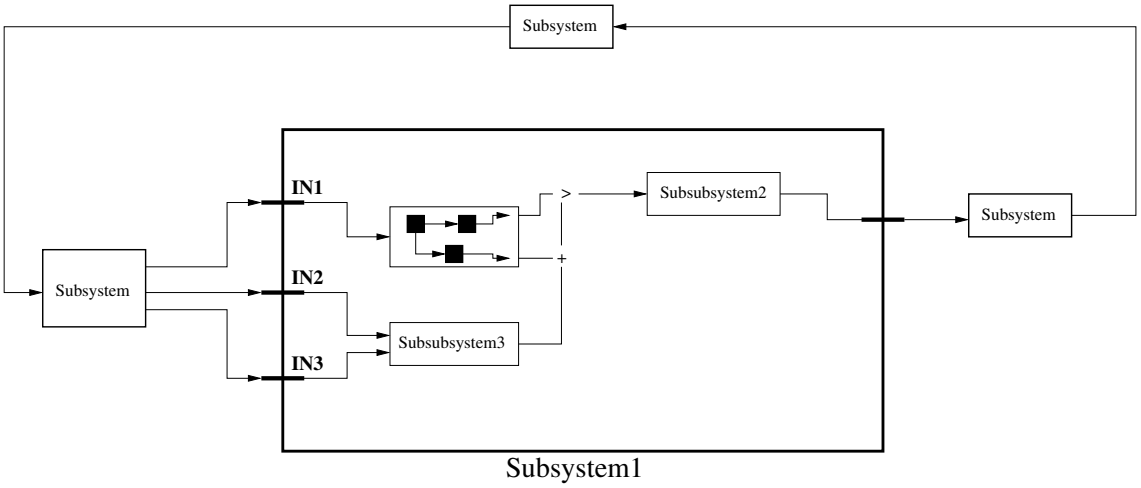


Abbildung 3.9: Realisierungsbeispiel von semantischem Fokus-und-Kontext in grafischen Datenflusssprachen

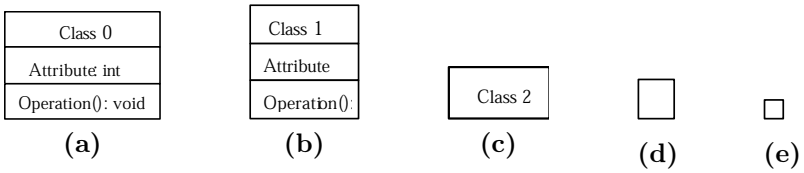


Abbildung 3.10: Informationsselektion in UML-Diagrammen durch unterschiedliche *level of detail*, von LOD 0 bis LOD 4 (a bis e). [30]

grenzung von Detail- und Kontextinformationen eines Datenflussmodells. Darüber hinaus ist eine Spezifikation unterschiedlicher Detailgrade sinnvoll, wobei in der höchsten Detailstufe möglichst alle sinnvollen Informationen eines Modells enthalten sein sollten und in der niedrigsten gerade noch diejenigen Informationen, die zur Wahrung des Kontexts einen Beitrag leisten. Musial und Jacobs [30] haben bei der Entwicklung einer Fokus-und-Kontext-Technik für UML-Diagramme bereits sechs sogenannte *Level-Of-Detail (LOD)* definiert. Abbildung 3.10 zeigt die Darstellung der LODs einer UML-Klasse von Abb. 3.10a bis 3.10e. Besitzt eine Klasse einen LOD von 5, wird sie nicht mehr angezeigt. Dies setzt voraus, dass das Interesse für Detailinformationen stärker wird, je kleiner der Abstand zum Fokus ist. Diese Technik schafft Platz für das fokussierte Objekt, so dass für dieses mehr Detailinformationen dargestellt werden können. Eine ähnliche Anwendung wäre auch für die Darstellung von Informationen in Datenflussmodellen denkbar. Zusätzlich zu den direkten Eigenschaften einer Box, wie Box- und Portlabel, Ports, Datentypen, etc., die beispielsweise je nach LOD und Abstand zum Fokus ein- oder ausgeblendet werden, könnte auch die hierarchische Struktur in die Definition eines LODs für Datenflussmodelle einfließen, so dass der Abstand eines Objektes zum Fokus nicht nur in der Darstellungsebene, sondern auch in der Darstellungstiefe berücksichtigt wird. Darüber hinaus kann diese Art der Informationsselektion auf Nicht-Block-Elemente

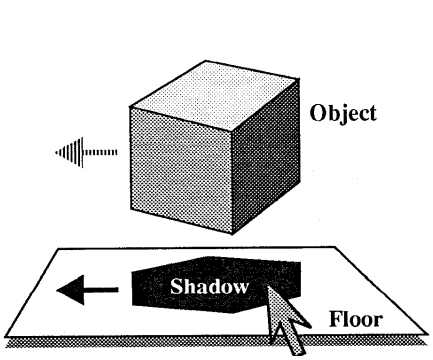
erweitert werden. So könnten mehrere, z. B. parallel laufende Datenflussleitungen ab einer bestimmten Detailstufe zu einer symbolischen Leitung zusammengefasst werden. Wichtig ist bei all diesen Abstraktionsformen, dass der Betrachter jederzeit in der Lage ist, das Modell eindeutig zu interpretieren. D.h. eine zusammengefasste, abstrahierte Darstellung mehrerer Verbindungslinien muss jederzeit als solche erkennbar sein und sich deutlich von einer herkömmlichen Leitung unterscheiden, da ansonsten Orientierungsprobleme drohen. Ähnlich verhält es sich bei der Anzeige von Boxen. Zwei Boxen unterschiedlicher Hierarchien müssen sich eindeutig voneinander unterscheiden.

Obwohl semantisches Fokus-und-Kontext eine sinnvolle Alternative zu herkömmlichen Visualisierungsmethoden wäre, ist dieser Ansatz nicht ohne Nachteil. Ähnlich wie bei optischen Fisheye-Views treten Probleme bei großen Hierarchien und Fokus-in-Fokus-Sichten auf. Der Grund dafür ist, je mehr Details eingeblendet werden (z. B. durch gleichzeitige Anzeige vieler Hierarchien eines Teilsystems im Datenflussmodell), desto mehr Platz beansprucht der fokussierte Bereich auf der Arbeitsfläche, so dass für den umgebenden Kontextbereich kein Platz mehr bleibt. D. h. die Fokus-Sicht degeneriert in Richtung eines *Explosive Zooms* und verliert damit ihren zusätzlichen Nutzen. Verschärft wird dieses Problem durch Einsatz polyfokaler Techniken. Das Erlauben von Überdeckungen und/oder die Platzierung von Objekten außerhalb des sichtbaren Bereichs bilden keine wirklichen Alternativen.

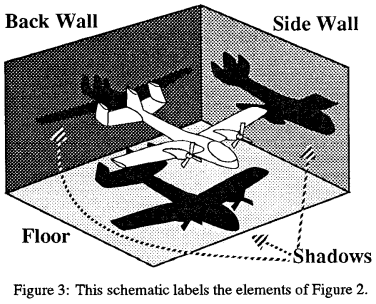
3.4 3D-Visualisierung

Die gewöhnliche zweidimensionale Arbeitsfläche ist bei der Darstellung stark im Platz beschränkt. Es scheint daher plausibel, dass das Hinzufügen von Tiefeninformationen die Arbeitsfläche effektiver ausnutzt und somit mehr Information dargestellt werden kann. Darüber hinaus fällt es dem menschlichen Betrachter leicht, sich in einer dreidimensionalen Umgebung zu orientieren. Allerdings hat die 3D-Visualisierung Nachteile: bei vollständig manueller Navigationskontrolle durch den Benutzer ergeben sich zu viele Freiheitsgrade. Generell existieren zur Benutzerinteraktion Eingabegeräte wie Maus und Tastatur, die nur die effektive Kontrolle in einer zweidimensionalen Umgebung erlauben. Hier müssen Techniken zur Reduktion der Freiheitsgrade und (halb-)automatische Navigationsmöglichkeiten entwickelt werden. Erste Ansätze hierfür bietet die *point-of-interest*-Navigation von Mackinlay u. a. [27], bei der durch Benutzereingabe ein interessantes Objekt angewählt werden kann und vom System automatisch eine geeignete Sicht zur Verfügung gestellt wird.

Ein ähnliches Problem wie bei der Navigation ergibt sich bei der Positionierung von Objekten innerhalb des dreidimensionalen Raums. Ideen zur Lösung sind selektierbare Ankerpunkte an der Bounding-Box des Objekts oder die Verwendung *interaktiver Schatten* von Herndon u. a. [16]. Bei dieser Visualisierungstechnik werden die



(a) Durch Ziehen der Schattenprojektion mit dem Mauszeiger bewegt sich auch die Box parallel zur Schattenoberfläche



(b) Die Projektion auf drei orthogonale „Wände“ erlaubt eine flexible Handhabung

Abbildung 3.11: Positionierung von Objekten im dreidimensionalen Raum durch *Interaktive Schatten* [16]

Oberflächennormalen eines Objekts auf orthogonale Wände projiziert und können interaktiv verändert werden. Abbildung 3.11 zeigt die Funktionsweise dieses Prinzips. Abgesehen von dem hohen Rechenaufwand hat eine empirische Untersuchung von Wanger [41] ergeben, dass diese Visualisierung die Orientierung im 3D-Raum erleichtert. Allerdings ist diese Art der Interaktion nicht sehr natürlich und intuitiv. Ein weiteres wichtiges Hilfsmittel bei der Visualisierung im dreidimensionalen Raum ist die Animation. Der Mensch kann sich in der natürlichen Umgebung nur deshalb so gut orientieren, weil er mehrere Informationen der Umgebung auswertet. Dabei setzt sich das Gesamtbild aus Schattenwurf, Bewegungsgeschwindigkeit (weiter entfernte Objekte bewegen sich langsamer, nah positionierte Objekte schneller) und den Informationen des stereoskopischen Sehens zusammen. Dadurch kann die Entfernung und Tiefe von Objekten eingeschätzt werden. Gerade die Information des stereoskopischen Sehens geht bei der Projektion auf eine zweidimensionale Oberfläche verloren. Die physikalisch korrekte Animation und Bewegung innerhalb des Raums substituiert das Fehlen des „zweiten Auges“ und kann die Orientierung erleichtern.

Ein Beispiel zur Visualisierung hierarchischer Informationen bilden die von Robertson u. a. [34] entwickelten *Cone-Trees*. Abbildung 3.12 zeigt die Standardansicht, bei der die komplette Hierarchiestruktur dargestellt wird. Zur Fokussierung eines bestimmten Astes klickt der Benutzer auf eine Box und das System wechselt die Perspektive automatisch in einer fließenden Bewegung, so dass das selektierte Objekt dem Benutzer am nächsten und die unteren Hierarchie-Ebenenen dahinter und versetzt platziert werden. Dadurch bleibt auch im Fokus ein gewisser Kontext erhalten. Die hierarchische Struktur wirft zusätzlich einen Schatten auf den Boden des virtuellen Raums, der die Objektdichte innerhalb von Teilhierarchien visualisiert. Ein Vorteil von Cone-Trees ist der Wegfall der sukzessiven Navigation durch die Hierarchien, da sämtliche Hierarchiestufen *a priori* dargestellt werden.

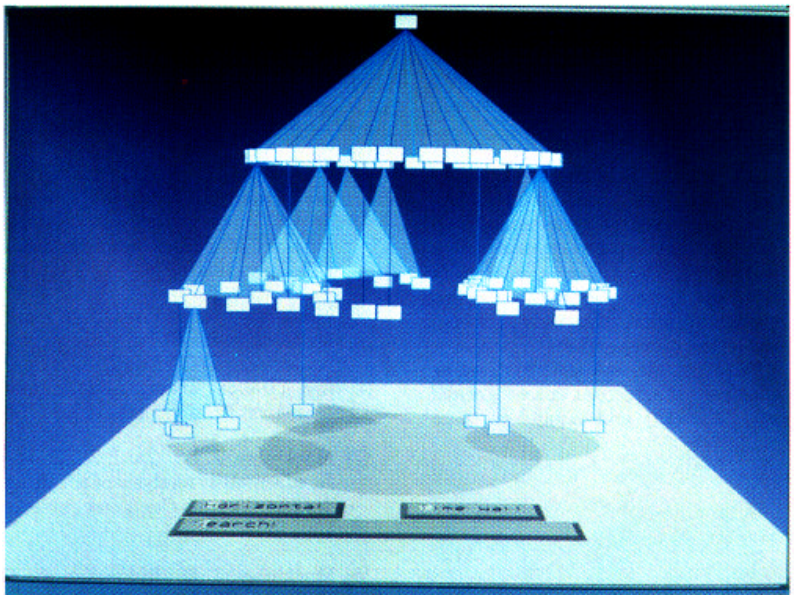


Abbildung 3.12: Visualisierung hierarchischer Informationen in einem Cone-Tree [34]

Betrachtet man die Visualisierung von Datenflusssprachen, sprechen einige Punkte gegen Cone-Trees. Die Umsetzung ist nur für sehr beschränkte Anwendungsfälle geeignet und Erweiterungsmöglichkeiten bestehen keine. Innerhalb einer Hierarchieebene lassen sich keine Verbindungen zwischen Objekten darstellen, die die eigentliche Datenfluss-Assoziation visualisieren würden. Es existieren keine Interaktionsmöglichkeiten mit Objekten, d. h. Objekte können nicht frei platziert werden. Darüber hinaus ist der Informationsgehalt der Schattenprojektion fraglich. Eine Verbesserung wäre schon durch Einführung interaktiver Schatten möglich. Basierend auf diesen Ansätzen müsste allerdings eine komplett neue Technik entwickelt werden.

Auch im allgemeinen Fall sind die Einsatzmöglichkeiten einer 3D-Visualisierung für Datenflussmodelle zumindest fraglich. Die Schwierigkeiten bei der Realisierung und bei Navigation und Layout von Objekten innerhalb einer solchen Umgebung sind meist größer als der eigentliche Nutzen. Nicht zuletzt suggeriert das Betrachten von Objekten in einem dreidimensionalen Raum das Vorhandensein realer Körper, so dass das zugrunde liegende Prinzip der Abstraktion von Prozessen – zumindest visuell – verloren gehen kann.

3.5 Textuelle Repräsentation

Bei der Entwicklung mit grafischen Modellierungswerkzeugen wird erfahrungsgemäßviel Zeit in die Entwicklung ästhetischer Modelle investiert, in der Hoffnung, die Lesbarkeit erhöhen zu können. Gerade bei großen Modellen nimmt dieser Aufwand erheblich zu, da der Benutzer unter Umständen viele Objekte verschieben muss,

um Platz für zusätzliche Elemente zu schaffen. Zusätzlich wird dieses Problem verschärft, wenn zusätzliche Verbindungslinien zwischen Objekten hinzukommen. In der Regel wird versucht, durch langwieriges Verschieben von Objekten, möglichst kreuzungsfreie Verbindungen herzustellen oder andere ästhetische Kriterien zu erfüllen. Dazu kommt, dass beispielsweise die Modellierung mathematischer Gleichungen mit grafischen Datenflusssprachen mühsam und zeitaufwändig ist, während die textuelle Beschreibung intuitiv erscheint. Auf der anderen Seite lassen sich komplexe modulare Systeme und Zusammenhänge zwischen diesen verständlicher in grafischer Form darstellen. Aus diesem Grund kann es sinnvoll sein, dem Benutzer neben der grafischen Bearbeitung die zusätzliche Möglichkeit der textuellen Erstellung eines Modells zu bieten.

Eine Möglichkeit dazu wäre die Anwendung des Prinzips der Trennung von Topologie und Layout, wie es schon in dem Textsatz-Paket LaTeX [24] zum Einsatz kommt und in dem Werkzeug KIEL [21] für die Entwicklung mit Statecharts eingesetzt wird. Die textuelle Beschreibung des Modells könnte so von unnötigem (Layout-)Ballast befreit werden. Allerdings setzt diese Technik einen vorhandenen Layout-Generator und in diesem Zusammenhang eine Definition geeigneter Layoutregeln voraus. Die daraus resultierenden Probleme werden im nachfolgenden Kapitel diskutiert.

Angenommen, ein automatisches Layout wäre vorhanden, so erhielte man auf diese Weise die Effizienz einer textuellen Sprache zusammen mit der Ausdruckskraft einer grafischen Abbildung. Die textuelle Eingabe eines Objekts könnte die parallele automatische Platzierung in der grafischen Modellsicht zur Folge haben, was den kognitiven Aufwand bei der Umsetzung von Ästhetikkriterien und Sekundärnotationen (Kreuzungsfreiheit, etc.) stark reduziert.

Zur Beantwortung der Frage, wie eine Sprache aussehen sollte, die zur Beschreibung von Datenflussmodellen geeignet ist, bietet sich zunächst eine Analyse von vorhandenen Lösungen an. Die Ausdrucksfähigkeit einer textuellen Sprache sollte mindestens die Beschreibung folgender Komponenten ermöglichen:

- Basisoperatoren (+, -, /, *, ...)
- Definition eigener Funktionen und Schnittstellen (Box und Port)
- Ist-Teil-Von-Beziehung zwischen Boxen (Hierarchie)
- Flussbeschreibungen (z. B. `box1.port1->box2.port1`)

Als Beispiel dient hier die synchrone Sprache *Lustre* [14], da sie sich gut für die Beschreibung von Datenflüssen eignet und bereits als Basis für die grafische SCADE-Syntax dient. Die Abbildung 3.13 zeigt die textuelle Beschreibung eines Modells in Lustre und die äquivalente grafische Repräsentation in SCADE.

```
node A(i1 : int ; i2 : int)
          returns (o : int) ;
let
  o = i1 + B(i2);
tel;

node B(Input1 : int)
          returns (Output1 : int);
let
  Output1 = Input1 * 3;
tel;

node System(i1 : int ; i2 : int)
          returns (o : int);
let
  o = A(i1 , i2);
tel;
```

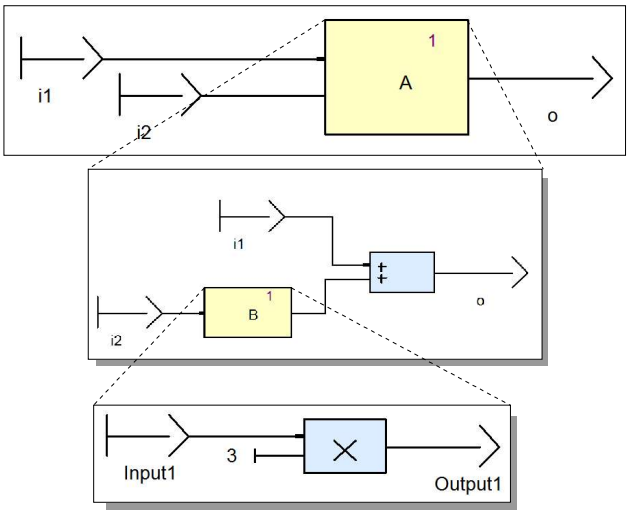


Abbildung 3.13: Äquivalente grafische und textuelle Repräsentation eines Modells

3.6 Zusammenfassung und Diskussion

Die visuelle Entwicklung von Datenflussmodellen erfordert mächtige und ergonomische Modellierungswerkzeuge. Heutige Standardkonzepte zur Visualisierung reichen oft nicht aus, um das Potential grafischer Sprachen vollständig zu nutzen. Dieses Kapitel hat gezeigt, welche Grundkriterien bei der Darstellung von Informationen zu berücksichtigen sind und welche Möglichkeiten bei der grafischen Präsentation von Modellen zur Verfügung stehen. Neben unterschiedlichen Fenster- und 3D-Techniken scheint der Einsatz von Fokus-und-Kontext-Konzepten eine sinnvolle Alternative zu herkömmlichen Visualisierungen zu sein. In diesem Zusammenhang wurde eine semantische Fokus-und-Kontext-Technik vorgestellt, die durch Informationsselektion und in-situ-Vergrößerung Detailinformationen in die globale Sicht einbettet und somit den kognitiven Aufwand bei Navigation und Orientierung verringert. Abschließend wurde der Einsatz von textuellen Sprachen als lohnendes Hilfsmittel zur Modellierung vorgeschlagen. Eine Symbiose von textueller und grafischer Entwicklung hätte hohe Effizienzgewinne und Ausdrucksfähigkeit zur Folge. Allerdings erfordert das textbasierte Modellieren grafischer Objekte Techniken für ein automatisches Layout. Die Grundlagen und resultierende Probleme dieser Thematik werden im nächsten Kapitel diskutiert.

4 Layout und dynamische Repräsentation

Das vorherige Kapitel zeigte einige Konzepte zur besseren Visualisierung von Datenflussmodellen. Dabei stand in erster Linie die Lösung des Platz- und Informationsdichteproblems und die Beantwortung der Frage, *welche* Informationen dargestellt werden, im Vordergrund. Dieses Kapitel beschäftigt sich mit der Layoutproblematik von Datenflussmodellen und gibt Antworten auf die Frage, *wie* Informationen präsentiert werden sollten. Ziel ist es vor allem die Möglichkeiten eines computergestützten, automatischen Layouts zu offenbaren.

Im Gegensatz zur textuellen Entwicklung wird in grafischen Modellierungssprachen viel Zeit für das Platzieren von Boxen und Konnektoren aufgewendet, oft ohne weitere Funktionalität hinzuzufügen, sondern um das Modellierte leicht verständlich zu machen. Gerade weil die grafische Entwicklung eines Systems selten geradlinig verläuft und Objekte in bereits bestehende Bereiche eingefügt werden müssen, führt dies häufig zu einem kompletten Re-Design des bestehenden Layouts, um möglichst Kanten- und Box-Überdeckungen zu vermeiden. Andernfalls können schnell unleserliche, schwer verständliche und fehleranfällige Modelle entstehen.

In den textuellen Sprachen existiert dieses Formatierungs-Phänomen zwar auch, allerdings – aufgrund der sequentiellen Struktur von Programmiersprachen und Textwerkzeugen – in geringerem Ausmaß. Darüber hinaus werden Entwickler bei der Einhaltung von Formatierungsregeln durch unzählige Guidelines, Style-Checker und sogenannte *Pretty-Printer*, die der automatischen Formatierung dienen, unterstützt. Bei visuellen Programmiersprachen und ihren Entwicklungsumgebungen ist diese Unterstützung oft noch nicht vorhanden.

Der nächste Abschnitt führt zunächst den von Petre [31] definierten Begriff der Sekundärnotation für visuelle Datenflusssprachen ein. Dazu werden einige Regeln vorgeschlagen, die nicht zur formalen Notation der Sprachen gehören, sondern gewisse objektive ästhetische Kriterien erfüllen. Abschließendes Ziel ist es, Grundlagen für die Realisierung eines automatischen Layouts im Sinne eines *Pretty-Printers* zu schaffen.

4.1 Sekundärnotation

Bei der grafischen Entwicklung von Systemen ergeben sich für den Anwender von Modellierungswerkzeugen viele Freiheiten. Die Möglichkeiten bei der Anpassung von Aussehen und Platzierung von Objekten wie Boxen, Labels und Konnektoren, sowie bei der Namensgebung sind vielfältig und von subjektiven Ästhetikkriterien abhängig. Dies verschlechtert die Vergleichbarkeit von Modellen untereinander und erhöht zusätzlich den Kommunikations- und Zeitaufwand bei der Einarbeitung und Weitergabe der Modelle. In textuellen Sprachen wird seit langem der Begriff der Sekundärnotation verwendet, um die subjektiven Gestaltungs- und Formatierungsmöglichkeiten zu beschränken und so einheitlichen und zugänglicheren Code zu schaffen. Diese Richtlinien existieren für fast alle herkömmlichen Programmmiersprachen. Nicht nur zur Verbesserung der oben genannten Punkte ist ein solches Regelwerk auch für grafische Datenflusssprachen sinnvoll. Will man das Layout eines Modells automatisch erstellen lassen, ist man ebenfalls auf eine Definition fester, algorithmisch umsetzbarer Kriterien angewiesen. Dabei unterscheidet man im allgemeinen (z. B. Siebenhaller [37]) zwischen *semantischen* und *syntaktischen* Ästhetikkriterien. In den folgenden Abschnitten werden beide Begriffe kurz erklärt und Kriterien für die Anwendung in Datenflussmodellen vorgeschlagen.

4.1.1 Syntaktische Ästhetikkriterien

Diese Kriterien bilden die generellen Anforderungen an die Formatierung und das Layout einer grafischen Repräsentation. Sie sind in der Regel von der Semantik des Systems unabhängig und lassen sich aus diesem Grund modellunabhängig in einem automatischen Layoutalgorithmus integrieren. Im folgenden werden einige Vorschläge für syntaktische Kriterien aufgezählt und näher erläutert.

Abstand zwischen Objekten: Analog zum Layout von Graphen ist eine zu dichte Platzierung von Objekten abträglich für die Lesbarkeit eines Modells. Dazu gehört, dass insbesondere Berührungen von Boxen oder Kanten, sowie von den zugehörigen Labels vermieden werden sollten. Bei einer engen Platzierung von Datenflusslinien besteht zusätzlich die Gefahr der Verwechslung.

Kantenlänge: Sehr lange Datenflusslinien verschlechtern gerade in komplexen Modellen die Lesbarkeit, da die Zuordnung von Start- und Endpunkt nur durch explizites Nachverfolgen der Verbindung möglich ist. Daher sollte bei einem Layout darauf geachtet werden, möglichst kurze Kantenlängen zu erreichen, wobei eine vorhandene Datenflussrichtung nicht zerstört werden darf.

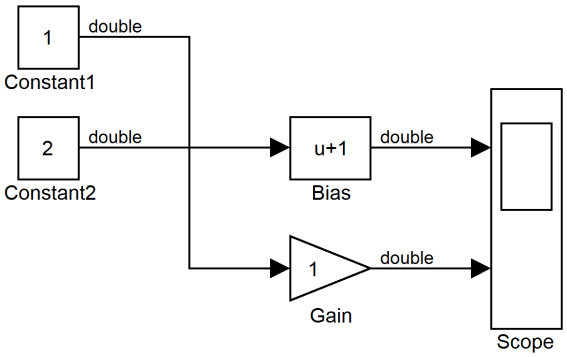


Abbildung 4.1: Problem der Kantenkreuzung: Es ist nicht ersichtlich, ob sich die Kanten berühren oder schneiden.

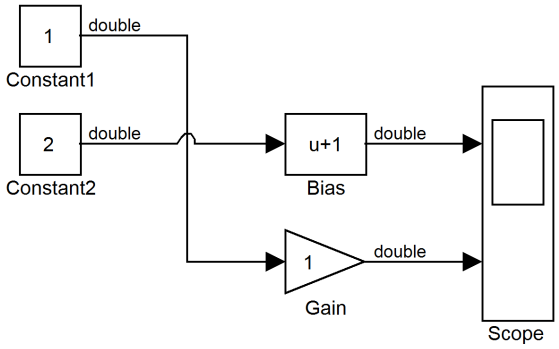


Abbildung 4.2: Ansatz zur Lösung des Kantenkreuzungs-Problems: Verwendung von *Bridges* (Bildmontage eines Simulink-Modells)

Anzahl der Kantenkreuzungen: Die Minimierung der Kantenkreuzungen ist ein aus dem Graphlayout entnommenes Kriterium. Für dieses NP-vollständige Problem existieren eine Reihe von Heuristiken, die relativ nah an die optimale Lösung konvergieren. Dabei hat die Erzeugung von Kreuzungsfreiheit häufig die Umordnung anderer Objekte zur Folge, was zu Kontextverlust und erhöhtem kognitiven Aufwand bei der Neuorientierung führt. Bei Datenflussmodellen ist daher der Erhalt der relativen Objektplatzierung der Minimierung von Kantenkreuzungen vorzuziehen. Ein Ausweg, das Kantenkreuzungsproblem (siehe Abbildung 4.1) zu lösen, ohne die Datenflussrichtung zu zerstören, bietet der Einsatz von *Bridges*, wie in den Abbildungen 4.2 und 4.3 zu sehen. Andere Möglichkeiten sind der Einsatz lokaler Variablen (SCADE) oder GOTO-/FROM-Blöcke (Simulink), um Verbindungslinien zu vermeiden. Allerdings kann der positive Effekt bei Einsatz dieser Sprachelemente durch zu häufige Benutzung innerhalb eines Modells verloren gehen. Zusätzlich ist darauf zu achten, dass mindestens eins der beiden miteinander kommunizierenden Objekte mit dem Rest des Modells verbunden ist (siehe Abbildung 4.4).

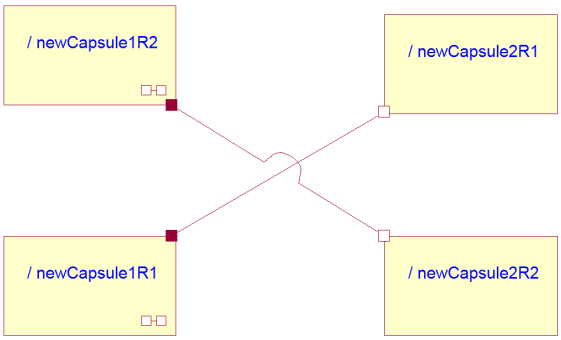


Abbildung 4.3: Automatische Behandlung von Kantenkreuzungen in RoseRT

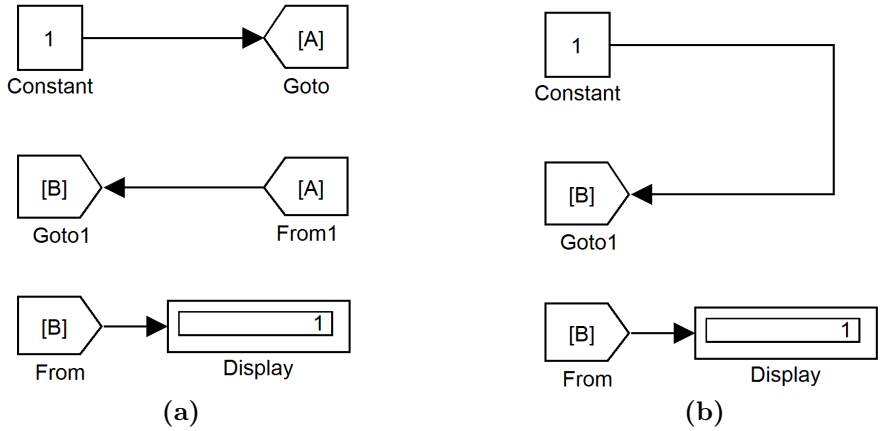


Abbildung 4.4: (a) Obwohl hier die Zuordnung noch klar ist, sollte mindestens einer der mittleren beiden Blöcke `Goto1` oder `From1` mit dem Rest des Modells verbunden sein. (b) Beispiel zur Auflösung des Problems

Anzahl der Kantenknicke: Bei Datenflussmodellen werden häufig orthogonale Verbindungen zur Darstellung von Datenfluss verwendet. Ursache dafür ist sicherlich die naheliegende Metapher der elektronischen Leiterbahnen, deren Layout ebenfalls aus orthogonalen Linien besteht. Ein gutes Layout sollte allerdings möglichst wenig Kantenknicke enthalten.

Symmetrie: Ein gutes Layout sollte vorhandene syntaktische Symmetrien berücksichtigen. Dadurch können ähnlich strukturierte Vorgänge schneller erfasst werden.

Seitenverhältnis (*aspect ratio*) Ein gutes Layout nutzt die zur Verfügung stehende Arbeitsfläche auf effiziente Weise. D. h. betrachtet man das gesamte Modell, so sollte es weder zu lang, noch zu breit sein. Gerade bei Einsatz von Fokus-und-Kontext-Techniken, die hierarchische Detailinformationen in den Gesamtkontext einbetten, können schnell Modelle mit ungünstigem Seitenverhältnis entstehen, die die Arbeitsfläche nur schlecht ausnutzen.

Art der Linienführung: Wenn das Werkzeug es zulässt, sollte abgewogen werden, ob nur rein orthogonale oder auch „schräge“ Verbindungen zulässig sind, oder ob sogar die Verwendung von Splines als Konnektoren zu einem besseren Verständnis des Modells führt.

Hierarchieübergreifende Platzierung: Schnittstellen-Elemente zwischen den Hierarchieebenen „verschlucken“ den Datenfluss um ihn zu ihrem Gegenstück in der anderen Hierarchieebene weiterzuleiten (Input/Output, GOTO/FROM). Gerade bei Betrachtung von semantischem Fokus-und-Kontext ist es sinnvoll, diese Elemente so anzuordnen, dass ihre Verbindungen leicht nachzuvollziehen sind.

4.1.2 Semantische Ästhetikkriterien:

Im Gegensatz zu den syntaktischen haben die semantischen Ästhetikkriterien einen semantischen Bezug zu dem Modell. Dabei hängt die Formatierung stark von dem zugrunde liegenden System ab, wodurch zusätzliches Wissen über die Funktionsweise des Modells nötig ist, um ein geeignetes Layout zu erstellen. In der Regel können semantische Ästhetikkriterien nur sehr aufwendig in einem automatischen Layout berücksichtigt werden.

Flussrichtung: Abhängig von den semantischen Zusammenhängen des Systems sollte ein vorhandener Datenfluss über mehrere Teilsysteme hinweg auch durch geeignete Platzierung der Ein- und Ausgänge von Boxen sowie der Kantenführung visualisiert werden.

Clustering: In erster Linie sollten zusammengehörige Elemente zu einem Teilsystem abstrahiert werden. Andererseits ist dies manchmal nicht gewollt, obwohl eine gewisser Zusammenhang zwischen Elementen besteht. In diesem Fall ist es sinnvoll, diese Objekte so zu positionieren, dass aufgrund ihrer geometrischen Lage eine Verbindung zwischen diesen und eine Abgrenzung zu anderen Objekten ersichtlich ist.

Platzierung von Beschriftungen/Labels: Die Platzierung von Labels kann sowohl syntaktisch als auch semantisch betrachtet werden. So kann z. B. die Platzierung von Kommentaren und Anmerkungen nicht auf rein syntaktischer Ebene vorgenommen werden, es sei denn, die Sprache erlaubt eine direkte Zuordnung dieser textuellen Bausteine zu einem Objekt.

Objektgrößen: Grafische Datenflusssprachen bieten durch Standardoperatoren zur Addition, Multiplikation, etc. eine grundlegende Ausdrucksfähigkeit an. Diese Operatoren werden im Allgemeinen grafisch kleiner dargestellt, als Objekte, die eine echte Abstraktion des modellierten Systems bilden. Aufgrund der unterschiedlichen Größen stellt der Betrachter oft eine semantische Verbindung zur Relevanz her. Ein großes Objekt ist wichtiger als ein kleines. Dieses Kriterium muss auch nach einem automatischen Layout erhalten bleiben.

4.2 Der MAAB-Styleguide

Da das Problem einer fehlenden Sekundärnotation für grafische Sprachen bekannt ist, existieren inzwischen einige Regelwerke, in denen diese Kriterien festgehalten sind. In erster Linie hat die Entwicklung dieser *Styleguides* das Ziel, wenig fehleranfällige, nachvollziehbare und wartbare Modelle zu entwickeln.

Ein frei verfügbarer Styleguide für Matlab, Simulink und Stateflow wurde durch eine Gruppe namhafter Automobilhersteller initiiert. Zur Gründung bestand das Konsortium mit dem Namen *MathWorks Automotive Adisory Board (MAAB)* aus den Firmen Ford, Daimler Benz (jetzt Daimler/Chrysler) und Toyota. Bei den spezifizierten Regeln hält sich der Bezug zur Automobilindustrie allerdings in Grenzen, so dass dieser Styleguide Unternehmen aus völlig unterschiedlichen Bereichen als Grundlage dient. Spezielle Erweiterungen dieses Quasi-Standards kommen beispielsweise von

MALC (Luftfahrt) und dem J-MAAB, einem Zusammenschluss japanischer Automobilkonzerne. Der MAAB-Styleguide wurde im April 2001 veröffentlicht, die Erweiterungen des J-MAAB wurden 2003 eingeführt. Zusätzlich existieren inzwischen häufig unternehmensinterne Erweiterungen für spezielle Anforderungen. Ein Nachteil der MAAB-Styleguides ist z. B. die fehlende Trennung von syntaktischen und semantischen Kriteren.

4.3 Automatisches Layout

Wie bereits erwähnt, existieren in textuellen Programmiersprachen sogenannte Pretty-Printer, die Code automatisch nach den Regeln einer bestimmten Sekundärnotation formatieren. Die Probleme bei der Umsetzung von Layoutregeln sind in der grafischen Notation allerdings vielfältiger. Für die Erfüllung der genannten syntaktischen Ästhetikkriterien wie Kreuzungs- und Überdeckungsfreiheit, Kantenlängen, gleichmäßige Ausnutzung der Darstellungsfläche, etc. gibt es Algorithmen und Heuristiken aus dem Graphlayout [8, 6, 15, 18]. Allerdings handelt es sich hierbei meist um Layout eines bereits vollständig bekannten Graphen. Ein Framework für dynamisches Layout von Graphen, in denen Knoten und Kanten entfernt und hinzugefügt werden können, wird z. B. von Huang u. a. [17] präsentiert.

Bereits bei dem semantischen Kriterium der Boxgröße werden allerdings die Schwierigkeiten eines automatischen Layouts deutlich. Bei visuellen Modellierungswerkzeugen korrespondiert die Größe von Boxen oft mit ihrer semantischen Relevanz innerhalb des Systems. Dabei werden „unwichtige“ Operator-Boxen oft kleiner dargestellt, als „echte“ modulare Teilsysteme des Modells (siehe Abbildung 4.5). Die rekursive Struktur der grafischen Datenflusssprachen bietet allerdings auch dem Anwender die Möglichkeit, eigene Operatoren zu modellieren, die innerhalb seines Modells nur eine untergeordnete Rolle spielen können, so dass eine automatische Skalierung, ohne eine Wissensbasis über die Funktionsweise des Systems zu besitzen, unmöglich ist. Berner [3] vertritt daher den Ansatz, die Umsetzung dieser Kriterien ganz dem Benutzer zu überlassen, und ein automatisches Layoutverfahren zu entwickeln, welches diese Notation beibehält, aber nicht verändert. Furnas [12] definiert eine sogenannte *Degree-of-Interest*-Funktion (DOI), die abhängig von einem *a priori interest (API)* und einem Abstand D die Relevanz eines Objekts x bezüglich des aktuell fokussierten Elements y berechnet:

$$DOI(x|. = y) = API(x) - D(x, y)$$

Somit entspricht die benutzerdefinierte semantische Relevanz eines Objekts (hier: die Größe einer Box) dem API und die aus dem Abstand zum aktuellen Fokus errechnete Größe dem DOI. Will man das Layout vollständig automatisieren, so muss man zur Generierung einer API semantische Informationen aus syntaktischen

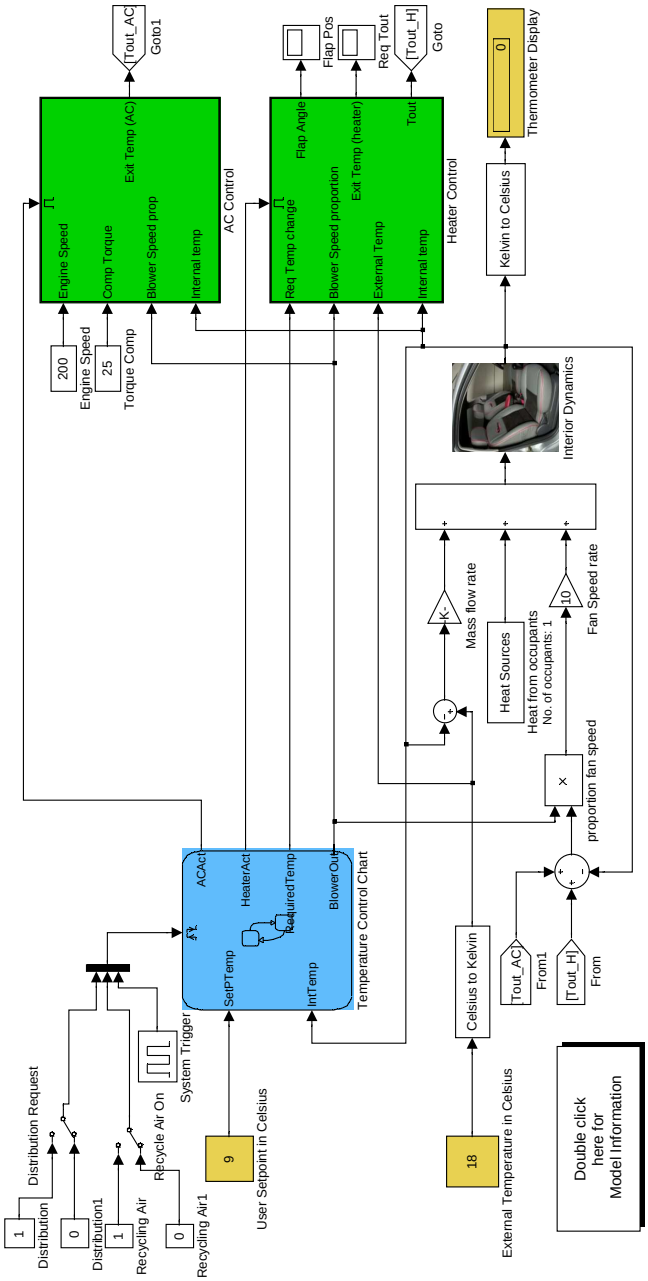


Abbildung 4.5: Beispielmodell einer Klimaautomatik in Simulink. Die Teilsysteme `AC Control` und `Heater Control` sind farblich hervorgehoben und nehmen eine größere Fläche als einfache Operatoren und Konstanten ein (SCADE).

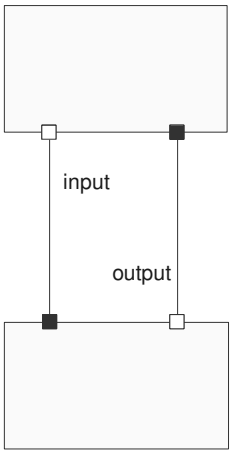


Abbildung 4.6: Werden Datenflusslabel generell rechts oberhalb des Konnektors platziert sind Mehrdeutigkeiten bei der Zuordnung ausgeschlossen (RoseRT).

„erraten“. Beispielsweise ließe sich der numerische Wert der API einer Box aus der Anzahl der enthaltenen Elemente und Unterhierarchien ableiten.

Ein weiteres Problem ist das Positionieren von Beschriftungen (Label). Im optimalen Fall sollte eine Zuordnung von Label und Objekt zu jeder Zeit eindeutig sein. Zusätzlich sollte die automatische Platzierung von Labels andere Objekte nicht verdecken, da ansonsten ein manuelles Verschieben der Beschriftung nötig wäre. Beispielsweise werden die Portlabels in den Werkzeugen Simulink und SCADE per default innerhalb der Box, direkt neben die Ports platziert, so dass nur bei einer extremen Verkleinerung der Box Objektüberschneidungen entstehen können.

RoseRT platziert die Label ähnlich, allerdings erfordert die große Länge der Beschriftungen (bestehend aus Name und *Classifier*) oft ein manuelles Verschieben durch den Benutzer, um verdeckte Bereiche anzuzeigen. Da in RoseRT nur 1 : 1-Verbindungen zwischen Boxen möglich sind, kann es hier sinnvoller sein, anstatt zwei Portlabels ein Datenflusslabel einzuführen und sinnvoll an die Datenflusslinie zu platzieren. Hierbei ist das Problem allerdings nur reduziert: es bleibt die Frage der Platzierung von Datenflussbeschriftungen. In Kapitel 2 wurde dieses Problem bereits angesprochen. Lösungsvorschläge lassen sich auch im Statechart-Layout finden.

Der Layouter der Statechart-Umgebung *KIEL* [21] vermeidet Objektüberlagerungen und platziert Labels ausgehend von der Pfeilrichtung rechts oberhalb der Verbindungslinie. Dies könnte auch in Datenflussmodellen angewendet werden (siehe Abbildung 4.6). Allerdings entstehen auf diese Weise z. B. bei der Verkleinerung/-Vergröberung von Boxen nicht immer überdeckungsfreie und somit lesbare Bereiche. Eine andere Möglichkeit ist das Platzieren des Labels direkt auf der Verbindungslinie, wobei es parameterabhängig sein kann, ob die Beschriftung mittig, am Anfang oder am Ende der Linie positioniert wird.

Unabhängig von dem Verfahren wird eine eventuelle Überdeckung innerhalb eines automatischen Layouts nie auszuschließen sein, weshalb weitere Techniken nötig werden. Berner [3] erwähnt in diesem Zusammenhang auch die Verwendung von Transparenztechniken, so dass keine Objektüberlagerungen im eigentlichen Sinne vorkommen und verdeckte Objekte weiterhin erkennbar bleiben.

4.4 Fokusbestimmung

Generell muss jeder Fokussierung eines Elements kognitiver Aufwand, also die Beantwortung der Frage nach der jeweiligen Relevanz eines Objekts vorangehen. Diese Relevanz verändert sich in der Regel mit der Zeit, so dass auch der Fokus angepasst werden muss. Neben dem kognitiven Aufwand fällt bei der Fokusbestimmung auch mechanischer Navigationsaufwand an. Gerade während der Ausführung einer Simulation ist ein häufiger Fokuswechsel nötig, um sämtliche relevante Daten aufzunehmen. Die nachfolgenden Abschnitte beschreiben den Weg von einer manuellen Fokusbestimmung mit hohem kognitiven und mechanischen Aufwand hin zu der Beschreibung einer automatischen Fokussierungstechnik, bei dem der mechanische und kognitive Aufwand stark reduziert werden kann.

4.4.1 Manuell

Bei der manuellen Fokus-Bestimmung wird die Fokussicht nur mit geringer Hilfe des Systems dem Benutzer zur Verfügung gestellt. Der Benutzer selektiert dabei – in der Regel mit der Maus – direkt im Modell das zu fokussierende Element, so dass das System die jeweilige Sicht bereitstellen kann. Dabei kann diese aktive Selektion auf unterschiedliche Weise geschehen. Zwei häufig genutzte Varianten werden hier kurz vorgestellt.

Focus-On-Click: Die bekannteste Art der Fokusbestimmung ist das Anklicken eines Objekts mit der Maus. Diese Fokussierung wird von fast allen Modellierungswerkzeugen bevorzugt. Allerdings lässt das verwendete Explosive-Zooming innerhalb der Werkzeuge auch keine Alternative zu.

Focus-Follows-Mouse: Hierbei wird der Fokus allein durch die Position des Mauszeigers bestimmt. Ein explizites Selektieren des Objekts ist nicht mehr nötig. Diese Technik setzt die Nutzung einer Fokus-und-Kontext-Technik voraus und erfordert von der Darstellungsumgebung hohe Natürlichkeit und „weiche“ Sichtwechsel, um Orientierungsprobleme zu vermeiden.

Auf Basis dieser Fokussierungstechnik ergeben sich zusätzliche Möglichkeiten, die zur Verbesserung der Visualisierung beitragen können. Vorstellbar ist ein Fokus-und-Kontext-Konzept, bei dem das Schweben des Mauszeigers (*Hovering*) über ein fokussierbares Objekt detailliertere Informationen anzeigt, aber die gröste Detailstufe erst beim Anklicken dargestellt wird. Da durch das hierarchische Konzept von Datenflusssprachen oft Teilsysteme wieder durch Teilsysteme visualisiert werden, kann sich dem Betrachter durch eine solche „Vorschau“ die Struktur eines Teilsystems erschließen, ohne Detailinformationen (mit erhöhtem Platzbedarf) darzustellen. Zusätzlich verringert es den physikalischen Navigationsaufwand und die Wahrscheinlichkeit einer „Sackgasse“ in der Navigation (Rückwärtsnavigation). Nachteil dieses Ansatzes kann der erhöhte Ressourcenbedarf bei der Berechnung der Sichten und die für das menschliche Auge ungewohnte und anstrengende häufige Anpassung der Modell-Visualisierung sein. Die *Show-When-Hovering*-Technik in Simulink kann als einfache nicht-grafische Variante dieser Technik interpretiert werden.

Ein weiteres Problem bei Fokusbestimmung durch Focus-Follows-Mouse ist das sogenannte *Overshooting* (Gutwin [13]). Dieses Phänomen tritt bei einer fokusbasierten Vergrößerung/Verkleinerung von Objekten auf und äußert sich dadurch, dass bei Fokussierung eines Objekts, sich dieses dem Mauszeiger entgegenbewegt und man so sehr schnell über das eigentliche Ziel „hinausschießt“. Als Lösung stellt Gutwin das sogenannte *speed-coupled flattening* vor, bei dem sich die Vergrößerung nach der Geschwindigkeit der Mausbewegung richtet. Bei zügigen Bewegungen ist der Grad der Vergrößerung nur sehr gering, während das Ausharren des Mauszeigers auf einem Objekt dieses nach kurzer Zeit vergrößert und in den Fokus bringt. Die Vorteile dieser Technik werden zusätzlich in einer empirischen Untersuchung nachgewiesen.

4.4.2 Aufzeichnung von Sichten

Als weiteres Hilfsmittel bei der Navigation und der Bestimmung eines Fokus dient die Möglichkeit der Aufzeichnung einzelner Perspektiven eines Modells. Zwar erlauben Simulink und SCADE das Navigieren zwischen den Sichten mit einer Art History-Mechanismus (Vorwärts, Zurück, oberste Hierarchie), das Speichern und Laden benutzerdefinierter Sichten ist aber nicht möglich. Eine solche Technik würde es dem Benutzer erlauben, ein Spektrum relvanter Sichten zu defnieren, um schnell zwischen den einzelnen Fokussen wechseln zu können. Benutzerdefinierte Bezeichnungen der gespeicherten Sichten würden eine direktere semantische Zuordnung erlauben und der kognitive Aufwand reduziert sich. Manuelle Fokusbestimmung erfordert häufig uneffizientes, sukzessives Navigieren durch mehrere Hierarchie-Ebenen. Die Speicherung einer Sicht ermöglicht eine direktere Navigation und verringert dadurch auch den mechanischen Aufwand. Bei Verwendung von „weichen“ Übergängen zwischen den Fokuswechseln bliebe auch der globale Kontext erhalten.

4.4.3 Suche

Die in Kapitel 3 vorgestellten Konzepte, insbesondere die (semantischen) Fokus-und-Kontext-Techniken helfen bei der Orientierung innerhalb eines Modells. Dennoch fällt die Suche nach bestimmten Elementen oder Teilsystemen schwer, da dies eine manuelle Fokussierung jedes einzelnen Teilbereichs durch den Benutzer erfordert. Zusätzlich kann sich die Position von Objekten auf der Arbeitsfläche durch Anwendung von Fokus-und-Kontext-Konzepten im Zusammenhang mit Layout-Verfahren verändern, ohne dass der Benutzer aktiv daran beteiligt ist. Daher rückt die Relevanz der grafischen Position von Objekten in den Hintergrund, und topologische Zusammenhänge in den Vordergrund. Der Entwickler muss in der Lage sein, ein bestimmtes Objekt zu fokussieren, ohne genaue Kenntnis über die grafische Position des Objekts zu haben. Die textuelle Eingabe von Suchkriterien in eine Suchmaske und die dadurch ermöglichte automatische Navigation könnte dabei helfen.

4.4.4 Automatische Fokusbestimmung

Semantisches Fokus-und-Kontext erlaubt die effektive Ausnutzung der begrenzten Arbeitsfläche ohne die Darstellung mit Information zu überladen. Der globale Kontext und die Details werden in einer gemeinsamen Sicht visualisiert. Der Betrachter fokussiert dazu einen Bereich, lässt Boxen aufklappen und verringert so den Abstraktionsgrad, um detailliertere Informationen zu erhalten. Dazu ist immer noch hoher Aufwand durch den Benutzer nötig, um den jeweiligen Fokus zu bestimmen und die Navigation durchzuführen. Insbesondere bei der Simulation von Modellen fällt die Navigation und Auswahl der Ansichten schwer, da die Relevanz eines Objekts zusätzlich vom Zeitpunkt und von der Veränderung der Daten innerhalb der Simulation abhängt. Wäre es möglich den aktuellen Fokus durch das System ermitteln und bereitstellen zu lassen, könnte der kognitive und mechanische Aufwand bei der Fokusbestimmung minimiert und die Analyse des Modells erleichtert werden.

Die Simulation dient in erster Linie zur Analyse der Daten eines Systems. Dabei geht es nicht um Syntaxfehler oder Fehler, die das sofortige Beenden der Simulation zur Folge hätten, sondern eher um die Evaluierung benutzerdefinierter Systemeigenschaften und Fehler im Systemverhalten. Der Benutzer beobachtet dazu in Scopes oder Tabellen die Daten an vorher ausgewählten Datenflüssen, während er die Eingaben an den Schnittstellen des Gesamtsystems verändert. So kann es beispielsweise in einem Teilsystem eines Modells wichtig sein, dass ein Wert innerhalb eines definierten Bereichs bleibt (z. B. die Beschleunigung eines Tempomats in PKW). Sobald die Bedingung verletzt ist, rückt dieses System in den Fokus, d. h. der Benutzer hat ein größeres Interesse an Detailinformationen dieses Systems und wird den Abstraktionsgrad in der Visualisierung verkleinern (z. B. durch semantisches Aufklappen der Box).

Daneben gibt es andere Bedingungen, die den Fokus noch klarer bestimmen lassen: ein Knoten (bzw. eine Box) eines asynchronen Datenflussmodells (z. B. in RoseRT) darf nach Definition erst dann seine Berechnung durchführen, wenn die Daten sämtlicher eingehenden Kanten verfügbar sind (siehe auch Kapitel 2). Das Element rückt in diesem Fall erst in den Fokus, wenn dieses Ereignis ausgelöst wurde. Bei speziellen Anwendungsfällen wie der Modellierung einer zeitgesteuerten Architektur (TTA) können diese *trigger events* durch den Empfang einer Nachricht ausgelöst werden.

Es sollte also die Möglichkeit geben, den Komponenten eines Modells benutzerdefinierte *trigger events* zuzuweisen, wodurch – im Sinne eines Fokus-und-Kontext-Ansatzes – Detailinformationen eines Teilsystems automatisch ein- und ausgeblendet werden können. Hierbei wird auch deutlich, dass eine polyfokale Visualisierung Voraussetzung ist und herkömmliche Techniken wie *Explosive Zooming* nicht geeignet sind.

Das automatische Fokussieren während der Simulation könnte das Verständnis für ein Modell verbessern und bei der Fehleranalyse helfen. Die häufig genutzte Black-Box-Simulation, bei der die Outputs in Abhängigkeit von den Inputs analysiert werden, wird auf diese Weise zu einer Grey-Box und die Funktionsweise des Gesamtsystems transparenter.

4.5 Zusammenfassung und Diskussion

Das grafische Layout ist ein wichtiger Aspekt bei der Entwicklung verständlicher Modelle. Die alleinige Umsetzung der „primären“ Notation berücksichtigt nicht die Platzierung und Gestaltung von Komponenten. Dabei hat sich herausgestellt, dass ungeübte Nutzer gestalterische Fehler machen, die die Lesbarkeit eines Modells verschlechtern können. Erfahrene Anwender verbringen dagegen einen nicht unwesentlichen Teil der Entwicklungszeit damit, Objekte unter Berücksichtigung gewisser Ästhetikkriterien zu platzieren um versteckte Zusammenhänge deutlich zu machen oder die Lesbarkeit (z. B. durch möglichst wenig Kantenkreuzungen) zu erhöhen. Analog zu den textuellen Programmiersprachen hat sich dabei der Begriff der *Sekundärnotation* etabliert. Es existieren bereits Styleguides, in denen umfangreiche Layoutregeln definiert sind. Als Quasi-Standard hat sich hierbei der MAAB-Styleguide in vielen Bereichen der Industrie durchgesetzt. Allerdings erfordert die Einhaltung dieser Regeln einen hohen Aufwand bei der Entwicklung. Als unterstützende Werkzeuge bieten sich daher Stylechecker an, die das Modell bezüglich der Styleguides verifizieren. Ein weiterer Schritt zu einem effizienten Layout wäre die automatische Umsetzung der Styleguides durch Pretty-Printer, so dass auch der mechanische Aufwand bei der manuellen Platzierung von Objekten verringert wird. Die Beispiele der Kriterien *Objektgröße* und *Labelplatzierung* machen allerdings die Schwierigkeiten bei der Entwicklung eines automatischen Layouts deutlich.

5 Grundlagen der Implementierung

Neben der Beurteilung der Werkzeuge bezüglich ihrer Visualisierungen und der theoretischen Diskussion über die Verbesserung der Informationsdarstellung ist eine genaue Analyse der Realisierbarkeit der vorgestellten Konzepte erforderlich. Dieses Kapitel soll eine möglichst gute Grundlage über die Möglichkeiten einer Implementierung bieten. Um das Rad nicht gänzlich neu zu erfinden, wird speziell eine Erweiterung vorhandener Werkzeuge um Visualisierungstechniken betrachtet. Hierfür ist es notwendig die vorhandenen Entwicklungswerkzeuge zu analysieren, um mögliche Schnittstellen und eventuelle Beschränkungen zu finden. Die zu untersuchenden Kandidaten sind die bereits bekannten Tools Simulink, SCADE und RoseRT.

Die vorherigen Kapitel zeigten, dass eine semantische Fokus-und-Kontext-Technik vorteilhaft bei der Darstellung von Datenflussmodellen sein kann. Daher bezieht sich dieses Kapitel speziell auf eine Umsetzung dieser Technik. Bevor die Schnittstellen der einzelnen Werkzeuge erläutert werden, folgt zunächst eine Bestandsaufnahme der nötigen Anforderungen an die Programmierschnittstellen (*APIs*):

Grafische Manipulation Ein API (*application programming interface*) muss die Möglichkeit der Manipulation grafischer Objekte bieten. Dabei müssen mindestens grundlegende Eigenschaften wie Position und Aussehen modifizierbar sein.

Anpassen des Event-Handlings Fokus-und-Kontext erfordert die Neuinterpretation von Benutzereingaben. D.h. vorhandene Callback-Funktionen von Maus- und Tastatur-Events sollten überladen werden können, um u.a. die Technik des Auf- und Zuklappens von hierarchischen Boxen realisieren zu können.

Animation Da Animationen in Fokus-und-Kontext einerseits wichtig zur Kontexterhaltung sind, aber andererseits bislang wenig bis gar nicht in den Werkzeugen eingesetzt werden ist eine diesbezügliche Analyse der APIs notwendig. Die Vorteile von Fokus-und-Kontext ohne Animation gegenüber herkömmlichen Visualisierungstechniken sind eher gering einzustufen und eine Umsetzung wäre nicht sinnvoll.

Objektorientierung Die Datenfluss-Elemente werden in der Regel um weitreichende und komplexe grafische Eigenschaften erweitert. Daher ist es sinnvoll, im Sinne des OOP von vorhandenen Objekten erben zu können, um wichtige

(semantische und syntaktische) Eigenschaften nicht zu verlieren oder neu implementieren zu müssen.

5.1 Matlab/Simulink

Matlab/Simulink/Stateflow bietet unterschiedliche Möglichkeiten auf Modelleigenschaften Einfluss zu nehmen. Zum einen werden die Modelldaten in Klartext in einer MDL-Datei gespeichert. Diese enthält auch grafische Attribute, die beim Laden in den Speicher aktualisiert werden. Leider sind auf diese Weise die Möglichkeiten sehr beschränkt und Event-Handling nicht möglich. Daher ist dies keine brauchbare Lösung, eine dynamische Visualisierung mit Animation und Benutzerinteraktion zu integrieren.

Das Werkzeug stellt allerdings noch eine andere Schnittstelle zur Verfügung: das *Stateflow API*. Über die Matlab-Konsole kann während der Laufzeit auf die Eigenschaften eines Modells Einfluss genommen werden. Zusätzlich lassen sich diese Kommandos in m-Skripte zusammenfassen, um automatisierte Abläufe zu integrieren. Obwohl der Name des API suggeriert, dass nur die Manipulation von Stateflow-Charts möglich ist, lassen sich auch reine Datenflussmodelle durch ein solches Skript generieren. Darüber hinaus ermöglicht das API das Überladen von Event-Handling-Funktionen (siehe Callback-Funktions-Tabelle in [29], Seiten 203 ff.), so dass sich das Verhalten von Elementen bezüglich Benutzereingaben durch eigene Skripte anpassen lässt. Die Listings 5.1 und 5.2 zeigen, wie einfach es ist, ein beliebiges selektiertes Objekt per Doppelklick zu vergrößern.

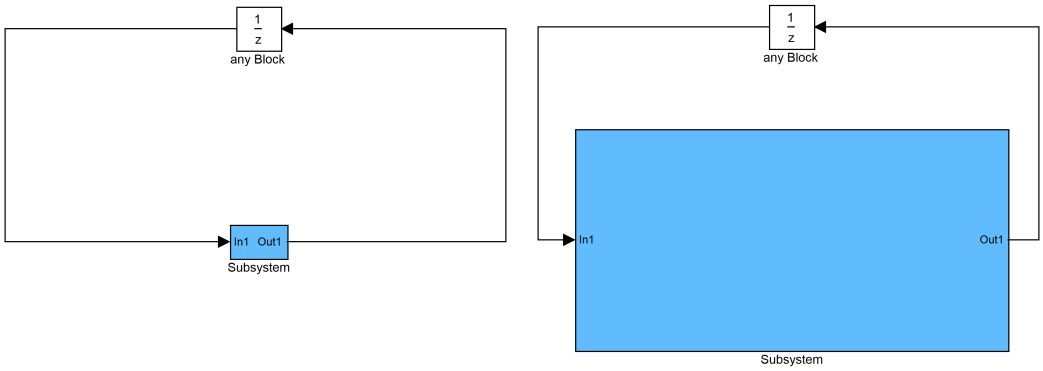
```
function explodeSubsystem(obj, scale)
% Skaliert Subsystem mit Handle 'obj' um 'scale' Einheiten
% zu jeder Seite
% Beispiel:
% explodeSubsystem(gcb, 30)
position = get_param(obj, 'Position'); % Hole aktuelle Positionsdaten
                                       % der der Bounding-Box:
                                       % [x1 y1 x2 y2]

width = position(3) - position(1);  % Berechne Breite des Subsystems
height = position(4) - position(2); % Berechne Hoehe des Subsystems
factor = width/height;              % Seitenverhaeltnis

% Skaliere Element 'obj' bei gleichbleibendem Seitenverhaeltnis
set_param(obj, 'Position',[
                position(1)-round(scale*factor)   % Ecke oben links, x-Achse
                position(2)-scale                 % Ecke oben links, y-Achse
                position(3)+round(scale*factor)   % Ecke unten rechts, x-Achse
                position(4)+scale]);              % Ecke unten rechts, y-Achse
```

Listing 5.1: explodeSubsystem.m

```
function myOpenFcn
% Ersetzt Standard-Callback-Funktion OpenFcn

t = timer('Period', 0.01,
          'ExecutionMode', 'fixedRate',
          'TasksToExecute', 15,
          'Tag', 'AnimationTimer');

t.TimerFcn = 'explodeSubsystem(gcb,5)'; % bei jedem Aufruf wird das aktuell
                                        % selektierte Objekt (gcb) um 5
                                        % Einheiten an jeder Seite
                                        % vergroessert.

t.StopFcn = 'deleteTimer';              % 'deleteTimer entfernt das Timer-
                                        % Objekt aus dem Speicher, sobald
                                        % die Animation beendet ist

start(t);                               % Timer starten
```

Listing 5.2: myOpenFcn.m

Zusätzlich muss nur noch die Standard-Callback-Funktion `OpenFcn` durch `myOpenFcn` ersetzt werden. Dies kann direkt im Modell in den *Block Properties* geschehen, oder ebenfalls per m-Skript. Das Ergebnis ist eine relativ flüssig animierte Vergrößerung bzw. Verkleinerung wie es in Abbildung 5.1 dargestellt wird. Ein weiteres Problem



(a) **(b)**

Abbildung 5.1: Einfaches Beispiel in Simulink: (a) Modell im Ausgangszustand. (b) ein Doppelklick auf das Subsystem vergrößert dieses um einen definierbaren Faktor.

bei der Implementierung von Fokus-und-Kontext ist die Einbettung von Objekten in die Fokus-Sicht eines Subsystems. Zwar bietet das Stateflow API den Zugriff auf alle Kind-Objekte eines Subsystems, aber es gibt keine Möglichkeiten, diese in eine andere zu kopieren oder zu verlinken (außer, es handelt sich um eine Library). Da das Simulink-Framework diese weitgreifenden Eingriffe nicht vorsieht, bleibt nur die Alternative der Verwendung von (statischen) Maskierungen. Wie in Kapitel 2 bereits beschrieben, lassen sich in diesen Masken Vektorgrafiken und grafische Symbole platzieren, so dass ein Abbild des Subsystem-Inhalts konstruiert werden kann. Fraglich ist erstens, ob dies effizient lösbar ist und zweitens, ob eine rein statische

Darstellung des Subsystem-Inhalts ohne Interaktionsmöglichkeiten wünschenswert ist.

Aufgrund der mathematischen Vielfältigkeit der Matlab-Kommandosprache sollte es möglich sein, Algorithmen für das automatische Layout von Modellen zu implementieren. Allerdings beschränkt vermutlich auch hier das Stateflow API zu sehr die Möglichkeiten. So ist bei der Definition und Platzierung einer Verbindung zwischen zwei Blöcken nur die Angabe von Start- und Endport möglich. Der Streckenverlauf wird automatisch durch das Simulink-Framework generiert. Der zugrundeliegende Layout-Algorithmus hat allerdings Schwächen und verursacht Konstellationen, die in einigen Fällen die Ästhetikkriterien aus Kapitel 4.1 verletzen. Darüber hinaus existieren für viele grafische Eigenschaften (z. B. die Portposition) nur Leserechte. Eine Anpassung ist somit nicht möglich.

5.2 SCADE

SCADE speichert seine Modelle ebenfalls im Klartext in SAOFD-Dateien ab. Dort sind grafische Daten der verwendeten Elemente enthalten, wodurch – analog zu Simulink – allerdings nur eingeschränkte Manipulationen möglich sind.

Zusätzlich bietet SCADE die Anpassung der IDE durch eine Teilmenge der TCL-Skriptsprache an. Dadurch werden weitgreifendere Anpassungen möglich. Im sogenannten *SCADE UML Metamodel* [11] sind sämtliche in SCADE verwendeten (Modell-)Objektklassen und Eigenschaften in UML-Klassendiagrammen dokumentiert. Bei den SCADE-spezifischen TCL-Befehlen wird zwischen API- und GUI-bezogenen Kommandos unterschieden. In Tabelle 5.1 werden als Beispiel die *API-Related TCL Commands* vorgestellt. Die Befehle bezüglich der Erweiterung der SCADE-GUI sind umfangreicher und werden im Handbuch [10] dokumentiert. Die Nutzung von TCL-Skripts erfordert die vorherige Registrierung in SCADE. Dies kann direkt im Werkzeug durch den Script-Wizard geschehen, oder manuell in der Windows-Registrierung durchgeführt werden. Neben der Anpassung von Objekteigenschaften des Metamodells bieten sich zusätzliche Möglichkeiten zur Veränderung der SCADE-GUI. So lassen sich neue Menüeinträge, Dialog-Boxen und weitere Standard-Kontrollelemente hinzufügen.

5.3 RoseRT

RoseRT bietet vielfältige Möglichkeiten, das Erscheinungsbild des Werkzeugs zu verändern. Es ist möglich dem Hauptmenü der Oberfläche neue Untermenüs hinzuzufügen, indem die Datei `rosert.mnu` angepasst wird. Zusätzlich können durch Rational Rose RealTime Scripts Diagramme, Klassen, Capsules, etc. automatisch

Funktion	Beschreibung
AttributesOf	Gibt eine Liste von Attributen des Klassennamens zurück. (bezüglich des Metamodells)
Call	Ruft eine Methode eines Objekts auf und gibt das Ergebnis zurück
Class	Gibt den Klassennamen eines Objekts zurück (bezüglich des Metamodells)
Classes	Gibt die Liste von Klassen zurück, die der TCL-Interpreter kennt (bezüglich des Metamodells)
Get	Gibt den Wert eines Objekt-Attributes zurück
GetRole	Gibt das Objekt zurück, welches mit einem anderen assoziiert werden kann (z. B. der Ordner einer Datei)
MapRole	Ruft eine beliebige Prozedur für alle Objekte einer angegebenen Gruppe auf
RolesOf	Gibt eine Liste von Aufgaben (*roles*) zurück, die von einer angegebenen Klasse unterstützt werden.

Tabelle 5.1: API-bezogene TCL-Kommandos in SCADE

erstellt werden. Zugriff von externen Werkzeugen wird über das Rational Rose RealTime Automation Object ermöglicht, so dass in einer externen Anwendung Funktionen in RoseRT ausgeführt werden können. Zusätzlich stellt Rational eine Bibliothek (`\$RATIONALHOME$\Rose RealTime\bin\win32\RtRes.dll`) zur Verfügung, so dass Zugriff auf RoseRT-Klassen und deren Eigenschaften und Methoden möglich wird. Diese Schnittstelle wird *Rational Rose RealTime Extensibility Interface (RRTEI)* genannt. In Abbildung 5.2 werden die Komponenten des RRTEI und ihre Zusammenhänge gezeigt. Als ein einfaches Beispiel enthält das Listing 5.3 den Code zur Erzeugung einer Capsule in einer laufenden RoseRT-Sitzung aus einer externen Visual-C++-Anwendung.

Um hohe Wiederverwendbarkeit und Plattformunabhängigkeit zu erreichen, ist es möglich, durch geeignete Middleware (z. B. *J-Integra* [19]) einen Zugriff von Java auf das RRTEI herzustellen. Durch die Möglichkeiten der OOP lassen sich vorhandene Elemente um komplexe grafische Eigenschaften erweitern. Die verfügbare Dokumentation beschreibt zwar einige, aber leider nicht alle Klassen und Methoden. Es kann daher hier nicht eindeutig geklärt werden, ob alle relevanten Modifikationen und Erweiterungen zur Realisierung von Fokus-und-Kontext möglich sind. Hier muss noch viel Arbeit in die Evaluation einfließen.

5.4 Zusammenfassung

Die Realisierung von Fokus-und-Kontext als Visualisierungskonzept in Datenflussmodellen erfordert umfangreiche Möglichkeiten zur grafischen Manipulation der Ob-

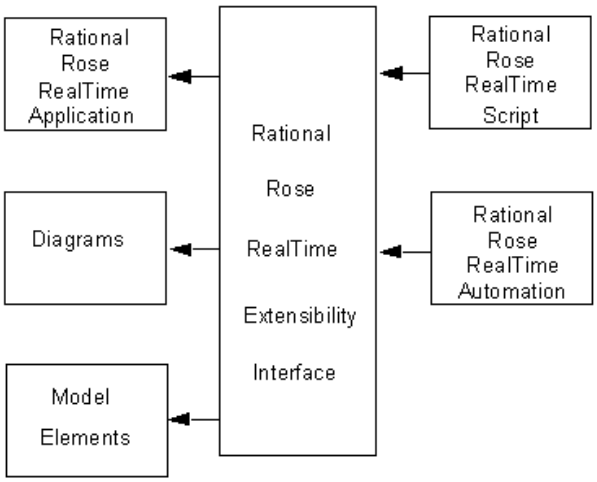


Abbildung 5.2: Die Komponenten des Rational Rose Extensibility Interface

jekte. Die Modellierungswerkzeuge bieten häufig Schnittstellen zur Anpassung des GUI. Zusätzlich ist es notwendig, Zugriff auf das Zeichenframework zu erhalten. Meist sind hier jedoch die Möglichkeiten zur Veränderung sehr begrenzt. Simulink, SCADE und RoseRT liefern jeweils Schnittstellen zur Manipulation der grafischen Komponenten, wobei der Freiheitsgrad unterschiedlich ausfällt. Während Simulink durch ein intuitives Event-Handling besticht, durch das sehr schnell dynamische Ergebnisse erzielt werden können, lassen sich tiefgreifende grafische Manipulationen nicht vornehmen. So verhindert das Stateflow-API beispielsweise das Überschreiben der Positionswerte eines Ports. Zudem ist fraglich, ob die hinzukommenden Änderungen und Animationen für Fokus-und-Kontext in dem grafischen Framework (basierend auf Matlab-Skript) performant realisierbar sind. Das Implementierungs-Beispiel zur Vergröberung/Verkleinerung eines Subsystems hat bereits kleine Probleme bei der Geschwindigkeit, sowie bei der Kontinuität der Animation, da das Framework als kleinsten Skalierungsschritt nur 5 Pixel zulässt.

SCADE unterstützt die Manipulation seiner Modellkomponenten durch die TCL-Skriptsprache. Der Zugriff auf die Objekte und ihre grafischen Eigenschaften ist durch ein UML-Metamodell dokumentiert. Diese Skripte müssen im Windows-Betriebssystem oder direkt in SCADE registriert werden und können somit auch direkt bei Programmstart als eine Art Plug-In zur Verfügung gestellt werden. Die wahre Mächtigkeit von TCL-Skripts bezüglich der grafischen Anpassung wird hier noch nicht deutlich und muss erst weiter untersucht werden. Nicht dokumentiert und vermutlich proprietär ist die Nutzung des TCL-Grafikframeworks TK, auf dem die SCADE-Grafik basiert.

RoseRT bietet größtmögliche Freiheiten bei grafischen Anpassungen durch Bereitstellung einer Funktions-Bibliothek, die in Visual-C++-Programme eingebunden werden kann. Es existiert geignete Middleware für andere Sprachen, die als Schnitt-

```
#include "stdafx.h"
  CLSID RoseRTID;
  HRESULT hr;
  hr = CLSIDFromProgID(L"RoseRT.Application", &RoseRTID);
  if (FAILED(hr))
  {
    CoUninitialize();
    /* Fehlerbehandlung */
  }
  LPUNKNOWN lpUnk;
  hr = GetActiveObject(RoseRTID, NULL, (LPUNKNOWN*)&lpUnk);
  if (FAILED(hr))
  {
    CoUninitialize();
    /* Fehlerbehandlung */
  }
  LPDISPATCH lpDispatch;
  hr = lpUnk->QueryInterface(IID_IDispatch,
                             (LPVOID FAR*)&lpDispatch);
  lpUnk->Release();
  if (FAILED(hr))
  {
     CoUninitialize();
    /* Fehlerbehandlung */
  }
  _Application roseRtApp(lpDispatch);
  {
    Model theModel = roseRtApp.GetCurrentModel();
    LogicalPackage thePackage;
    thePackage = theModel.GetRootLogicalPackage();
    CapsuleCollection theCollection;
    theCollection = thePackage.GetCapsules();
    if (theCollection.FindFirst("Top") == 0)
    {
      Capsule theCapsule;
      theCapsule = thePackage.AddCapsule("Top");
    }
  }
}
```

Listing 5.3: : Erzeugung einer RoseRT-Capsule aus einer externen Visual-C++-Umgebung

stelle zwischen dem Windows-COM-Interface und der Sprache fungiert. So lässt sich das Werkzeug zusammen mit der Nutzung von *J-Integra* komplett über eine externe Java-Anwendung steuern. Der Nachteil der RoseRT-Schnittstelle ist die unvollständige Dokumentation. Sie erklärt in erster Linie die Slave-Simulation aus externen Anwendungen heraus und nicht die Manipulation der internen Grafikkomponenten.

6 Zusammenfassung und Ausblick

Die grafische, modellbasierte Entwicklung von Systemen erfordert mächtige IDEs, bei denen neben einer ergonomischen Handhabung die Visualisierung der Modelle eine große Rolle spielt. Heutige Werkzeuge wie SCADE, Simulink und RoseRT basieren eher auf konservativen Konzepten zur Visualisierung von Datenflussmodellen, so dass die Effizienz von grafischen Sprachen nicht vollständig ausgenutzt wird. Die Evaluation alternativer Visualisierungstechniken zeigt, dass die Nutzung semantischer Fokus-und-Kontext-Techniken eine sinnvolle Ergänzung ist. Das Modell kann auf diese Weise effizienter entwickelt werden und ist leichter verständlich. Darüber hinaus verhilft die grafische Abstraktion durch *Levels-Of-Detail* zu einer Verringerung des Platz- und Informationsdichteproblems, so dass die Darstellungsfläche optimal genutzt wird. Eine weitere wichtige Erkenntnis ist, dass weder bildliche Repräsentationen den textuellen Beschreibungen in allen Bereichen überlegen sind, noch umgekehrt. Die gleichzeitige textuelle und grafische Entwicklung in einem gemeinsamen Werkzeug kann daher ebenfalls zu Effizienzgewinnen führen.

Für letztere Technik ist ein automatisches Layout Voraussetzung, während es für Fokus-und-Kontext einen erheblichen Zusatznutzen verspricht. Anhand von Ästhetikkriterien wurde definiert, wodurch ein gutes Modell-Layout charakterisiert ist. Dabei lassen sich einige generelle Kriterien durch Algorithmen und Heuristiken aus dem Graph- und Statechart-Layout in leicht modifizierter Form erfüllen, wobei für Spezialitäten der Datenflussmodelle wie z. B. Position der Portlabel und Boxgrößen eigene Lösungen entwickelt werden müssen.

Neben den Problemen eines manuellen Layouts von grafischen Komponenten, ist auch die Bestimmung des aktuell relevanten Fokusses mechanisch und kognitiv aufwändig. Zwar lässt sich der Fokus während des Editierens nicht völlig automatisch bestimmen, da dieser in der Regel rein benutzerabhängig ist, allerdings kann durch die Implementierung einer Suchmöglichkeit für Komponenten und einer automatischen Bereitsstellung der gewünschten Sicht der mechanische Aufwand stark verringert werden. In der Simulation wird die Relevanz eines Fokusses zusätzlich durch Zeitpunkte und Daten bestimmt und eine Automatisierung wird durch die Definition sogenannter *event trigger* möglich. Dadurch wird die Transparenz eines Modells erhöht und man erhält zusammen mit einer geeigneten Visualisierungstechnik die Möglichkeit zu Grey-Box-Simulationen. Ein interessanter Aspekt für weitere Forschungen in diesem Zusammenhang wäre die Analyse einer geeigneten Klassifizierung in modellunabhängige und modellspezifische *event trigger*. So ließe sich –

analog zu den syntaktischen Ästhetikkriterien – eine *a priori* festgelegte Regelbasis definieren, die für alle Modelle unabhängig anwendbar ist.

Im letzten Kapitel wurden die Grundlagen einer Realisierung der Konzepte disuktiert und die Möglichkeiten der Kopplung mit den vorhandenen Werkzeugen. Hier wurde festgestellt, dass die Schnittstellen häufig nur schlecht dazu geeignet sind, grafische Manipulationen der Komponenten durchzuführen. Allerdings geben die Dokumentationen wenig Aufschluss darüber, was wirklich möglich ist, so dass weitere Untersuchungen in diesem Bereich nötig werden. Eine Alternative wäre die Realisierung einer von den Werkzeugen getrennten grafischen Umgebung mit Schnittstellen zu den Werkzeugen zum Im- und Export von Modellen sowie für die Simulation. Eine Schwierigkeit dabei ist die Integration der teilweise unterschiedlichen Semantiken (siehe Kapitel 2) in eine einheitliche, eindeutige Syntax.

Unabhängig von der Entwicklung eines eigenen Werkzeugs fällt zusätzlicher Arbeitsaufwand auch bei der Definition der Detailgrade (LOD) für Datenflussmodelle an. So müssen die Modell-Informationen in geeignete Level eingeteilt werden. Dabei ist das Kernproblem, herauszufinden, welche Informationen zur Erhaltung des Kontexts und welche für das Detailwissen relevant sind. Neben der Abstraktion der Standardkomponenten Box und Datenfluss sowie ihrer dialekt-abhängigen Spezialitäten sollte auch die Datenvisualisierung in den LOD eine Rolle spielen. So ist es vorstellbar, dass während einer Simulation in einem Level mit hohem Detailgrad auch Daten (z. B. in kleinen Scopes an der Datenflusslinie) eingeblendet werden.

In dieser Arbeit wurde nur in geringem Maße darauf eingegangen, wie die Darstellung großer Datenmengen verbessert werden kann. Generell ist die Verwendung von Tabellen gut geeignet, wobei das Platzproblem ein beschränkender Faktor sein kann. Das Prinzip der Fisheye-Views oder Fokus-und-Kontext lässt sich allerdings auch auf diese Darstellungen übertragen. Mit der von Rao und Card [32] entwickelten *Table Lens* existiert in diesem Zusammenhang eine interessante Fisheye-View Technik für Tabellen, bei der die textuelle und grafische Repräsentation miteinander fusioniert.

Nicht zuletzt wären zukünftige empirische Untersuchungen über den zusätzlichen Nutzen von semantischem Fokus-und-Kontext in Datenfluss sinnvoll, so dass fehlerhafte und für die Praxis unnötige Konzeptvorschläge nicht weiter betrachtet werden.

A Literaturverzeichnis

[1] Altova: *Company Homepage*. http://www.altova.com

[2] Beard, D. ; Walker, J.: Navigational techniques to improve the display of large two-dimensional spaces. In: *Behaviour & Information Technology* 9 (1990), November, Nr. 6, S. 451–466

[3] Berner, Stefan: *Modellvisualisierung für die Spezifikationssprache ADORA*. Universität Zürich, Schweiz, Wirtschaftswissenschaftliche Fakultät, Dissertation, Juni 2002

[4] Brooks Jr., Frederick P.: No silver bullet: essence and accidents of software engineering. In: *Computer* 20 (1987), Nr. 4, S. 10–19. – ISSN 0018-9162

[5] Card, Stuart K. ; Mackinlay, Jock ; Shneiderman, Ben: *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, Januar 1999. – ISBN 1558605339

[6] Castelló, Rodolfo ; Mili, Rym ; Tollis, Ioannis G.: ViSta. In: Mutzel, P. (Hrsg.) ; Jünger, M. (Hrsg.) ; Leipert, S. (Hrsg.): *Graph Drawing : 9th International Symposium, GD 2001* Bd. 2265. Springer, 2002, S. 481–482. – URL http://link.springer-ny.com/link/service/series/0558/bibs/2265/22650481.htm;http://link.springer-ny.com/link/service/series/0558/papers/2265/22650481.pdf. – ISSN 0302-9743

[7] Cox, D. ; Chugh, Jasdeep S. ; Gutwin, Carl ; Greenberg, Saul: The Usability of Transparent Overview Layers. In: *Companion Proceedings of the CHI '98 Conference on Human Factors in Computing Systems*, ACM Press, 1997, S. 301–302

[8] Davidson, Ron ; Harel, David: Drawing graphs nicely using simulated annealing. In: *ACM Transactions on Graphics* 15 (1996), Oktober, Nr. 4, S. 301–331. – URL http://www.acm.org/pubs/toc/Abstracts/0730-0301/234538.html. – ISSN 0730-0301

[9] Efroni, Sol ; Harel, David ; Cohen, Irun R.: Reactive Animation: Realistic Modeling of Complex Dynamic Systems. In: *Computer* 38 (2005), Januar, Nr. 1, S. 38–47

[10] Esterel Technologies (Veranst.): *SCADE Technical Manual*. 5.1. Februar 2006

[11] Esterel Technologies (Veranst.): *SCADE UML Metamodel – Technical Reference Card*. 5.1. März 2006

[12] Furnas, Gerge W.: Generalized Fisheye Views. In: *Human Factors in Computings Systems CHI '86 Conference Proceedings*, 1986, S. 16–23

[13] Gutwin, Carl: Improving focus targeting in interactive fisheye views. In: *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 2002, S. 267–274. – ISBN 1-58113-453-3

[14] Halbwachs, Nicolas ; Caspi, Paul ; Raymond, Pascal ; Pilaud, Daniel: The synchronous data-flow programming language LUSTRE. In: *Proceedings of the IEEE* 79 (1991), September, Nr. 9, S. 1305–1320. – URL `http://citeseer.nj.nec.com/halbwachs91synchronous.html`

[15] Harel, D. ; Yashchin, G.: An Algorithm for Blob Hierarchy Layout. In: *The Visual Computer* 18 (2002), S. 164–185

[16] Herndon, Kenneth P. ; Zeleznik, Robert C. ; Robbins, Daniel C. ; Conner, D. B. ; Snibbe, Scott S. ; Dam, Andries van: Interactive shadows. In: *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 1992, S. 1–6. – ISBN 0-89791-549-6

[17] Huang, Xiaodi ; Eades, Peter ; Lai, Wei: A framework of filtering, clustering and dynamic layout graphs for visualization. In: *ACSC '05: Proceedings of the Twenty-eighth Australasian conference on Computer Science*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2005, S. 87–96. – ISBN 1-920-68220-1

[18] Huang, Xiaodi ; Lai, Wei: Force-transfer: a new approach to removing overlapping nodes in graph layout. In: *ACSC '03: Proceedings of the twenty-sixth Australasian conference on Computer science*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2003, S. 349–358. – ISBN 0-909-92594-1

[19] Intrinsyc Software: *J-Integra*. November 2006. – URL `http://j-integra.intrinsyc.com/`

[20] Keahey, T.A.: The Generalized Detail-In-Context Problem. In: *INFOVIS* (1998), S. 44. – ISSN 1522-404X

[21] KIEL Project, The: *Project Homepage*. 2004. – URL `http://www.informatik.uni-kiel.de/~rt-kiel/`. – Kiel Integrated Environment for Layout

[22] Kreuseler, Matthias ; Lopez, Norma ; Schumann, Heidrun: A Scalable Framework for Information Visualization. In: *INFOVIS '00: Proceedings of the IEEE Symposium on Information Vizualization 2000*. Washington, DC, USA : IEEE Computer Society, 2000, S. 27. – ISBN 0-7695-0804-9

[23] Lamping, John ; Rao, Ramana: Visualizing large trees using the hyperbolic browser. In: *CHI '96: Conference companion on Human factors in computing*

systems. New York, NY, USA : ACM Press, 1996, S. 388–389. – ISBN 0-89791-832-0

[24] Lamport, Leslie: *LaTEX A Document Preparation System*. Adddison-Wesley, 1994

[25] Lee, Edward A. ; Messerschmitt, David G.: Synchronous Data Flow. In: *Proceedings of the IEEE* Bd. 75, IEEE Computer Society Press, September 1987, S. 1235–1245. – ISSN 0018-9219

[26] Leung, Y. K. ; Apperley, M. D.: A Review and Taxonomy of Distortion-Oriented Presentation Techniques. In: *ACM Transactions on Computer-Human Interaction* 1 (1994), Juni, Nr. 2, S. 126–160

[27] Mackinlay, Jock D. ; Card, Stuart K. ; Robertson, George G.: Rapid controlled movement through a virtual 3D workspace. In: *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1990, S. 171–176. – ISBN 0-201-50933-4

[28] Mackinlay, Jock D. ; Robertson, George G. ; Card, Stuart. K.: The Perspective Wall: Detail and context smoothly integrated. In: *Proc. of CHI '91*, April 1991, S. 173–179

[29] Mathworks Inc.: *Simulink – Simulation and Model-Based Design*. 6.5R2006b. Natick, MA: The Mathworks, Inc. (Veranst.), September 2006. – URL `http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_using.pdf`

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

[31] Petre, Marian: Why looking isn't always seeing: readership skills and graphical programming. In: *Communications of the ACM* 38 (1995), Juni, Nr. 6, S. 33–44

[32] Rao, Ramana ; Card, Stuart K.: The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In: *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1994, S. 318–322. – ISBN 0-89791-650-6

[33] Rational Rose Realtime: *IBM*. – URL `http://www.rational.com/rosert`

[34] Robertson, George G. ; Mackinlay, Jock D. ; Card, Stuart K.: Cone Trees: animated 3D visualizations of hierarchical information. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, 1991, S. 189–194. – ISBN 0-89791-383-3

[35] RTCA/EUROCAE (Veranst.): *Software Considerations in Airborne Systems and Equipment Certification*. Dezember 1992

[36] Sarkar, Manojit ; Brown, Marc H.: Graphical Fisheye Views of Graphs. In: *Proceedings of the ACM SIGCHI 1992 Conference on Human Factors in Computing Systems*, 1992, S. 83–91

[37] Siebenhaller, Martin: *Automatisches Layout von UML-Klassendiagrammen*. Eberhard-Karls-Universität Tübingen, Fakultät für Informations- und Kognitionswissenschaften, Diploma thesis, Juni 2003

[38] Spence, R. ; Apperley, M.: Data Base Navigation: An Office Environment for the Professional. In: *Behavior and Information Technology* 1 (1982), Nr. 1, S. 43–54

[39] Stasko, John T. ; Domingue, John B. ; Brown, Marc H. ; Price, Blaine A.: *Software Visualization – Programming as a Multimedia Experience*. Cambridge, MA, USA : MIT Press, 1997. – ISBN 0262193957

[40] The Object Management Group: *UML Homepage*. – URL `http://www.uml.org/`

[41] Wanger, Leonard: The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In: *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*. New York, NY, USA : ACM Press, 1992, S. 39–42. – ISBN 0-89791-467-8