Christian-Albrechts-Universität zu Kiel

Diplomarbeit

Überprüfung Syntaktischer Robustheit von Statecharts auf der Basis von OCL

cand. inf. Ken Bell
(Mat.Nr. 574455)

November 2006

Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:
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 Verbreitung von eingebetteten Echtzeitsystemen wird immer größer. Insbesondere die Systeme, die zur Steuerung von sicherheitskritischen Anwendungen entwickelt werden, sollten fehlerfrei sein. Zur Fehlervermeidung in der Softwareentwicklung gibt es verschiedene Style-Guides, mit denen eine Programmiersprache auf einen Umfang eingeschränkt wird, von dem man annimmt, dass dieser weniger fehleranfällig ist. Statecharts sind ein Formalismus zur Beschreibung reaktiver, eingebetteter Systeme. Klassische, d. h. für textuelle Programmiersprachen entwickelte Style-Guides, lassen sich nicht ohne weiteres auf Statecharts übertragen. Diese Diplomarbeit befasst sich mit der Fehlervermeidung in der Softwareentwicklung mit Statecharts.

Danksagungen

Als erstes möchte ich Steffen Prochnow für seine unermüdliche Betreuung danken. Die Diskussionen mit ihm haben die vorliegende Arbeit ermöglicht.

Als nächstes danke ich meinen Eltern, die mich stets unterstützt haben.

Herzlicher Dank gilt natürlich auch meinen beiden Korrektoren: Christiane Stiefel und Christopher Hlubek.

Abschließend möchte ich mich bei der b+m Informatik AG für die freundliche Zusammenarbeit bedanken.

Inhaltsverzeichnis

Tabellenverzeichnis xi

Abbildungsverzeichnis xiii

Verzeichnis der Auflistungen xv

1. Einleitung 1
1.1. Verwandte Arbeiten 4

2. Entwicklung eingebetteter, reaktiver Systeme 5
2.1. Statecharts als Entwicklungstechnik 5
2.2. Statechart Dialekte 8

3. Einführung der OCL 15
3.1. Constraint-Sprachen 15
3.2. Anwendungsgebiete der OCL 19
3.3. Werkzeugunterstützung 20

4. Style-Checking von Statecharts 25
4.1. Taxonomie des Style-Checking von Statecharts 25
4.2. Ein Style-Guide für Statecharts 27
4.3. Style-Checker 33

5. Implementierung 35
5.1. KIEL 35
5.2. Verfahren für das Style-Checking 36
5.3. Style-Checking in KIEL 37
5.3.1. OCL-Toolkits 39
5.3.2. Verwendung des Dresden OCL Toolkits 42
5.3.3. Arbeitsweise des Style-Checkers 46
5.4. XMI-Import 48

6. Ergebnisse 51
6.1. KOCL Usability-Analyse 51
6.2. Laufzeitanalyse 53
6.3. Ein reales Beispiel 56

7. Zusammenfassung und Ausblick **61**

A. Literaturverzeichnis **63**

B. Regelprofile **71**

C. Die EBNF von KOCL **75**

D. Das Metamodell **77**

E. Parserspezifikationen im SableCC Format **79**
E.1. KOCL-Grammatik 79
E.2. JavaTransitionLabels-Grammatik 80

F. KOCL-Spezifikationen **87**
F.1. *Well-formedness*-Regeln 87
F.1.1. UML 1.3 87
F.1.2. UML 2.0 90
F.2. Robustheitsregeln 92

G. Java Code **95**
G.1. XMI-Konverter 96
G.1.1. XMIConverter.java 96
G.1.2. XMISaxConverter.java 98
G.2. Regel-Generator 102
G.2.1. RuleParser.java 102
G.2.2. RuleSemanticAnalyzer.java 104
G.2.3. Translator.java 105
G.3. Checking-Plug-In 112
G.3.1. StateChartCheckerBase.java 112
G.3.2. BaseCheck.java 119
G.3.3. CheckerOutputGUI.java 121
G.3.4. CheckingProblem.java 122
G.3.5. CheckingProperties.java 123
G.3.6. MyCellRenderer.java 127
G.3.7. IStateChartVisitor.java 128
G.3.8. StateChartDepthFirstAdapter.java 129
G.4. XMI-Fileinterface 130
G.4.1. XMI.java 130
G.4.2. XMIFileFilter.java 132
G.4.3. ArgoUMLReader.java 133
G.4.4. KielUMLAttribute.java 146
G.4.5. KielUMLDataType.java 147
G.4.6. LabelGenerator.java 149
G.4.7. XMIGenerator.java 153

Tabellenverzeichnis

2.1. Übersicht ausgewählter Statechart-Elemente und deren Bedeutung . . 9

3.1. Spezifikationssprachen und deren zugehörige Constraint-Sprachen . . 18
3.2. Modellierungswerkzeuge mit OCL-Unterstützung 21

5.1. Anforderungen an untersuchte OCL-Toolkits 41

6.1. Umgesetzte *Well-formedness*-Regeln 51
6.2. Umgesetzte Regeln syntaktische Robustheit 53
6.3. Laufzeitergebnisse 54

Abbildungsverzeichnis

1.1. Fehlervermeidung in der Softwareentwicklung 1

2.1. Grundelemente von Statecharts 7
2.2. Ein Statechart in UML-Notation 10
2.3. Ein Statechart in Safe State Machine Notation 11
2.4. Ein Statechart in Stateflow Notation 12

3.1. In *Z*-Notation spezifiziertes Geburtstagsverzeichnis 16
3.2. Klassifikation der Notation von Constraint-Sprachen 18
3.3. Beziehungen von Sprachen, die zur OCL geführt haben 19
3.4. Constraints im Softwareentwicklungsprozess 19

4.1. Statechart Style Checking Taxonomie 26
4.2. Beispiel der *Well-formedness*-Regel *CompositeState* Nummer 1 28
4.3. Beispiel für die Regel *ORStateCount* 29
4.4. Beispiel für die Regel *DefaultFromJunction* 30
4.5. Beispiel für die Regel *InterlevelTransition* 31
4.6. Beispiel für die Regel *Connectivity* 32

5.1. Die Komponenten von KIEL nach dem MVC-Konzept geordnet . . . 36
5.2. Darstellung der durchgeführten Arbeitsschritte zur Übersetzung von OCL in Java-Code 38
5.3. Illustration der Beziehung zwischen dem Checking-Plug-In und den unterschiedlichen Regelmengen 39
5.4. Übersicht über die Funktionsweise des XMI-Konverters 43
5.5. An der OCL-Transformation beteiligte Module des *Dresden OCL Toolkit*s 44
5.6. Screenshot von KIEL mit ausgeklappten Checking-Menüpunkt 47
5.7. Das *Fileinterface*-Modul 48
5.8. Visualisierung der Zwischenschritte des implementierten Reader-Moduls 49
5.9. Arbeitsweise des XMI-Exports 50

6.1. Laufzeiten des Style-Checkers 55
6.2. Ein transformiertes Aktivitätsdiagramm 57
6.3. Verletztes Constraint der bmiag 58
6.4. Verletzte *Well-formedness*-Regel *PseudoState 1* 58
6.5. Überlappende Transitionen 59

C.1. Die KOCL-Grammatik in EBNF. 76

D.1. Das vereinfachte Metamodell der topologischen Statechart Datenstruktur. 78

G.1. Übersicht über die Abhängigkeiten des Checking-Plug-Ins. 95

Verzeichnis der Auflistungen

3.1. OCL-Ausdruck für die Bedingung an die AddBirthday-Methode . . . 18
5.1. Kommandozeilenausgabe des XMI-Konverters 43
5.2. OCL-Ausdruck der *Well-formedness*-Regel CompositeState-1 44
5.3. Erzeugter Java-Code 44
5.4. Ein KOCL-Beispiel für das KIEL-Metamodell 45
5.5. Kommandozeilenausgabe des *RuleParser* 46

Liste der Abkürzungen

CASE *Computer Aided Software Engineering*

DOM *Document Object Model*

KIEL *Kiel Integrated Environment for Layout*

KOCL *KIEL wrapped OCL*

MAAB *MathWorks Automotive Advisory Board*

MISRA *Motor Industry Software Reliability Association*

OCL *Object Constraint Language*

OMG *Object Management Group*

OMT *Object Modelling Technique*

PRG *Programming Research Group*

SAX *Simple API for XMI*

UML *Unified Modelling Language*

XMI *XML Metadata Interchange*

XML *Extended Markup Language*

1. Einleitung

Eingebettete Systeme sind Computer, die in größere Umgebungen integriert sind und häufig nicht als solche wahrgenommen werden. Das immer größer werdende Anwendungsgebiet von Computern führt insbesondere durch die eingebetteten Systeme zu einer Vielzahl von Anwendungen, bei denen ein Ausfall der Software schwere Folgen für Mensch und Umwelt nach sich ziehen kann. Längst übersteigen die Kosten, die für das Testen neu entwickelter Systeme und die später anfallenden Wartungsarbeiten aufgewendet werden, die reinen Entwicklungskosten um ein Vielfaches. Dementsprechend wird versucht, die Fehlervermeidung in möglichst großem Umfang zu automatisieren und damit die Kosten zu senken, denn von Programmen detektierte Fehler verursachen weniger Kosten als vom Menschen detektierte Fehler [93].

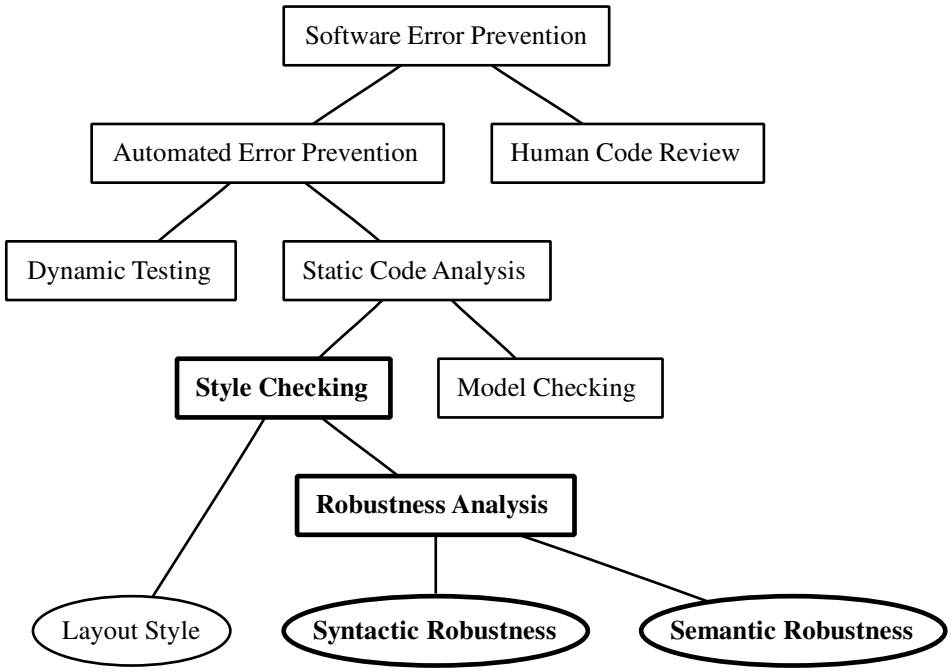


Abbildung 1.1.: Fehlervermeidung in der Softwareentwicklung (Quelle: [85])

Die Arten der Fehlervermeidung lassen sich, wie in Abbildung 1.1 angegeben, grundsätzlich in zwei Bereiche aufteilen. Auf der einen Seite die manuellen Code-Reviews (*human code review*). Das Hauptaugenmerk sollte in diesem Bereich dem Zusammenhang der einzelnen Funktionen in einem Programm gelten, da sich dieser meist nicht automatisch überprüfen lässt. Die Qualität der unter diesem Aspekt

durchgeführten Arbeiten basiert auf der Erfahrung der Entwickler, der noch vorhandenen Zeit bis zum Auslieferungstermin und vor allem von der Qualität des Quellcodes ab. Denn wie unter anderem Parnas [72] beobachtet hat, lassen sich Entwickler schnell durch Programm unabhängige Probleme von der eigentlichen Problemstellung ablenken.

Die automatisierte Fehlervermeidung (*automated error prevention*), auf der anderen Seite, versucht daher Programm-unabhängige Fehler vor den manuellen Code-Reviews mit verschiedenen Mitteln zu beseitigen, oder zumindest zu lokalisieren, falls eine automatische Beseitigung des Problems nicht möglich ist. Ein Teilgebiet der automatisierten Fehlervermeidung ist, wie dargestellt, der Vorgang des *Style Checking*s. Im Rahmen des Style-Checkings werden Regeln überprüft, die in speziellen Dokumenten gesammelt wurden. Diese Regelsammlungen heißen *Style-Guides*. Ein Teil der in den Style-Guides gesammelten Regeln behandelt das Layout von Quellcode, ein anderer Teil gibt Kriterien für robusten Quellcode an. Diese robusten Regeln schränken eine Programmiersprache auf eine sichere Teilmenge (*safe subset*) des möglichen Gesamtumfangs ein von der man annimmt, dass sie weniger fehlerträchtig ist [60, 84]. Style-Guides haben zwei Hauptanwendungsgebiete. Zum Einen dienen sie als Richtlinie für Entwickler. Zum Anderen können sie als Konfiguration für spezielle Programme verwendet werden, die automatisiert überprüfen, ob die Regeln eingehalten werden. Diese speziellen Programme heißen *Style-Checker*.

Es hat sich gezeigt, dass es wichtig ist, in der Softwareentwicklung konsistent zu einem Style-Guide zu arbeiten, da durch die Anwendung von Style-Guides die Wartbarkeit von Software stark verbessert wird. Und da etwa 80% der gesamten Entwicklungskosten einer Software in Wartungsarbeiten investiert wird [88], ist die Einhaltung von Style-Guides von entsprechend großer Bedeutung.

Eingebettete Systeme werden mit graphischen Sprachen entwickelt. *Statecharts* [34] haben sich zur Entwicklung eingebetteter Systeme bewährt. Statecharts sind ein visueller Formalismus mit denen das Verhalten von eingebetteten Systemen beschrieben werden kann. Style-Guides für klassische, d.h. textuelle, Programmiersprachen lassen sich jedoch nicht auf Statecharts übertragen. Daher wurden auch für die modellbasierte Entwicklung verschiedene Style-Guides entwickelt.

Im Rahmen dieser Arbeit wird der von Schaefer [85] vorgestellte Statechart Style-Guide aufgegriffen und erweitert. Zur automatischen Überprüfung der im entwickelten Style-Guide enthaltenen Regeln wurde ein Style-Checker als Plug-In in das Modellierungswerkzeug *Kiel Integrated Environment for Layout* (KIEL) [43] integriert. KIEL ist ein prototypisches Modellierungswerkzeug, das am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme an der Christian-Albrechts-Universität zu Kiel entwickelt wird.

Die Konfiguration des entwickelten Style-Checkers erfolgt in dieser Arbeit mit Regeln, die mit der *Object Constraint Language* (OCL) [92] formuliert werden. Die OCL wurde gewählt, da diese keine Kenntnisse einer Programmiersprache voraussetzt [63] und eine leicht verständliche Syntax besitzt. Die Auswertung von OCL kann mittels eines *interpretativen* oder eines *transformativen* Ansatzes durchgeführt werden. Der *interpretative* Ansatz verwendet ein *Interpreter* genanntes Programm,

das den OCL-Code zur Laufzeit einliest, analysiert und anschließend die entsprechenden Überprüfungen vornimmt. Beim *transformativen* Ansatz wird der OCL-Code in eine Programmiersprache übersetzt und anschließend wird dieser Code zur Ausführung compiliert. In dieser Arbeit wird eine transformative Auswertung von OCL-Code vorgestellt.

Die Beiträge dieser Diplomarbeit zur Fehlervermeidung in der Softwareentwicklung mit Statecharts sind: (1) Die Erweiterung eines generellen Style-Guides für Statecharts um syntaktische Korrektheit und syntaktische Robustheit betreffende Regeln. (2) Die Entwicklung eines dialektunabhängigen Statechart Style-Checkers auf Basis einer transformativen OCL-Auswertung und (3) die Beurteilung der durchgeführten Analyse der Robustheit industrieller Statecharts.

Diese Arbeit ist wie folgt aufgebaut:

- Im nächsten Abschnitt dieses Kapitels werden verwandte Arbeiten zur Fehlervermeidung in der modellbasierten Softwareentwicklung und deren Defizite betrachtet, die zu der Entwicklung des in dieser Arbeit vorgestellten Statechart Style-Checkers geführt haben.

- In Kapitel 2 werden Statecharts und verschiedene Dialekte vorgestellt.

- Im darauffolgenden Kapitel werden Constraint-Sprachen und die Entstehung der OCL vorgestellt. Des Weiteren werden Einsatzgebiete und Werkzeugunterstützung untersucht.

- In Kapitel 4 wird eine Taxonomie zur Fehlervermeidung auf Statecharts vorgestellt. Anschließend wird der im Rahmen dieser Arbeit erweiterte Style-Guide zur Fehlervermeidung in der modellbasierten Entwicklung vorgestellt. Basierend auf der eingeführten Taxonomie werden verschiedene Modellierungswerkzeuge und Style-Checker untersucht und die Entwicklung eines Style-Checkers motiviert.

- Kapitel 5 betrachtet als erstes verschiedene Techniken, die zur Auswertung von Regeln auf Statecharts angewendet werden können. Anschließend werden die Ergebnisse einer Untersuchung verschiedener Toolkits zur Arbeit mit OCL vorgestellt. Die Integration des gewählten *Dresden OCL Toolkit*s [19] in KIEL und welche Arbeiten durchgeführt wurden, um ein flexibles Checking-Plug-In zu entwickeln, werden anschließend beschrieben. Damit die Regeln des hier entwickelten Style-Guides auch auf Statecharts aus der *Unified Modelling Language* (UML) angewendet werden können, wird abschließend der entwickelte *XML Metadata Interchange* (XMI)-Importer vorgestellt.

- Im Kapitel 6 werden die Ergebnisse einer Usability-Analyse des entwickelten Style-Checker vorgestellt.

- Im abschließenden Kapitel wird eine Zusammenfassung der Arbeit und ein Ausblick auf offene Aufgaben gegeben.

Die in dieser und der verwandten Diplomarbeit [85] beschriebenen Ergebnisse sind im Oktober 2006 in Genua auf einem Workshop mit dem Titel *Modeling and Analysis of Real-Time and Embedded Systems (MARTES' 06)* vorgestellt worden [75].

1.1. Verwandte Arbeiten

Wie einleitend erwähnt, enthalten Style-Guides Regeln zur Vermeidung fehleranfälliger Konstruktionen. Style-Guides für die modellbasierte Entwicklung mit Statecharts wurden unter anderem von der Ford Motor Company [28] und dem Mathworks Automotive Advisory Board (MAAB) [54] entwickelt. Diese beiden Style-Guides sind auf den Statechart-Dialekt *Matlab Simulink/Stateflow* beschränkt. Scaife *et al.* [84] haben außerdem eine sichere Teilmenge der *Stateflow*-Sprache vorgestellt. Die vorgestellten Style-Guides sind dialektspezifisch und eignen sich daher nicht für die in dieser Arbeit angestrebte dialektunabhängige Analyse der Robustheit von Statecharts.

Für UML *StateMachine*s erklären die in der UML-Spezifikation [66] enthaltenen *Well-formedness*-Regeln die Bedeutung der Statechart-Elemente. Darüberhinaus wurden weitere Regeln für UML *StateMachine*s unter anderem von Mutz [63] aufgestellt. In den in dieser Arbeit vorgestellten dialektunabhängigen Style-Guide für Statecharts wurden geeignete, d. h. dialektunabhängige, Regeln der vorgestellten Style-Guides und aus der eigenen Erfahrung stammende Regeln aufgenommen.

Die Überprüfung der Regeln eines Style-Guides kann, wie erwähnt, mit Hilfe eines Style-Checkers automatisiert durchgeführt werden. Die Modellierungswerkzeuge *Esterel Studio* und *Matlab Simulink/Stateflow* beinhalten bereits Funktionen zur statischen Analyse. Diese sind jedoch nicht explizit aufrufbar und die mitgelieferten Analysen lassen sich weder erweitern, noch an eigene Bedürfnisse anpassen. Ein weiterer Nachteil der mitgelieferten Analysen ist, dass sie entweder sehr einfache oder sehr anspruchsvolle Analysen von der Art eines *Model Checking*s bieten [33, 46, 64]. Die in dieser Arbeit betrachtete Analyse der syntaktischen Robustheit von Statecharts lässt sich nicht durchführen.

Neben den in die Modellierungswerkzeuge integrierten Analysefunktionen wurden kommerzielle und nicht-kommerzielle Style-Checker zur Analyse entwickelt. Die Werkzeuge *Mint* [81], *State Analyzer* [32, 46] und *Guideline Checker* [61] sind speziell für den Dialekt Stateflow entwickelt worden und eigenen sich daher ebenfalls nicht für die in dieser Arbeit angestrebte dialektunabhängige Analyse der syntaktischen Robustheit von Statecharts.

Der dialektunabhängige *Regel Checker* [64] bietet die Möglichkeit, die Regelmenge flexibel mit Java-Klassen und OCL-Ausdrücken zu erweitern und mitgelieferte Regeln an eigene Bedürfnisse anzupassen. Die Auswertung von OCL-Ausdrücken erfolgt im *Regeln Checker* in einem interpretativen Ansatz. Dieses Vorgehen zur Auswertung schließt die vorgestellten Nachteile einer interpretativen Auswertung mit ein. Ein weiterer Nachteil ist, dass mit dem *Regel Checker* keine Analysen der semantischen Robustheit auf der Basis eines Theorembeweisers möglich ist.

2. Entwicklung eingebetteter, reaktiver Systeme

Ein *reaktives System* ist nach Harel ein System, das ereignisgesteuert arbeitet und fortwährend auf innere und äußere Einflüsse reagiert [34]. Beispiele für reaktive Systeme sind der Thermostat einer Klimaanlage oder die Airbagsteuerung. Zur Steuerung der meisten reaktiven Systeme werden eingebettete Systeme verwendet. Im Fall der Airbagsteuerung wird im besonderen Maße die sicherheitskritische Bedeutung solcher Systeme bewusst. Da die hergestellten reaktiven Systeme jedoch immer komplexer werden, hat sich relativ bald herausgestellt, dass die vormals häufig zur visuellen Beschreibung eines Systems verwendeten Zustandsdiagramme den gesteigerten Anforderungen an Komplexität, Simulierbarkeit und Sicherheit nicht gewachsen waren. Daher wurde lange Zeit nach einem Weg gesucht, wie sich auch komplexe reaktive Systeme präzise, formal und vor allem intuitiv beschreiben lassen und dabei gleichzeitig von einem Computer simuliert werden können. Insbesondere die Simulation von reaktiven Systemen im sicherheitskritischen Umfeld muss zuverlässig durchführbar sein, um schon während der Entwicklung möglichst alle Fehler auszuschließen. Schließlich könnten bei einem Ausfall eines reaktiven Systems Menschenleben gefährdet sein. Beispiele für den Ausfall eingebetteter Systeme sind die Explosion einer Rakete vom Typ Ariane [52] (1996), das Unglück des Lufthansa-Flugs DLH-2904 bei der Landung in Warschau [50] (1993) und die Fehlfunktion des Steuerungssystems von Raketen des Typs Patriot im Golfkrieg [10] (1991).

1987 hat David Harel eine Forschungsarbeit [34] veröffentlicht, in der die Probleme, die bei der Verwendung von Zustandsdiagrammen in der Softwaremodellierung auftreten, analysiert werden. Basierend auf den Ergebnissen hat Harel die endlichen Automaten erweitert und die Statecharts vorgestellt. Im folgenden Abschnitt wird die Entwicklung der Statecharts vorgestellt. Anschließend werden verschiedene, für diese Arbeit relevante, Statechart-Dialekte betrachtet.

2.1. Statecharts als Entwicklungstechnik

In den 1950er Jahren haben Rabin und Scott [78], basierend auf den Erfahrungen mit Turingmaschinen und beeinflusst von Kleene's Arbeit [44] zu McCulloch und Pitts *nerve-nets* [56], die endlichen Automaten (*Finite State Machines*) zur Beschreibung von Computersystemen vorgestellt. Endliche Automaten zeichnen sich dadurch aus, dass die benötigte Menge von Arbeitsmaterial, im Gegensatz zur Verwendung von Turingmaschinen, endlich ist, und dass sie dabei gleichzeitig vollkommen ausreichend

zur Formalisierung von Computern sind. Da sich ein Computer zu jedem diskreten Zeitpunkt immer in genau einem wohldefinierten Zustand befindet, und Transitionen (Zustandsübergänge) durch eine Eingabe ausgelöst werden, sind Zustände und Transitionen die einzigen beiden benötigten Elemente, um einen Computer essentiell zu beschreiben [78, S. 116].

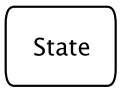
Endliche Automaten können durch gerichtete Graphen visualisiert werden. Die Knoten des Graphen repräsentieren die Zustände des Automaten, die gerichteten Kanten die Transitionen, und Kantenbeschriftungen die Bedingung, unter welcher Eingabe die Transition ausgelöst wird, repräsentieren. Gerichtete Graphen, die zur Repräsentation von Endlichen Automaten verwendet werden, werden als *Zustands-Übergangs-Diagramm* oder kurz *Zustandsdiagramm* (*state-diagramm*) bezeichnet.

Endliche Automaten nach Rabin und Scott berechnen bool'sche Funktionen und liefern als Ergebnis *yes* oder *no* [78]; weitere Ausgaben etwa zur Interaktion mit der Umgebung werden nicht erzeugt. Dies ist jedoch für die Modellierung von reaktiven Systemen notwendig, da diese relevante Ereignisse von der Umgebung als Eingabe erhalten und entsprechende Ausgaben erzeugen. Unter anderem Moore und Mealy erweitern daher das Konzept der endlichen Automaten um Techniken zur Erzeugung von Ausgaben. Moore [59] verknüpft Ausgaben mit Zuständen, Mealy [57] hingegen verknüpft die Ausgabe mit Transitionen.

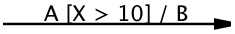
Die endlichen Automaten nach Rabin und Scott weisen einige gravierende Schwächen auf. Aufgrund dieser ist ihr Einsatz zur Visualisierung von komplexen Systemen mit enormen Aufwand verbunden und ab einer gewissen Größe des Automaten im Grunde nicht mehr vom Menschen durchführbar. Eine Ursache ist die exponentiell wachsende Anzahl von Zuständen – bedingt durch fehlende Methoden zur Strukturierung – die bei der Modellierung von komplexen (reaktiven) System benötigt werden. Dieses Problem wurde bereits 1969 von Parnas beschrieben [71].

Als Erweiterung der endlichen Automaten und deren visueller Repräsentation durch Zustandsdiagramme führt Harel die *Statecharts* [34] ein, um mit ihnen einige der Probleme zu beseitigen. Es ist besonders interessant, dass die Autoren des Artikels schreiben: „. . . we strongly believe in the virtues of visual descriptions“ [34, S. 233]. Daher werden die Statecharts nur in graphischer Notation eingeführt.

Harel analysiert die Probleme, die bei der Verwendung von Zustandsdiagramme auftreten, und führt Lösungen für diese ein. Er stellt unter anderem fest, dass ein Problem klassischer Zustandsdiagramme die unvollständige Nutzung der Fläche der Zeichnung ist. Grund hierfür ist, dass sie nur aus Knoten, in Form von Punkten oder Kreisen, für die Zustände und gerichteten Kanten, in Form von Linien, für die Transitionen bestehen. Um den Platz besser ausnutzen zu können, führt Harel Higraphen ein [35]. In diesen werden für die visuelle Repräsentation von Zuständen Rechtecke mit abgerundeten Ecken und optionaler Beschriftung wie in Abb. 2.1a verwendet. Die so dargestellten Zustände nutzen den Platz aus und bieten gleichzeitig die Möglichkeit weitere Zustände in ihrem Inneren anzuordnen (*depth*), so dass sich verschiedene Hierarchie-Ebenen modellieren lassen. Hierarchische Zustände, die weitere Zustände enthalten, nennt Harel „OR“-Zustände. Des Weiteren greift Harel Moores Konzept auf und sieht vor, das Ausgaben von Zuständen erzeugt werden



(a) Ein einfacher Zustand.



(b) Transition mit Beschriftung.

Abbildung 2.1.: Grundelemente von Statecharts

können. Diese Ausgabe erfolgt mit Aktionen. Harel definiert *Entry*-, *During*- und *Exit*-Aktionen vor, die beim Betreten, beim Verweilen und beim Verlassen eines Zustands ausgelöst werden.

Die Transitionen erweitert Harel um die Möglichkeit, wie bei Mealy, Aktionen in die Beschriftung einzufügen. In Abb. 2.1b ist eine Transition mit einer Beschriftung im Format *Trigger [Bedingung] /Aktion* angegeben. Ein *Trigger* ist ein von außen, das heißt von der Umgebung, in der das System eingebettet ist, in das Statechart eingegebenes oder durch das Statechart generiertes Signal. Ist das im *Trigger* angegebene Signal vorhanden wird die Transition gefeuert und die *Aktion* ausgelöst. Ein Beispiel für eine Aktion kann etwa die Generierung eines solchen Signals sein. Mit einer optionalen *Bedingung* können weitere Einschränkungen an eine Transition gemacht werden. Die abgebildete Transition darf nur genommen werden, wenn das Signal `A` anwesend ist und die Variable `X` größer zehn ist.

Die bisher noch nicht berücksichtigte Parallelität (*orthogonality*) führt Harel mit einer einfachen Technik ein. Für zwei parallele, voneinander unabhängige Bereiche eines Systems wird ein OR-Zustand durch eine gestrichelte Linie geteilt und die parallelen Bereiche des modellierten Systems werden in den so entstandenen Flächen angeordnet. Diese so konstruierten parallelen Zustände nennt Harel „AND"-Zustände [34, S. 242].

Zusätzlich definiert Harel, dass es innerhalb des Statecharts Datenaustausch über einen Broadcasting-Mechanismus gibt. Der Datenaustausch erfolgt mit Hilfe von Signalen und Variablen. Diese Daten sollten in der Statechart Definition berücksichtigt werden. Harels ursprüngliche Statechart Definition [34, Seite 233] wurde daher wie folgt erweitert [36]:

> Statecharts = state-diagrams + depth + orthogonality + broadcast-communication + data.

Harels Vorschläge wurden von Softwareentwicklern aufgenommen und haben in verschiedenen Statechart-Dialekten Anwendung gefunden, da die originale Statechart-Definition in verschiedenen Punkten unterschiedliche Interpretation zulässt. Eine detaillierte Untersuchung von 21 verschiedenen Dialekten und deren Unterschiede hat von der Beeck bereits 1994 veröffentlicht [8]. Die in dieser Arbeit relevanten Dialekte werden im folgenden Abschnitt vorgestellt.

2.2. Statechart Dialekte

Die verschiedenen Dialekte, in welchen Harels Statecharts angewandt wurden, weisen zwei Hauptunterschiede auf: Zum Einen mögliche Unterschiede in der Syntax und zum Anderen unterschiedliche Ausführungsmodelle.

Die Syntaxen selbst weisen Unterschiede sowohl in der graphischen Repräsentation, als auch in der Menge der zur Verfügung stehenden Statechart-Elemente auf. Tabelle 2.1 listet einige verschiedene zur Verfügung stehende Elemente und deren Bedeutung auf. Unterschiede in der graphischen Ausprägung der einzelnen Elemente können den abgebildeten Beispielen der nachfolgend betrachteten Statechart-Dialekte entnommen werden.

Die Ausführungsmodelle lassen sich, basierend auf dem zugrundeliegenden Zeitmodell, in zwei verschiedene Kategorien unterteilen: Zum Einen das *synchrone* und zum Anderen das *asynchrone* Zeitmodell [8]. Die im Rahmen dieser Arbeit betrachteten Dialekte und einige Werkzeuge, in welchen diese umgesetzt sind, werden im Folgenden vorgestellt.

*StateMachine*s

Die UML [66, 67, 90] als graphische Spezifikationssprache enthält Statecharts zur Spezifikation des Verhaltens modellierter Elemente. Beispielsweise kann mit ihnen die Interaktion zwischen verschiedenen Klassen modelliert werden. Die graphische Repräsentation einiger Statechart-Elemente ist in Abbildung 2.2 enthalten. Statecharts werden in der UML *StateMachine*s genannt.

Die Semantik von *StateMachine*s ist in textueller Notation in UML-Spezifikationen enthalten. Eine Referenzimplementierung zur Simulation existiert jedoch nicht, so dass Entwickler von *Computer Aided Software Engineering* (CASE)-Tool oftmals damit werben, dass ihr entwickeltes Produkt „UML-konform“ ist. Dennoch gibt es unterschiedliche Ansätze, um Unklarheiten aus der Spezifikation der *Object Management Group* (OMG) zu beseitigen und *StateMachine*s simulieren zu können.

Grundsätzlich unterscheidet sich der Einsatzzweck von *StateMachine*s und der dafür angebotene Funktionsumfang von Programm zu Programm zum Teil jedoch enorm. Ein frei verfügbares CASE-Programm ist *ArgoUML* [4]. Dieses Programm bietet die Möglichkeit, mit der UML Software-Systeme ausgehend von Klassendiagrammen zu spezifizieren. Die *StateMachine*s dienen in diesem Programm – wie von der OMG vorgesehen – dazu, Verhalten und Interaktion von modellierten Komponenten zu visualisieren. Das Werkzeug bietet jedoch keinerlei Möglichkeit zur Simulation der *StateMachine*s. Der Entwickler ist also auf seine eigenen Kenntnisse und ausgiebiges Testen der Software angewiesen, um eventuelle Fehler zu lokalisieren. Des Weiteren liefert das Werkzeug bei der Modellierung von *StateMachine*s nur Hinweise, wenn etwas im Widerspruch zur *StateMachine*-Spezifikation modelliert wurde. Syntaktische Fehler, die im Grunde leicht zu vermeiden sind, werden nicht ausgeschlossen. Beispielsweise ist es möglich, beliebig viele initiale Zustände

Tabelle 2.1.: Übersicht ausgewählter Statechart-Elemente und deren Bedeutung

Element	Bedeutung
Zustand	Graphisches Element zur Illustration diskreter Zustände eines modellierten Systems (siehe Abbildung 2.1a).
Transition	Grundlegendes Element zur Modellierung von (bedingten) Zustandsübergängen. *Transitionen* können Beschriftungen der Form *Trigger [Bedingung] / Aktion* tragen. Eine *Transition* wird nur ausgelöst, wenn das triggernde Signal präsent und die Bedingung erfüllt ist. Im Dialekt Safe State Machine optionale Prioritäten bestimmen die Reihenfolge, in der Transitionen genommen werden.
Root-Zustand	In der topologischen Sicht der oberste Zustand eines Statecharts. Alle weiteren Zustände sind in diesem enthalten. In der UML *Top-Zustand* genannt.
Pseudozustand	Eine Klasse von Zuständen, in denen ein modelliertes System nicht verweilt. Als *Transition* wird die Sequenz bestehend aus einer hinein- und einer hinausgehenden *Transition* bezeichnet. Solche *Transitionen* werden auch zusammengesetzt (*compound*) genannt. Teilstücke heißen *Segment*.
OR-Zustand	Ein hierarchischer Zustand, der weitere Zustände enthält. Ist ein enthaltener Zustand aktiv, ist auch der *OR-Zustand* aktiv.
AND-Zustand	Ein hierarchischer, paralleler Zustand. Enthält Regionen, die sich wie *OR-Zustände* verhalten. Alle enthaltenen parallelen Regionen sind gleichzeitig aktiv.
Initialer Zustand	Ein *Pseudozustand*. In solch einem Zustand wird die Ausführung eines Statechart, beziehungsweise eines hierarchischen Zustand begonnen.
Finaler Zustand	Wird ein solcher Zustand erreicht, ist die Ausführung des Statechart, beziehungsweise des ihn enthalteden Zustand beendet. Im Dialekt SSM können sogar Hierarchische Zustände als Final gekennzeichnet werden.
Shallow History	Ein *Pseudozustand*. Wird mit einem hierarchischen Zustand verwendet. Wird der enthaltende hierarchische Zustand verlassen, speichert der *Shallow History* Zustand, in welchem enthaltenen Zustand das System beim Verlassen war. Beim erneuten Betreten des enthaltenden Zustands wird der zuletzt aktive Zustand wieder hergestellt. Dieses Verhalten ist beschränkt auf die Hierarchiebene, in der ein *Shallow History* Zustand enthalten ist.
Deep History	Ein *Pseudozustand*. Die Variante *Deep History* stellt das Verhalten des *Shallow History* Zustands über alle weiteren Hierarchieebenen zur Verfügung. Dieser Zustand ist eine Besonderheit des Statechart Dialekts der UML.
Choice (oder Conditional oder Connective junction)	Ein *Pseudozustand*. Ein Choice-Zustand dient der bedingten Strukturierung des Kontrollflusses.

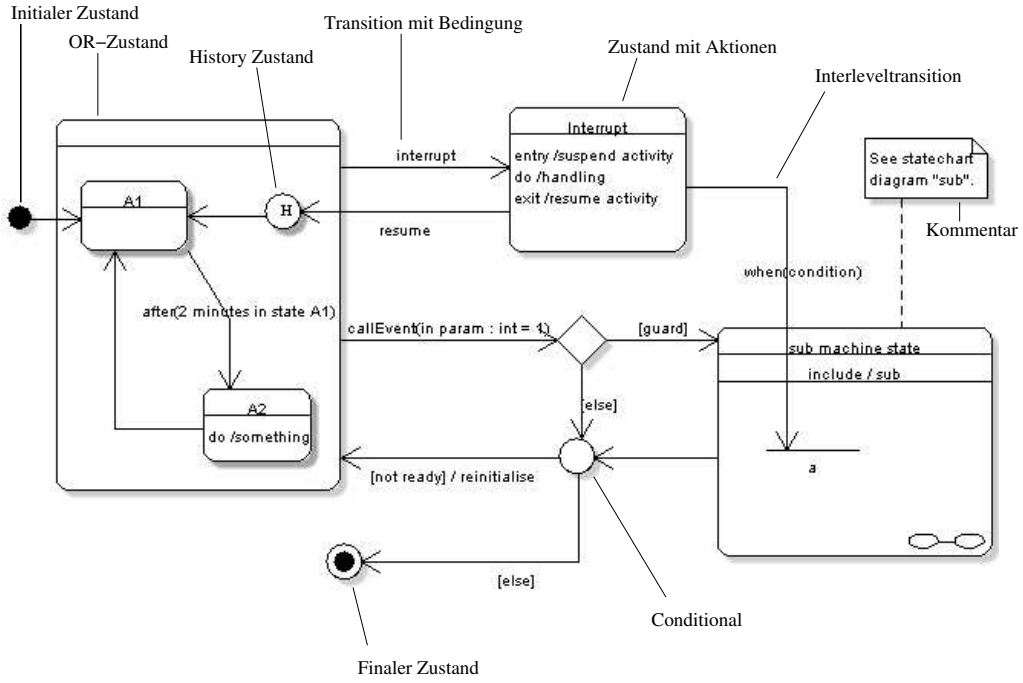


Abbildung 2.2.: Ein Statechart in UML-Notation (Quelle: [4])

in einem OR-Zustand zu verwenden. Die OMG spezifiziert jedoch, dass es in einem OR-Zustand nur maximal einen initialen Zustand geben darf [66, Kapitel 2.12.3.1]. Auch die in *ArgoUML* enthaltene Funktion zur Codegenerierung bietet keinen großen Funktionsumfang. Aus den modellierten Klassen werden Quellcode-Dateien für Java erzeugt. Die erzeugten Klassendateien enthalten jedoch nur die Attribute und Methodenrümpfe. Verhalten, das mit *StateMachine*s spezifiziert wurde, wird nicht berücksichtigt.

Neben *ArgoUML* gibt es andere Werkzeuge, wie die *Rational*-Werkzeugfamilie [79] von IBM, *Rhapsody* [80] von I-Logix und *ARTiSAN Studio* [5], die auch die UML zur Spezifikation verwenden. Im Gegensatz zu *ArgoUML* ist in letzteren eine Semantik für *StateMachine*s implementiert, mit der Simulationen des Modells durchgeführt werden können. Es ist also möglich, Fehler im Programm schon während der Spezifikation aufzudecken. *Rhapsody* und *ARTiSAN Studio* sind Entwicklungswerkzeuge, die von ihren Herstellern explizit zur Entwicklung von eingebetteten Echtzeitsystemen angepriesen werden. *ARTiSAN Studio* wird unter anderem in der Entwicklung von Systemen im Luftfahrtbereich, in der Automobilindustrie oder im medizinischen Umfeld eingesetzt [5].

Aber auch diese Entwicklungswerkzeuge weisen Schwächen in der Statechart-Analyse auf. Beispielsweise lassen sich keine syntaktische Analysen durchführen, die auf eventuelle Fehler im modellierten Statechart im Sinne der Robustheit hinweisen. Semantische Analysen sind häufig nicht vorhanden.

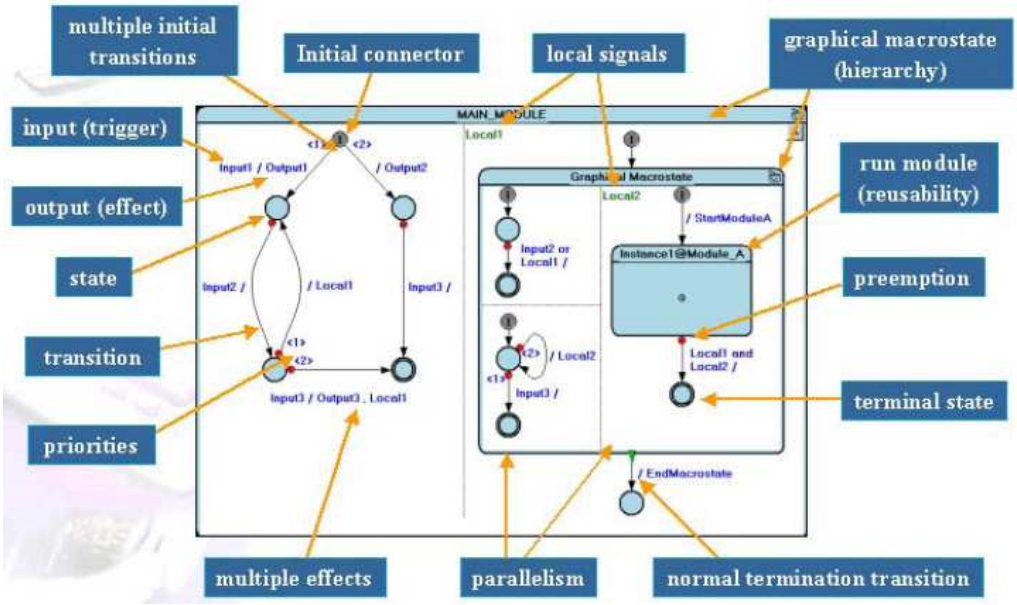


Abbildung 2.3.: Ein Statechart in Safe State Machine Notation (Quelle: [22])

Safe State Machines

Ein weiterer Statechart Dialekt ist *Safe State Machine* (SSM) [3]. Dieser Dialekt ist eine kommerzielle Erweiterung der von André entwickelten *SyncCharts* [2]. Der Dialekt SSM weist sowohl syntaktische als auch semantische Unterschiede zu Harels Statecharts auf; beispielsweise sind keine Interlevel-Transitionen erlaubt. Die graphische Repräsentation einiges SSM-Statecharts ist in Abbildung 2.3 angegeben. Den SSMs ist ein *synchrones* Zeitmodell mit der „Null-Zeit"-Annahme zugrunde gelegt [9]. Der Dialekt wurde im Werkzeug *Esterel Studio* umgesetzt. Mit diesem Werkzeug lässt sich aus modellierten Systemen unter anderem C- und C++-Code erzeugen, der für die Zielplattform des Systems übersetzt und anschließend direkt auf der Zielplattform ausgeführt werden kann. Aufgrund der Tatsache, dass der Dialekt, wie bereits im Namen enthalten, *safe* – also sicher – ist, existieren keine externen Analyse-Werkzeuge. Der syntaxgerichtete Editor von *Esterel Studio* verhindert durch integrierte Funktionen syntaktisch fehlerhafte Konstrukte, so z. B. Interlevel-Transitionen. Über die Syntax hinausgehende Analysen werden mit *Modellchecking*, welches im *Build*-Prozess durchgeführt wird, vorgenommen.

Stateflow

Statecharts im Stateflow-Dialekt sind in *Simulink*-Modelle in Mathworks *Matlab Simulink/Stateflow* Entwicklungswerkzeug eingebettet [55, 89]. Die graphische Repräsentation eines Stateflow-Statecharts ist in Abbildung 2.4 dargestellt. *Matlab Simulink/Stateflow* wird vor Allem in der Automobilindustrie verwendet. Ähnlich wie in *Esterel Studio* lässt sich auch in diesem Werkzeug eine Simulation des entwi-

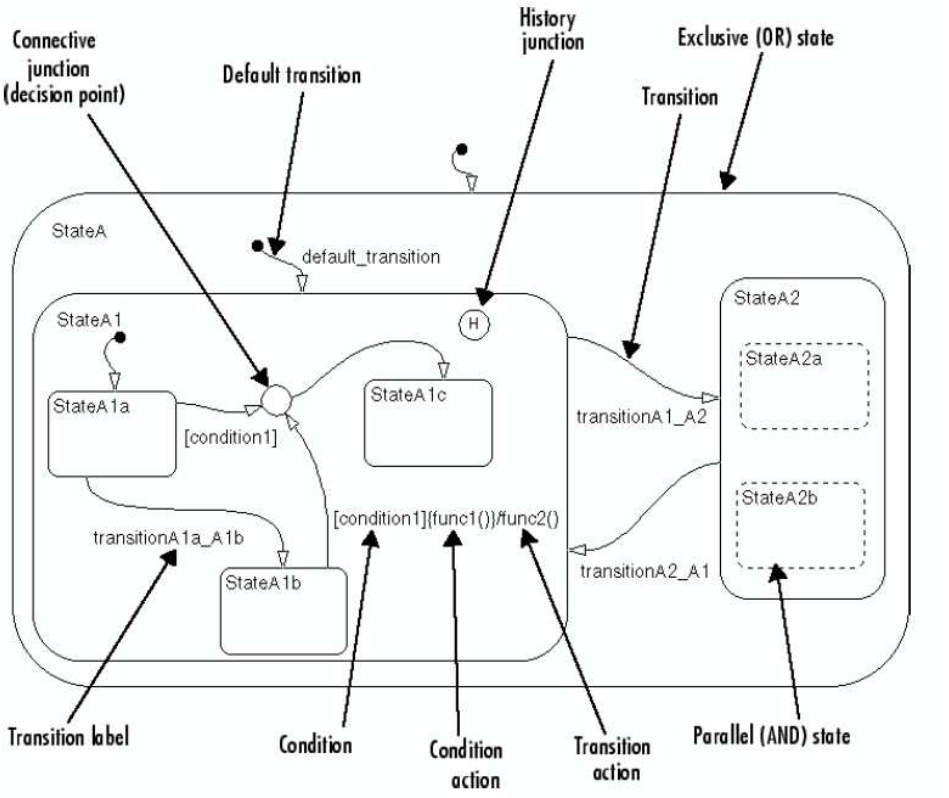


Abbildung 2.4.: Ein Statechart in Stateflow Notation (Quelle: [89])

ckelten Systems durchführen, um so Fehler frühzeitig zu erkennen. Die im Werkzeug enthaltenen Analysefunktionen sind auf statische Analysen beschränkt.

Diese werden mit *Matlab Simulink/Stateflow* entwickelt. Im Dialekt *Stateflow* sind, im Gegensatz zum Dialekt SSM, Interlevel-Transitionen erlaubt. Style-Guides raten jedoch von deren Verwendung ab [28, 54]. Weitere Probleme der Semantik sind unter Anderem die nicht intuitive Auswertungsstrategie, welche Transition als nächstes genommen wird, da es keine Möglichkeit gibt, explizit Prioritäten zu vergeben. Sind alle von einem Zustand ausgehende Transitionen unbeschriftet, basiert die Strategie auf einer Auswertung der Anordnung der Transitionen um den Zustand herum im Uhrzeigersinn von zwölf Uhr ausgehend. Diese Strategie stellt insbesondere für ungeübte Entwickler eine häufige Fehlerquelle dar, da die graphische Anordnung in der Modellierung oft unberücksichtigt bleibt. Darüberhinaus kann die Verwendung von graphischen Informationen in der Auswertungsstrategie beim Einsatz von Layout-Werkzeugen zu Problemen führen. Ändern solche Werkzeuge beim automatischen Layout die Anordnung von Transitionen, wird auch das Verhalten des Modells geändert.

Fazit

Statecharts haben sich als Technik zur Entwicklung von eingebetteten, reaktiven Systemen bewährt. Die unterschiedlichen der Ausführung zugrunde gelegten Zeitmodelle bieten die Möglichkeit, je nach Situation das bessere Modell zu wählen und sich die jeweiligen Stärken eines Modells zu Nutze zu machen.

Die UML, die als Industriestandard angesehen wird, weist einige Schwächen hinsichtlich der Semantik auf. Außerdem möchte sie als Sprache für viele Einsatzbereiche in der Softwarespezifikation verstanden werden [66] und ist daher nicht speziell auf die Entwicklung eingebetteter, reaktiver Systeme ausgerichtet. Wie vorgestellt, gibt es neben der allgemeinen Umsetzung von Statecharts in der UML spezielle Implementierungen von Statecharts zur Modellierung eingebetteter, reaktiver Systeme. Solche Implementierungen von Statecharts konzentrieren sich jedoch auf die Modellierung von eingebetteten, reaktiven Systemen mit Statecharts und deren Simulation. Im Gegensatz zur UML ist der Einsatzzweck also beschränkt.

Die visuellen Formalismen zur Spezifikation von Softwaresystemen – wie die hier betrachteten Statecharts – reichen jedoch nicht immer aus, um alle Anforderungen an eine Software zu formulieren. Bestimmte Sachverhalte, wie etwa vom modellierten System unabhängige Anforderungen an Statecharts selbst, können nicht oder nur sehr umständlich spezifiziert werden. Dazu gibt es Constraintsprachen wie die OCL. Diese wird von der OMG in der UML zur Formulierung von *Well-formedness*-Regeln verwendet, die in dieser Arbeit ebenfalls in der Analyse von Statecharts ausgewertet werden. Daher wird im folgenden Kapitel eine Einführung in die OCL gegeben.

3. Einführung der OCL

Spezifikationssprachen werden seit den 1970er Jahren eingesetzt, um die verschiedenen Anforderungen an eine Software formal zu formulieren. Einschränkungen, denen Systeme unterliegen, lassen sich mit den allgemeinen Spezifikationssprachen jedoch schlecht erfassen. Daher gibt es für einige Spezifikationssprachen eine Constraint-Sprache, mit der sich zusätzliche Einschränkungen formulieren lassen. Mitte der 1990er Jahre gab es einige miteinander konkurrierende Spezifikationssprachen, bis die OMG die UML veröffentlicht hat. Nach der Vorstellung der UML stellten die Anwender fest, dass auch die von der UML vereinigten objekt-orientierten Methoden der Softwaremodellierung alleine nicht ausreichen, um alle Facetten eines Systems zu erfassen [67]. Daher wurde die UML in Version 1.1 um die 1995 von IBM entwickelte *Object Constraint Language* (OCL) [92] erweitert. Mit der OCL können Einschränkungen an Objekte genauer spezifiziert werden, als es mit der graphischen Notation der UML möglich wäre.

Einschränkungen, die mit der OCL formuliert werden, heißen *Constraints*. Ein *Constraint*, auch *Assertion* genannt, ist in der Informatik bereits seit längerem bekannt. Sie sind ein Ausdruck zur Beschreibung des Zwecks eines Elements. Assertions werden in die folgenden drei Klassen eingeteilt: (1) Vorbedingungen, (2) Nachbedingungen und (3) Invarianten. Assertions werden bezüglich einer Klasse definiert. Sie kommen in der Sprache Eiffel [58] zum Einsatz und stellen dort die Basis für das dort umgesetzte *Design-by-Contract*-Prinzip dar. Für die in dieser Arbeit umgesetzte Robustheitsanalyse von Statecharts werden Invarianten bezüglich des Statechart Metamodells ausgewertet. Zur Formulierung dieser Invarianten wird die Constraint-Sprache OCL verwendet.

Welchen Ursprung die OCL hat, welche Einsatzmöglichkeiten es gibt, und wie sich die Unterstützung der OCL durch aktuelle Werkzeuge im Softwareentwicklungsprozess darstellt, wird im Folgenden aufzeigt.

3.1. Constraint-Sprachen

Der Software-Entwicklungsprozess beginnt immer mit dem Erfassen und Formulieren von Anforderungen, die die zu entwickelnde Software erfüllen muss. Werden die erfassten Anforderungen in einem umgangssprachlich formulierten Text festgehalten, kann es zu Mehrdeutigkeiten kommen, die wiederum zu Fehlern in der Software führen können. Der Fokus von Spezifikationssprachen liegt daher auf der formalen Beschreibung zu entwickelnder Systeme, ohne dabei Details der Implementierung zu betrachten. Bereits Ende der 1970er Jahre begann am Oxford University Computing

$[NAME, DATE]$

BirthdayBook

$known : \mathbb{P}\, NAME$

$birthday : NAME \pfun DATE$

$known = \mathrm{dom}\, birthday$

(a) Basistypen und Zustandsraum des Geburtstagskalenders in Schema-Schreibweise

AddBirthday

$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

(b) Bedingungen an die Methode zum Hinzufügen eines Geburtstags zum Verzeichniss

Abbildung 3.1.: In *Z*-Notation spezifiziertes Geburtstagsverzeichnis (Quelle: [86])

Laboratory die *Programming Research Group* (PRG) [68] mit Arbeiten an einer Sprache, die durch ihre formale Grundlage keinen Raum für Mehrdeutigkeiten in Softwarespezifikationen lassen sollte.

Die schließlich von der PRG entwickelte formale und typisierte Spezifikationssprache *Z* [14] basiert auf der Mengentheorie von Zermelo-Fraenkel und Prädikatenlogik erster Ordnung und wurde nicht entworfen, um ausführbar zu sein, sondern um aussagekräftig und für Menschen lesbar zu sein [14, S. 4]. Dies wird vor allem durch eine einheitliche Formatierung, kombiniert mit algebraischer Notation, erreicht. In Abbildung 3.1 ist exemplarisch ein Teil einer Spezifikation eines Geburtstagskalenders in *Z* angegeben [86]. Die verwendete Formatierung ist von der Spezifikation vorgegeben. Eine Einführung in die Syntax und die Verwendung von *Z* wurde von Spivey [86] veröffentlicht. Die Strukturierung wird durch eine Unterteilung in Schemata vorgenommen. In Abbildung 3.1a werden die im Geburtstagskalender enthaltenen Klassen definiert. Mit diesen wird in Abbildung 3.1a anschließend der Zustandsraum des Geburtstagskalenders definiert. Für die Methode `AddBirthday` wird anschließend in Abbildung 3.1b spezifiziert, dass ein Paar, bestehend aus einem Namen und einem Geburtstag, nur zum Verzeichnis hinzugefügt werden darf, wenn der Name vorher nicht bereits im Verzeichnis vorhanden ist.

Für die Unterstützung bei der Arbeit mit *Z* gibt es verschiedene Programme mit zum Teil vollkommen unterschiedlichen Einsatzgebieten. Die Palette reicht von WYSIWYG-Editoren für die Erstellung von *Z*-Spezifikationen bis hin zu Program-

men, die *Z*-Spezifikationen aus LaTeX-Quellen [51] extrahieren und die extrahierten Spezifikationen auf Typsicherheit und korrekte Syntax überprüfen. *Z* wurde in den 1980er Jahren in verschiedenen Projekten in der Industrie eingesetzt [16, 7] und 2002 als ISO-Standard verabschiedet [41]. Dennoch konnte sich die Sprache bisher nicht entscheidend durchsetzen, da Entwickler Sprachen mit hohem Anteil algebraischer Notation ungern einsetzen [26].

Mit dem Aufkommen der objektorientierten Methoden zur Modellierung von Software Anfang der 1990er Jahre wurden neue Techniken zur Spezifikation, wie die *Object Modelling Technique* (*OMT*) [83] und *Booch* [12], vorgestellt. Techniken dieser Art haben durch die visuelle Notation mit grafischen Symbolen eine große Anzahl von Anwendern gefunden, da für diese nicht mehr das mathematische Hintergrundwissen nötig ist, wie dies etwa bei der Verwendung von *Z* der Fall war. Allerdings hat sich gezeigt, dass den Diagrammen durch die nicht vorhandene Constraint-Sprache Möglichkeiten fehlen. Modellierungssprachen ohne Möglichkeiten zur Spezifikation von Anforderungen, die über das mit grafischen Ausdrücken Formulierbare hinausgehen, werden im Allgemeinen auch als Objekt-Modellierungssprachen der ersten Generation bezeichnet.

In den frühen 1990er Jahren haben Steve Cook und John Daniels gemeinsam an der Sprache *Syntropy* [17] gearbeitet, um akzeptierte graphische Notationen einer Modellierungssprache mit eine Constraint-Sprache zu vereinen. Die Sprache *Syntropy* ist eine objekt-orientierte Modellierungssprache, die Konzepte der Modellierungssprachen der ersten Generation (insbesondere *OMT*) aufgreift, verfeinert und durch eine Constraint-Sprache erweitert. Die Constraint-Sprache von *Syntropy* basiert dabei auf einer Teilmenge von *Z*, so dass es auch in *Syntropy* zu der Verwendung von mathematischen Symbolen aus der Mengentheorie und der Prädikatenlogik kommt. Es war nun immerhin möglich, die grafischen Modelle um zusätzliche Einschränkungen und Anforderungen zu erweitern, so dass die Probleme der Modellierungsprachen der ersten Generation beseitigt waren. Aber auch *Syntropy* konnte sich nicht entscheidend durchsetzen, da es daneben weitere Modellierungssprachen der zweiten Generation gibt und sich die Anwender nicht auf einen Standard einigen konnten. Ein Problem war, dass *Syntropy*, wie auch die anderen Modellierungssprachen der zweiten Generation, zwar eine Constraint-Sprache enthält, die Constraints aber, ähnlich wie in *Z*, hauptsächlich in algebraischer Notation formuliert werden.

Das Problem mit der algebraischen Notation war, dass die wenigsten Anwender eine entsprechende mathematische Ausbildung hatten, um mit ihr umgehen zu können [26, 67]. Daher wurde 1995 bei IBM (durch Jos Warmer und Steve Cook) an der OCL gearbeitet, die ohne algebraische Notationen auskommt und ein formales mathematisches Fundament hat [92]. Eine Einordnung der Notation verschiedener Constraint-Sprachen ist in Abbildung 3.2 dargestellt.

In Auflistung 3.1 sind die Constraints an die `AddBirthday`-Methode der Klasse `BirthdayBook` aus Abbildung 3.1 mit der OCL formuliert. Wie mit *Z* spezifiziert, werden auch mit der OCL Vor- und Nachbedingungen an die Methode formuliert. Die Vorbedingung – gekennzeichnet durch den Identifier `pre` – besagt, dass der hinzuzufügende Name nicht in der Menge `known` enthalten sein darf. Die Nachbedingung

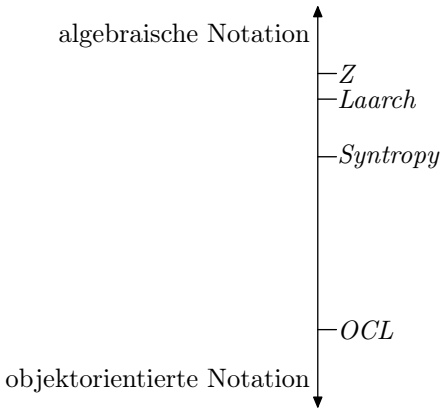


Abbildung 3.2.: Klassifikation der Notation von Constraint-Sprachen

Auflistung 3.1: OCL-Ausdruck für die Bedingung an die AddBirthday-Methode

```
context BirthdayBook::AddBirthday(name: NAME, date: DATE)
pre: self.known.select(n | n = name)->size = 0;
post: self.known.select(n | n = name)->size = 1;
```

– gekennzeichnet durch den Identifier `post` – stellt sicher, dass der Name nur einmal hinzugefügt wurde.

Die OCL wurde, wie erwähnt, von der OMG in die Version 1.1 der UML aufgenommen und dient seitdem, wie in Tabelle 3.1 dargestellt, als Constraint-Sprache für die UML. Die Ursprünge der OCL liegen in der Constraint-Sprache von *Syntropy* [67, 92]. Die historische Entwicklung der OCL ist in der Darstellung in Abbildung 3.3 aufgezeigt. Die UML bietet seit der Integration der OCL also die Möglichkeit, zusätzliche Einschränkungen modellierter Systeme formal zu spezifizieren. Zusätzliche Einschränkungen an ein Softwaresystem können auf verschiedene Phasen des Softwareentwicklungsprozesses Einfluss haben. Im folgenden Abschnitt wird dieser Einfluss betrachtet.

Tabelle 3.1.: Spezifikationssprachen und deren zugehörige Constraint-Sprachen

Spezifikationssprache	Constraint-Sprache
Z	Z constraints
OMT	-
Booch	-
Syntropy	Syntropy constraints
UML	OCL

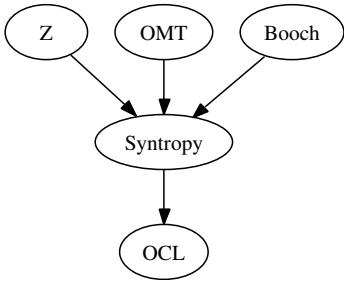
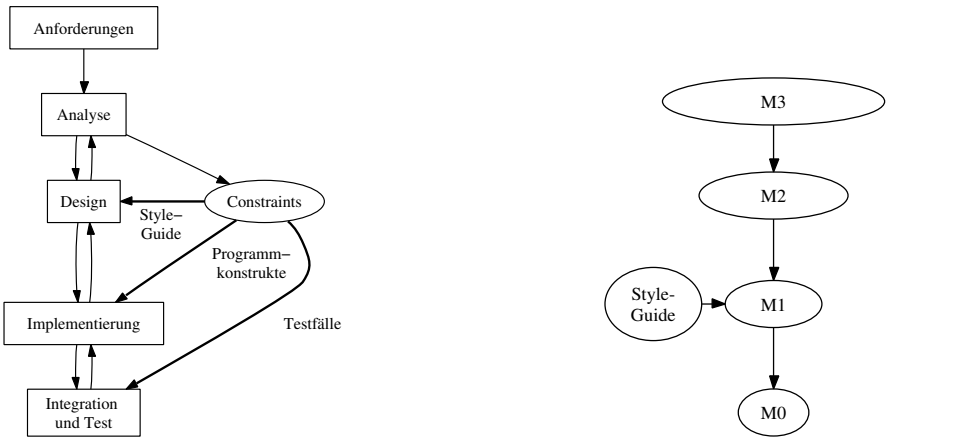


Abbildung 3.3.: Beziehungen von Sprachen, die zur OCL geführt haben

3.2. Anwendungsgebiete der OCL

Verschiedene Phasen der Softwareentwicklung werden, basierend auf ihrer zeitlichen Reihenfolge, in eine Sequenz aufeinander folgender Phasen unterteilt. Ein bekanntes Modell für die Anordnung dieser Phasen ist das *Wasserfallmodell* [31]. Die verschiedenen Phasen (Analysieren, Designen, Implementieren, Testen) sind als Sequenz in Abbildung 3.4a enthalten. Im Wasserfallmodell haben Erfahrungen, die in einer Phase gemacht werden, Einfluss auf vorhergegangene Phasen. Wurden diese Erfahrungen entsprechend umgesetzt, werden die Aufgaben aller nachfolgenden Phasen erneut ausgeführt.

Constraints werden, wie dargestellt, im Softwareentwicklungsprozess aus den, in der Analyse erhobenen, Daten parallel zum Design angelegt. Die aufgestellten Constraints haben Einflüsse auf das Design und können dann in der Implementierung in die Software integriert werden. So kann sichergestellt werden, dass die im Programm erzeugten Instanzen des entworfenen Modells den analysierten Anforderungen entsprechen. Werden geeignete Mechanismen in der Software bereitgestellt, lassen sich Constraints auch zur Laufzeit des entwickelten Softwaresystems auf den erzeugten



(a) Einflüsse im Wasserfallmodell **(b)** Einflüsse auf Metamodellebene

Abbildung 3.4.: Constraints im Softwareentwicklungsprozess

Modellinstanzen überprüfen. Auf Instanzen, die in Datenbanken verwaltet werden, können Invarianten mit Datenbankanfragen in der *Structured Query Language* (SQL) ausgewertet werden [18, 24, 53].

Die Softwareentwicklung mit Statecharts erfolgt in der Design-Phase des Wasserfallmodells. Der Einfluss von Constraints auf diese Phase ist die Einschränkung des Sprachumfangs auf eine robuste Teilmenge. Mit den in dieser Arbeit aufgestellten Constraints (siehe Kapitel 4.2) wird das Metamodell der Statecharts eingeschränkt. Betrachtet man die vier Ebenen der Metamodellierung [66, Kapitel 2.2.1], haben die Constraints des in dieser Arbeit aufgestellten Statechart Style-Guides also Einfluss auf Ebene M1. Dies ist in Abbildung 3.4b dargestellt.

Die OMG schlägt die OCL unter anderem zur Spezifikation der folgenden Arten von Constraints vor [67, Kap. 6.1.2]:

Invarianten: Diese werden bezogen auf Typen und Klassen im Klassendiagramm (Metamodell) spezifiziert. Die Einhaltung von Invarianten kann über geeignete Routinen in der entwickelten Software sichergestellt werden. Zusätzlich lassen sich die Invarianten zu einem diskreten Zeitpunkt auf einer Modellinstanz, die in der Software erstellt wird, über geeignete Methoden auswerten. Diese Auswertung kann keine Änderungen an der Instanz vornehmen, weil die Auswertung von OCL frei von Nebeneffekten ist [67].

Vor- und Nachbedingungen: Diese werden auf Methoden bezogen spezifiziert. Sie sollten in der Software umgesetzt werden, denn häufig sind in ihnen Informationen zur Funktionsweise einer Methode enthalten.

Die Auswertung von Constraints ist gemäß der einleitend vorgestellten Taxonomie zur Fehlervermeidung in Software (Abbildung 1.1) den Methoden zur automatisierten Fehlervermeidung zuzuordnen. Constraints werden auf dem Metamodell einer Software spezifiziert. Eine Auswertung erfolgt auf Instanzen des Metamodells, oder auch Modellinstanzen. Erfüllt eine Modellinstanz nicht alle Constraints, ist die Implementierung fehlerhaft. Auf diese Arbeit bezogen bedeutet dies, dass die Regeln zur Robustheit von Statecharts auf dem Metamodell der Statecharts formuliert werden. Der Inhalt der Regeln ist somit unabhängig von dem zu entwickelnden System. Konkrete Statecharts, mit denen ein System modelliert wurde, sind also Instanzen des Metamodells.

Damit die OCL zur Spezifikation von zusätzlichen Bedingungen in der Softwareentwicklung verwendet werden kann, muss es Werkzeuge geben, die zur Softwaremodellierung eingesetzt werden und geeignete Techniken zur Unterstützung anbieten. Im nächsten Abschnitt wird eine Auswahl einiger CASE-Programme mit OCL-Unterstützung vorgestellt.

3.3. Werkzeugunterstützung

In diesem Abschnitt werden die Ergebnisse einer Untersuchung verschiedener Modellierungswerkzeuge mit OCL-Unterstützung präsentiert. Das Ziel der Untersuchung

Tabelle 3.2.: Modellierungswerkzeuge mit OCL-Unterstützung

Werkzeug	OCL- Funktionalität				Editor		Add-In
	Syntax- / Typüberprüfung	Semantische Überprüfung	Code Generator	Constraint Auswertung	Syntax Highlighting	Code Vervollständigung	
ArgoUML	-	-	○[1]	-	-	-	-
Bold for Delphi	●	-	-	●	-	●	●
KeY	●	●	-	●	-	●	●
Oclarity	●	●	-	-	●	-	●
OCLE	●	●	●	●	●	●	-
Octopus	●	●	●	-	-	-	-
UMLAUT	●	-	●	-	-	-	-
USE	●	●	-	●	●	-	-

[1] OCL-Ausdrücke werden bei der Generierung nicht berücksichtigt.

war die Beurteilung, ob eines der betrachteten Modellierungswerkzeuge als Style-Checker für die in dieser Arbeit vorgestellte Robustheitsanalyse von Statecharts verwendet werden kann und welche Techniken bei der Auswertung von OCL zum Einsatz kommen. Für die Robustheitsanalyse ist eine Auswertung von Constraints auf Instanzen des Statechart-Metamodells notwendig. Tabelle 3.2 listet eine Auswahl verfügbarer CASE-Programme und Add-Ins auf, die OCL-Unterstützung in verschiedener Ausführung anbieten. Es wurden Funktionen aus zwei Bereichen untersucht.

Der erste betrachtete und für diese Arbeit relevante Bereich ist der Umfang der angebotenen OCL-Funktionalität. Zu diesem Bereich gehört unter anderem die Auswertung von Constraints auf Modellinstanzen. Diese Funktion ist wichtig für einen Einsatz als Style-Checker. Weitere betrachtete Funktionen dieses Bereichs sind die Syntax-Überprüfung, semantische Checks und der Umfang des Code Generators. Die Syntax-Überprüfung analysiert, ob eingegebene OCL-Ausdrücke eine korrekte Syntax aufweisen; beispielsweise sollte eine falsche Schreibweise der OCL-Identifier erkannt werden. Die semantischen Checks überprüfen, ob die verwendeten Klassen-, Methoden- und Attributnamen in den eingegebenen OCL-Ausdrücken dem Modell entsprechend korrekt verwendet werden. Ist ein Code Generator in einem Werkzeug enthalten, wurde betrachtet, ob eingegebene OCL-Ausdrücke im erzeugten Code berücksichtigt wurden.

Die daneben betrachteten Funktionen des in allen Werkzeugen vorhandenen OCL-Editors sind keine Voraussetzung zur Arbeit mit der OCL. Diese Funktionen werden jedoch bei einem Einsatz eines der betrachteten Werkzeuge als Style-Checker interessant, wenn Robustheisregeln angelegt oder bearbeitet werden sollen. Insbesondere

Funktionen, die den Entwickler bei der Arbeit mit OCL unterstützen, wurden hier betrachtet. Dies sind etwa Syntax-Highlighting und Funktionen zur Code-Vervollständigung. Die in Tabelle 3.2 aufgeführten Werkzeuge werden im Folgenden kurz vorgestellt.

ArgoUML: Das Open Source UML Modellierungswerkzeug *ArgoUML* [4] ist vollständig kompatibel zum UML 1.4 Standard. Aus den erstellten Diagrammen lässt sich Java-Code automatisch generieren. Jedoch werden eventuell eingegebene Constraints in Java Kommentaren im Quellcode eingefügt. Die OCL-Ausdrücke lassen sich gezielt zu Klassen, Attributen und Methoden hinzufügen und auch nachträglich bearbeiten. Die Unterstützung bei der Eingabe der OCL-Ausdrücke ist auf einige Schlüsselwörter und mathematische Operatoren beschränkt, die aus einer Drop-down Liste ausgewählt werden müssen. Am Ende der Eingabe wird die korrekte Syntax, nicht aber die korrekte Typisierung überprüft. Eine Auswertung der Constraints auf Modellinstanzen ist nicht möglich.

Bold for Delphi: *Bold for Delphi* von Borland [13] ist eine CASE-Programm, das als Bestandteil der *Borland Delphi* Entwicklungsumgebung kommerziell vertrieben wird. Die im angebotenen OCL-Editor eingegeben Bedingungen werden auf korrekte Syntax überprüft. Aus den eingegebenen Bedingungen lassen sich SQL-Abfragen erzeugen. Im entwickelten Programm lässt sich zur Laufzeit durch den Aufruf spezieller Methoden überprüfen, ob die Instanzen modellierter Klassen alle formulierten Bedingungen erfüllen. Die aktuelle Weiterentwicklung von *Bold for Delphi* wird unter dem Namen *ECO* ebenfalls als Bestandteil der Borland Entwicklungsumgebungen vermarktet.

KeY: Das an der Universität Karlsruhe entstandene Projekt *KeY* [42] ist ein Add-In für *TogetherCC*. Mit Hilfe von *KeY* lassen sich zum Einen die OCL-Ausdrücke auf korrekte Typisierung und Semantik überprüfen, und zum Anderen lassen sich konkrete Implementierungen gegen das Modell verifizieren. *KeY* übersetzt die OCL-Bedingungen in Ausdrücke in dynamischer Logik, die wiederum als Eingabe eines Theorem-Beweisers verwendet werden.

Oclarity: Das kommerziell vertriebene Rational Rose Add-In *Oclarity* der Firma EmPowerTec [21] integriert verschiedene Funktionen zur Arbeit mit OCL in das CASE-Werkzeug. Neben Funktionen zur Überprüfung der Syntax und Semantik der Ausdrücke gemäß OCL 2.0, bietet der Editor Syntax-Highlighting und Code-Vervollständigung. Die Überprüfung der Syntax und Semantik einzelner Constraints lässt sich an verschiedenen Stellen im Programm aufrufen. Um einen Überblick über alle spezifizierten Constraints zu erhalten, lassen sich alle Constraints über einen Menüpunkt einer Überprüfung unterziehen.

OCLE: Das *Object Constraint Language Environment* (*OCLE*) ist an der Universität in Cluj-Napoca (Rumänien) entwickelt worden [6]. Schwerpunkt des Pro-

gramms ist das Anlegen und die Überprüfung von OCL-Ausdrücken auf bestehenden UML-Modellen. Der bereitgestellte Editor unterstützt den Benutzer bei der Eingabe mit Syntax-Highlighting und Code-Vervollständigung. Die eingegebenen OCL-Ausdrücke lassen sich auf korrekte Syntax und Semantik überprüfen. Der integrierte Code-Generator erzeugt aus ausgewählten Modellen und den darüber spezifizierten OCL-Ausdrücken Java-Code.

Octopus: Das *OCL Tool for Precise UML Specifications* (*Octopus*) ist ein Eclipse-Plugin [91]. Mit *Octopus* können Syntax, Semantik und korrekte Typisierung von OCL-Ausdrücken der OCL (Version 2.0) überprüft werden. Zusätzlich bietet Programm die Möglichkeit Java-Code zu erzeugen.

UMLAUT: Das Werkzeug *Unified Modeling Language All pUrposes Transformer* (*UMLAUT*) [40] wird nach seiner Fertigstellung als Freeware verfügbar sein. Es ist ein Werkzeug, das zur Bearbeitung von UML-Modellen eingesetzt werden kann. Für die Arbeit mit OCL bietet *UMLAUT* einen Syntax-Checker und die Möglichkeit Eiffel-Code zu erzeugen.

USE: Das *UML-based Specification Environment* (*USE*) [82] wurde von Mark Richters an der Universität Bremen entwickelt. Als Implementationssprache wurde Java verwendet. Das Programm arbeitet mit einer Teilmenge der UML und dient zur Spezifikation von Informationssystemen. Das spezifizierte System lässt sich visuell simulieren. Auf Zuständen eines simulierten Systems lassen sich alle eingegebenen OCL Bedingungen überprüfen, um sicherzustellen, ob das System die Anforderungen erfüllt.

Neben diesen hier vorgestellten Programmen gibt es Programme, wie etwa *Fujaba* [69], die OCL-Unterstützung für kommende Versionen geplant haben. Basierend auf dem *Dresden OCL Toolkit* wird eine OCL-Unterstützung für *Fujaba* entwickelt. So soll es möglich sein, mit OCL Bedingungen für Klassen- und *Story*-Diagramme zu formulieren, die automatisch überprüft werden können [87].

Des Weiteren wurde im Rahmen des IST Omega Projektes [1] durch Kyas eine formale Verifikation von OCL-Ausdrücken und den zugehörigen UML-Modellen umgesetzt. Im Rahmen dieses Projektes wurde der Sprachumfang der UML und der OCL jedoch beschränkt, um als Eingabesprache für den Theorembeweiser PVS dienen zu können. Mit Hilfe des Theorembeweisers lässt sich die formale Verifikation entsprechend übersetzter UML-Modelle und OCL-Ausdrücke manuell durchführen [48, 49].

Fazit

Die untersuchten Modellierungswerkzeuge bieten OCL-Unterstützung in verschiedenster Ausprägung an. Allen gemein ist, dass sie die Möglichkeit bieten, OCL-Ausdrücke aufzunehmen. Die vom Werkzeug angebotene Unterstützung bei der Eingabe wie Syntax-Highlighting oder Code-Vervollständigung ist bei der Arbeit mit OCL-Code keineswegs selbstverständlich – nur *OCLE* bietet beides. Auch bei den

angebotenen Funktionen zur Analyse der Eingabe fallen gravierende Unterschiede auf. Die meisten der untersuchten Werkzeuge bieten zwar eine Überprüfung der Syntax – eine semantische Überprüfung ist jedoch keineswegs Standard. Auch bei der unterstützten Version der OCL fallen Unterschiede auf. Bisher unterstützen nur *Oclarity*, *Octopus* und *OCLE* die Version 2.0 der Constraint-Sprache.

Funktionen zur Auswertung der Constraints werden nur von drei der untersuchten Werkzeugen angeboten. (1) *Bold* bietet diese Funktion jedoch nur aus Delphi-Programmen heraus an, was einer plattformunabhängigen Verwendung entgegenwirkt. (2) *OCLE* bietet zur Auswertung auf Modellinstanzen eine Übersetzung in Java an. Die erzeugten Java-Code-Fragmente müssen jedoch manuell in ein Programm integriert werden, was wiederum Kenntnisse der Programmiersprache und der Anwendung voraussetzt und damit für jede neu formulierte Bedingung mitunter tiefgreifende Anpassungen an einem (bereits bestehenden) Programm nötig macht. (3) *USE* bietet eine Auswertung nur während der Simulation eines modellierten Informationssystems an. Eine ausschließliche Simulation von Statecharts ist jedoch nicht vorgesehen. Auch hier ergeben sich also gravierende Nachteile für die Verwendung in der Robustheitsanalyse von Statecharts liegen. Also eignet sich keines dieser Werkzeuge für einen Einsatz als Style-Checker auf der Basis von OCL. Ein Style-Checker auf der Basis von OCL sollte die folgenden drei Funktionen beinhalten: (1) Syntaxprüfung von OCL-Ausdrücken, (2) Semantische Überprüfung der OCL-Ausdrücke und (3) Methoden zur Constraint-Auswertung. Die ersten beiden Funktionen sind für Entwickler von neuen Regeln hilfreich, da der Style-Checker bei fehlerhafter Syntax oder semantischen Fehlern entsprechende Meldungen zurückliefern kann. Funktionen zur Constraint-Auswertung sollten bei nicht erfüllten Constraints entsprechende Hinweise liefern und bilden die Kernfunktionalität eines Style-Checkers.

In der UML wurde die OCL zur Spezifikation von Regeln der syntaktischen Korrektheit verwendet. Neben anderen Regeln wurden auch die Regeln aus der UML in den im nächsten Kapitel vorgestellten Style-Guide aufgenommen. Bevor der Style-Guide jedoch vorgestellt wird, erfolgt zu Anfang des Kapitels die Einführung einer Taxonomie des Style-Checkings auf Statecharts. Basierend auf dieser Taxonomie wird die Regelmenge des Style-Guides in drei Bereiche unterteilt. Neben den Regeln aus der UML werden auch die Regeln aus dem Bereich der syntaktischen Robustheit mit der OCL spezifiziert.

4. Style-Checking von Statecharts

Statecharts, die zu den visuellen Programmiersprachen zählen, werden, wie bereits erwähnt, in der modellbasierten Entwicklung von zum Teil sicherheitskritischen Soft- und Hardware-Systemen verwendet. Insbesondere in der Entwicklung von solchen sicherheitskritischen Systemen ist Fehlervermeidung wichtig, da ein Ausfall des Systems, wie erwähnt, enormen wirtschaftlichen Schaden nach sich ziehen kann (siehe Kapitel 2). Wie einleitend erwähnt, wurden auch für die modellbasierte Entwicklung Style-Guides zum Style-Checking von Statecharts entwickelt. Das Style-Checking lässt sich auch auf Statecharts mit Style-Checkern automatisiert durchführen. Aus der automatischen Anwendung von Style-Guides auf Statecharts ergeben sich hier ebenfalls die Vorteile wie bessere Lesbarkeit, Wartbarkeit und geringere Fehleranfälligkeit. Ein besonderer Vorteil der Anwendung von Style-Guides auf Statecharts ist, dass der Datenaustausch zwischen verschiedenen Modellierungswerkzeugen ermöglicht wird [62].

Im Folgenden wird die von Schaefer [85] eingeführte Taxonomie des Style-Checkings auf Statecharts kurz vorgestellt. Dieser Taxonomie folgend, wird anschließend ein Regelkatalog für das Style-Checking auf Statecharts eingeführt. Abschließend wird die Unterstützung bei der Robustheitsanalyse durch verschiedene Modellierungswerkzeuge und Style-Checker betrachtet.

4.1. Taxonomie des Style-Checking von Statecharts

Statische Analysen von Statecharts werden nach Schaefer wie in Abbildung 4.1 in zwei Bereiche unterteilt [85, S. 40ff]: Zum Einen in den Bereich der die syntaktische und semantische Korrektheit analysiert (*Correctness*), und zum Anderen in den Bereich des Style-Checkings. Die syntaktische Korrektheit eines Statecharts ist Grundvoraussetzung bevor das Style-Checking wertvolle Hinweise liefern kann und sollte daher immer vor dem Style-Checking überprüft werden. Das Style-Checking wird weiter in die Bereiche der syntaktischen Analysen und den Bereich der Semantischen Robustheit unterteilt. Im Bereich der syntaktischen Analysen wird zwischen folgenden Bereichen unterschieden

Lesbarkeitsanalysen werden in Style-Guides für Statecharts verwendet, um ein einheitliches Layout zu garantieren (bspw. „Einen Initial-Zustand immer links, einen Final-Zustand immer rechts im Statechart anordnen.“). Im Projekt KIEL, in dem der im Rahmen dieser Arbeit entwickelte Style-Checker als Plug-In

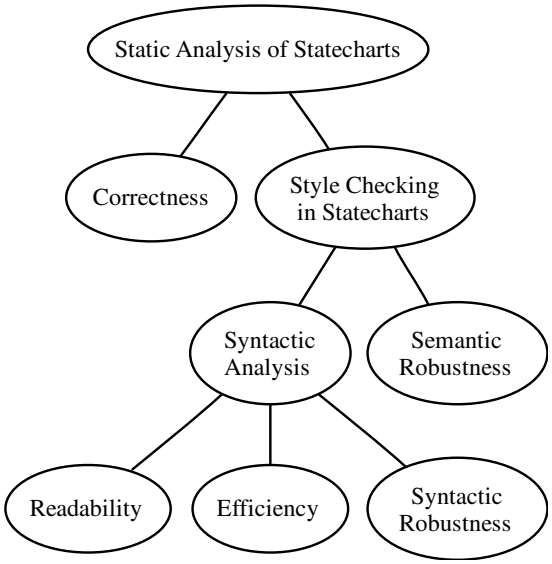


Abbildung 4.1.: Statechart Style Checking Taxonomie (Quelle: [85])

integriert wurde, wird die Lesbarkeit durch integrierte Funktionen zum automatischen Layout gewährleistet [45, 74].

Die Effizienz von Statecharts, in diesem Fall gemessen an der Anzahl der Zustände, auch als Kompaktheit bezeichnet, lässt sich durch das Optimieren von modellierten Systemen beeinflussen. Beispielsweise kann ein *OR-CompositeState* mit nur einem enthaltenen Zustand durch einen einzigen Zustand ersetzt werden (vergleiche Abbildung 4.3). Eine Funktion, die diese und andere Optimierungen automatisiert durchführt, wurde in das KIEL-Projekt integriert [47, 76].

Syntaktische Robustheit Die Regeln zur Syntaktischen Robustheit zielen darauf ab, fehleranfällige Konstruktionen oder solche, die das Verständnis des Modells erschweren (z.B. Interlevel Transitionen in Stateflow/Simulink [54]), zu vermeiden, beziehungsweise Fehler, die im Entwicklungsprozess entstehen („Jeder Zustand außer dem *Root*-Zustand und Initial-Zuständen sollte mindestens eine eingehende Transition besitzen.“ [63]), aufzuzeigen. Es gibt Regeln, die dialektspezifisch d.h. nur auf einem Statechart-Dialekt gültig sind, und dialektübergreifende, die auf jedem Dialekt ausgewertet werden können.

Die Semantische Robustheit von Statecharts befaßt sich mit Problemen der Modellierung, die weit über syntaktische Analysen hinausgehen, wie beispielsweise die Überprüfung auf Deadlocks oder Erreichbarkeit. Style-Guides, mit Regeln aus den vorgestellten Bereichen, wurden unter Anderem von Mutz [63], dem *MathWorks Automotive Advisory Board* (MAAB) [60] und der Ford Motor Company [28] veröffentlicht. Die letzten beiden Arbeiten befassen sich explizit mit Matlab Simulink/-Stateflow. Scaife et al. [84] und Huuck [39] haben daneben weitere Ansätze für die sichere Entwicklung mit Simulink/Stateflow vorgestellt, indem sie die Sprache auf

eine „sichere“ Teilmenge einschränken. Im folgenden Abschnitt wird ein dialektübergreifender Style-Guide, entsprechend der eingeführten Taxonomie (Abbildung 4.1), vorgestellt.

4.2. Ein Style-Guide für Statecharts

In dem Bestreben einen Style-Guide für Statecharts zu entwickeln, der möglichst auf einer Vielzahl von Dialekten, wie UML, Safe State Machines oder Simulink/Stateflow anwendbar ist, wurden Regeln verschiedener Style-Guides hinsichtlich ihrer Anwendbarkeit auf andere Dialekte untersucht. Die Regeln wurden gemäß der oben eingeführten Taxonomie kategorisiert. Wie erwähnt, ist syntaktische Korrektheit von Statecharts die Grundvoraussetzung, bevor weitergehende Checks die Robustheit betreffend ausgewertet werden können. Die also in diesen Style-Guide aufgenommenen Regeln zur Syntaktischen Korrektheit werden im folgenden Abschnitt vorgestellt.

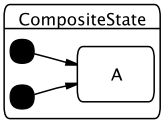
Syntaktische Korrektheit

Es gibt CASE-Programme zur Modellierung mit der UML, die die korrekte Syntax von *StateMachine*s nicht überprüfen (z. B. *ArgoUML*, vergleiche Kapitel 2.2). Also sollte die korrekte Syntax von *StateMachine*s explizit überprüft werden, um verlässliche Aussagen über die Robustheit zu erhalten. Eine Überprüfung der korrekten Syntax von Statecharts, die aus Modellen im XMI-Format eingelesen werden, ist ebenfalls unerlässlich, da nicht sicher gestellt werden kann, dass das erzeugende Programm die korrekte Syntax der exportierten Statecharts überprüft. Die syntaktische Korrektheit von Statecharts anderer Dialekte wird durch einige Modellierungswerkzeuge, wie *Esterel Studio* oder *Matlab Simulink/Stateflow*, über einen syntaxgerichteten Editor oder Syntax-Checks während der Code-Erzeugung sichergestellt.

Der hier entwickelte Style-Guide zielt, wie erwähnt, jedoch darauf ab, die Robustheit von Statecharts verschiedener Dialekte analysieren zu können. Damit auch für *StateMachine*s verlässliche Aussagen über die Robustheit getroffen werden können, enthält dieser hier vorgestellte Style-Guide also auch die syntaktische Korrektheit von *StateMachine*s betreffende Regeln. Diese Regeln sind der UML-Spezifikation [66, 65] entnommen. Sie werden in der UML *Well-formedness*-Regeln genannt. Ihr Zweck wird von der OMG wie folgt beschrieben: Die *Well-formedness*-Regeln „[. . .] specify constraints over attributes and associations defined [with]in the [Statechart] meta model“ [66, Kapitel 2.3.2.2]. Die Regeln sind in der UML durch einen entsprechenden OCL Ausdruck und eine natürlich-sprachliche Beschreibung der jeweiligen Regel angegeben. Die *Well-formedness*-Regel *CompositeState*-1 beispielsweise besagt: „A composite can have at most one initial vertex“ [66, Kapitel 2.12.3.1]. In Abbildung 4.2a ist der zugehörige OCL-Ausdruck angegeben. Eine Verletzung der Regel *CompositeState*-1 ist in Abbildung 4.2b angegeben. Zusätzlich zu der Möglichkeit, mit den *Well-formedness*-Regeln Bedingungen an einzelne Modellelemente und deren Beziehungen untereinander zu definieren, werden mit ihnen auch Mehr-

```
self.subvertex->select(v | v.oclIsKindOf(Pseudostate))
  ->select(p: Pseudostate | p.kind = #initial)->size <= 1
```

(a) Der OCL-Ausdruck



(b) Verletzung der Regel

Abbildung 4.2.: Beispiel der *Well-formedness*-Regel *CompositeState* Nummer 1

deutigkeiten, die sich aus der UML-Spezifikation ergeben, beseitigt. Die in Version 1.3 der UML angegebenen 31 *Well-formedness*-Regeln für *StateMachine*s wurden in Version 2.0 der UML auf eine Gesamtanzahl von 57 erweitert, um auf Änderungen im Metamodell zu reagieren.

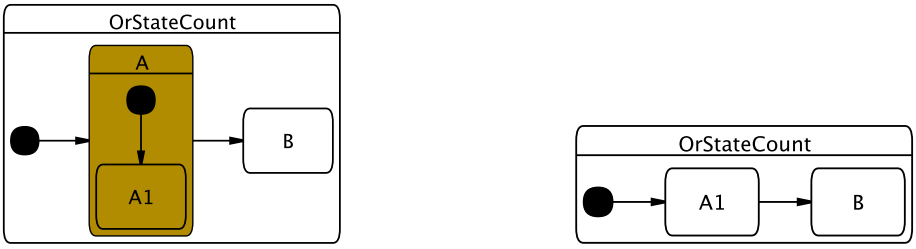
Mit den oben erwähnten und in diesen Style-Guide aufgenommenen *Well-formedness*-Regeln lässt sich also die syntaktische Korrektheit von Statecharts – insbesondere den *StateMachine*s – überprüfen. Wie aus dem angegebenen Beispiel einer *Well-formedness*-Regel ersichtlich, sind die überprüften Sachverhalte nicht kompliziert. Sie können sicherlich durch den Menschen überprüft werden, führen aber in einer manuellen Code-Review oft dazu, dass der Entwickler von den eigentlichen Problemen abgelenkt wird [72]. Eine automatisierte Überprüfung dieser Regeln ist deshalb zu bevorzugen. Über die relativ einfachen Überprüfungen im Rahmen der *Well-formedness*-Regeln hinausgehend, sind Robustheitsanalysen der nächste Schritt, der im Rahmen des *Style Checking* durchgeführt werden kann. Gemäß der vorgestellten Taxonomie (vergleiche Abbildung 4.1) enthält dieser hier vorgestellte Style-Guide auch Regeln zur syntaktischen und semantischen Robustheit. Die bereits in der verwandten Diplomarbeit [85] vorgestellten Regeln zur syntaktischen Robustheit werden im folgenden Abschnitt aufgegriffen, ausführlich beschrieben, und um neue Regeln erweitert.

Syntaktische Robustheit

Die Regeln für die Syntaktische Robustheit von Statecharts, die in diesem Abschnitt präsentiert werden, stellen das Ergebnis einer Untersuchung verschiedener Statechart Style-Guides, beziehungsweise Arbeiten und praktischer Erfahrungen aus der Modellierung auf diesem Gebiet dar.

Die folgenden Regeln sind, durch die Arbeit von Mutz [63, Seite 144f] inspiriert, in das Regelwerk aufgenommen worden.

EqualNames: Alle Zustände eines Statecharts sollten unterschiedlich benannt werden. Wird diese Regel in einem Statechart durchgängig eingehalten, so werden anfallende Wartungsarbeiten oder Fehlerlokalisationen erheblich vereinfacht, da zu bearbeitende Zustände einwandfrei anhand des Namens identifiziert werden können. Eine Verletzung dieser Regel kann in jeder Phase der Modellierung durch Unachtsamkeit des Entwicklers verursacht werden, indem er



(a) Fehlerhaftes Statechart. **(b)** Optimiertes Statechart.

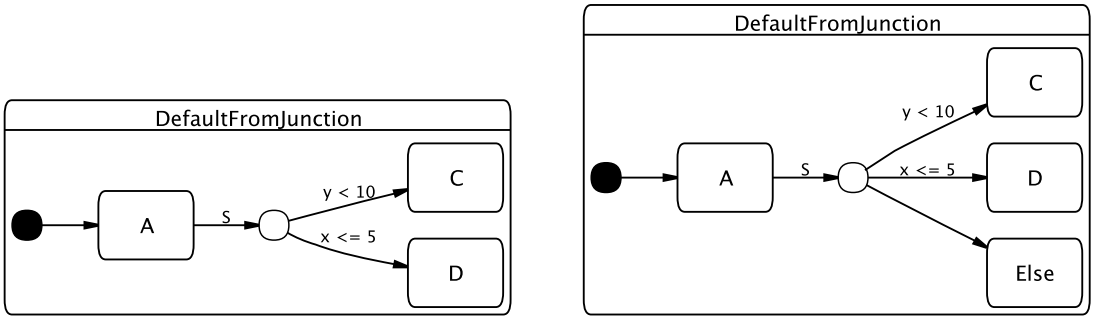
Abbildung 4.3.: Beispiel für die Regel *ORStateCount*

einen Zustand hinzufügt und einen Namen für den neuen Zustand vergibt, der bereits einmal verwendet wurde oder einen Bereich eines Statecharts kopiert und einfügt.

IsolatedState: Alle Zustände sollten ein- und ausgehende Transitionen haben. Ausnahmen sind finale und initiale Zustände und der Root-Zustand, für die entsprechend der syntaktischen Korrektheit andere Richtlinien gelten. Finale Zustände dürfen keine ausgehenden Transitionen haben. Dies wird durch die *Well-formedness*-Regel *FinalState-1* (Siehe Kapitel 6, Tabelle 6.1) überprüft. Initiale Zustände dürfen keine eingehenden Transitionen aufweisen. Auch hierfür existiert eine *Well-formedness*-Regel (*PseudoState-1*, siehe Tabelle 6.1). Root-Zustände dürfen weder ein- noch ausgehende Zustände aufweisen. Hat eine Zustand weder ein- noch ausgehende Transitionen ist er isoliert (*isolated*). Ein isolierter Zustand besitzt keinerlei Funktion und sollte daher ebenfalls vermieden werden, um die Wartbarkeit des Systems zu erhöhen.

InitialState: Alle Regionen, beziehungsweise nicht-parallelen hierarchischen Zustände, sollten genau einen initialen Zustand enthalten. Diese Regel geht einher mit der Regel *InterlevelTransition*. Wird ein hierarchischer Zustand über eine Interlevel-Transition betreten, beginnt die Ausführung des betretenen hierarchischen Zustands während der Simulation nicht an einem initialen Zustand. Das Verständnis des Modells wird jedoch erleichtert, wenn die Ausführung in einem Statechart immer an einem initialen Zustand beginnt.

OrStateCount: Alle OR-Zustände sollten mehr als einen Zustand enthalten. Eine Auswertung dieser Regel liefert Hinweise für mögliche Optimierungen eines Statecharts, da OR-Zustände, die nur einen Zustand enthalten dialektunabhängig optimiert werden können. Ein in dieser Hinsicht optimiertes Statechart erhöht die Lesbarkeit für den Entwickler, da die Anzahl der benötigten graphischen Objekte verringert wird. In Abbildung 4.3a ist exemplarisch ein Statechart angegeben, in dem der OR-Zustand A wie in Abbildung 4.3b angegeben optimiert werden kann.



(a) Fehlerhaftes Statechart. **(b)** Korrigiertes Statechart.

Abbildung 4.4.: Beispiel für die Regel *DefaultFromJunction*

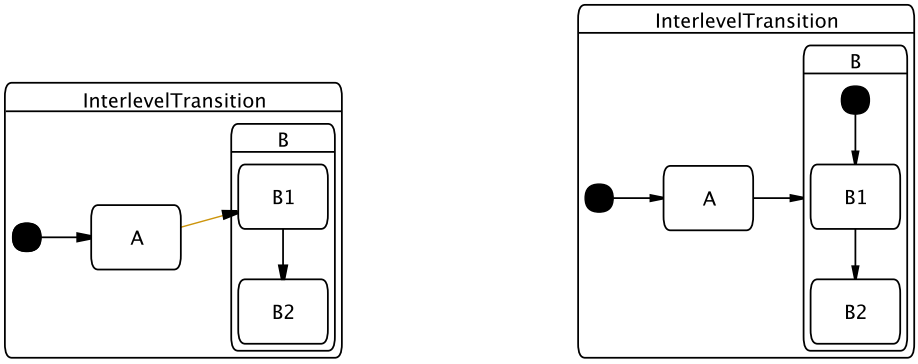
RegionStateCount: Alle parallelen Regionen eines AND-Zustands sollten mehr als einen Zustand enthalten. Eng verwandt mit der Regel *OrStateCount*, liefert auch die Auswertung dieser Regel Hinweise auf Optimierungen, mit denen die Lesbarkeit erhöht werden kann.

Aus dem Ford-Style-Guide [28] wurde die folgende Regel extrahiert, da diese auch auf anderen Dialekten ausgewertet werden kann.

DefaultFromJunction: Von Connective junctions (Siehe Tabelle 2.1) sollte immer ein Teilstück einer Transition ohne Trigger und Bedingung abgehen, damit der Programmablauf nicht stoppt. Das unbeschriftete Teilstück einer Transition wird dann die *Default*-Transition genannt. Diese wird genommen, wenn die Bedingungen aller anderen ausgehenden Transitionen nicht erfüllt sind. Fehler im Zusammenhang mit dieser Regel werden sowohl durch Unachtsamkeit als auch durch Unwissenheit des Entwicklers verursacht. In Abbildung 4.4a ist der fehlerhafte Teil eines Statecharts angegeben. In dem Fall, daß die Variable `x` den Wert 10 hat, ist keine der beiden Bedingungen erfüllt. Der Kontrollfluss stoppt, weil kein unbeschriftetes Segment modelliert wurde. In Abbildung 4.4b ist ein Beispiel für die entsprechende Korrektur angegeben.

Aus der eigenen praktischen Erfahrung im Umgang mit Statecharts wurden die folgenden Regeln erstellt.

InterlevelTransition: In einem Statechart sollten keine Interlevel-Transitionen verwendet werden. Interlevel-Transitionen sind Transitionen, die Hierarchie-Grenzen überschreiten, wie in Abbildung 4.5a gezeigt. Solche Transitionen sollten vermieden werden, weil ihre Verwendung gegenüber einer entsprechenden Modellierung mit initialen Zuständen keinen Mehrwert bietet und zudem das Verständnis eines Modells erschwert wird. Insbesondere ungeübte Entwickler können das mit Interlevel-Transitionen ausgedrückte Verhalten missverstehen; beispielsweise die Reihenfolge der ausgeführten (*Entry*-)Aktivitäten und die



(a) Fehlerhaftes Statechart. **(b)** Korrigiertes Statechart.

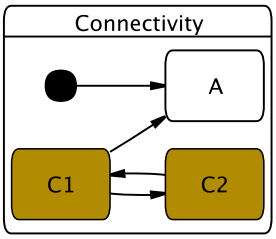
Abbildung 4.5.: Beispiel für die Regel *InterlevelTransition*

gleichzeitige Aktivierung aller parallelen Bereiche. Neben der Erleichterung für den Entwickler, ist die Einhaltung dieser Regel außerdem ein wichtiger Aspekt beim Datenaustausch zwischen verschiedenen Modellierungswerkzeugen, da es in einigen Dialekten (u. a.Safe State Machines) keine Interlevel-Transitionen gibt. Werden durch die Auswertung dieser Regel Interlevel-Transitionen in einem Statechart lokalisiert, können diese wie in Abbildung 4.5b dargestellt behoben werden. Man setzt als Ziel der Interlevel-Transition den umgebenden hierarchischen Zustand des vorherigen Zielzustands und fügt entsprechend initiale Zustände in den hierarchischen Zustand ein.

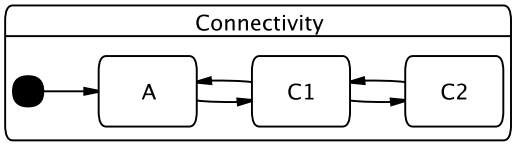
TransitionLabels: Alle Transitionen sollten mit einem Trigger beschriftet werden. Für zusammengesetzte Transitionen bedeutet dies, dass mindestens ein Teilstück einer Transition mit einem Trigger beschriftet sein sollte. Diese Regel soll das Verständnis des Lesers erhöhen, da die Bedeutung sofort ersichtlich ist. Es gibt Statechart Dialekte mit einem Default-Signal (z.B. `tick` in *Esterel Studio*). Ist kein Trigger für eine Transition beschriftet, so wird das Default-Signal während der Simulation für den Entwickler unsichtbar einer Transition zugeordnet. Daher sollten alle Transitionen explizit beschriftet werden.

Connectivity: Für jeden Zustand sollte es einen gerichteten Pfad von einem initialen Zustand geben. Diese Regel erweitert die Regel *MiracleStates* von Mutz [63, Seite 144], die besagt, dass alle Zustände außer initialem und root-Zustand eingehende Transitionen haben sollten. Darüber hinaus zeigt die Regel *Connectivity* aber wie in Abbildung 4.6a sogar Fehler im Modell auf, in denen ein Zustand (`C1`, `C2`) nicht betreten werden kann, obwohl er eingehende Transitionen aufweist. Die Abbildungen 4.6b und 4.6c zeigen zwei Möglichkeiten, wie fehlerhafte Statecharts korrigiert werden können.

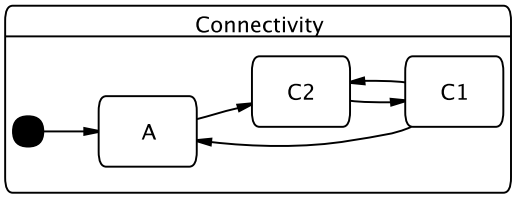
So unterschiedlich die Regeln für die syntaktische Robustheit sind, so vielfältig sind auch die Vorgehensweisen zur Behebung von Verletzungen der Regeln. Wie vorgestellt, muss in jedem Fall von Neuem mit der Spezifikation abgeglichen werden,



(a) Fehlerhaftes Statechart.



(b) Korrektur durch Transition von A nach C1.



(c) Korrektur durch Transition von A nach C2.

Abbildung 4.6.: Beispiel für die Regel *Connectivity*

was zu tun ist, um ein fehlerhaftes Statechart zu korrigieren. Die hier angegebenen korrigierten Statecharts stellen nur Beispiele zur Illustration möglicher Korrekturen dar. Eine Ergebnisanalyse über alle im entwickelten Style-Checker umgesetzten Regeln ist in Kapitel 6 enthalten.

Semantische Robustheit

Die folgenden Regeln für den Bereich der Semantischen Robustheit werden hier der Vollständigkeit halber angegeben, sind allerdings Gegenstand der engverwandten Diplomarbeit von Schaefer und werden dort ausführlich behandelt [85]. Als Prädikate werden hier die Trigger und optionalen Bedingungen einer Transition bezeichnet.

Dwelling: Die Prädikate aller eingehenden und ausgehenden Transitionen eines Zustands sollten paarweise disjunkt sein oder zumindest nicht vollständig überlappend sein.

Race Condition: Nebenläufige Schreib- oder Lese-/Schreibzugriffe auf eine Variable sollten in parallelen Zuständen nicht auftreten.

Transition Overlap: Alle von einem Zustand (direkt oder indirekt) ausgehenden Transitionen sollten disjunkte Prädikate haben.

Wie erwähnt können Regeln aus Style-Guides von Style-Checkern automatisch überprüft werden. Im folgenden Abschnitt werden exemplarisch verschiedene Modellierungswerkzeuge und Style-Checker für Statecharts untersucht. Es wird betrachtet, wie ausgeprägt die Unterstützung zur Analyse der Robustheit von Statecharts im Rahmen des hier vorgestellten Style-Guides ist.

4.3. Style-Checker

Wie einleitend erwähnt, werden Style-Checker zur automatisierten Überprüfung von Designregeln, die in Style-Guides gesammelt werden, verwendet. In der textuellen Programmierung sind diese Style-Checker sehr ausgereift und lassen sich umfassend an eigene Bedürfnisse anpassen. In der modellbasierten Entwicklung, in der eine automatische Analyse von Designregeln – wie den vorgestellten Regeln zur Robustheit – wertvolle Unterstützung bei der Fehlervermeidung darstellt, ist das Style-Checking bisher jedoch nicht sehr ausgereift.

Wie schon erwähnt, ist bereits in der Entwicklung insbesondere mit der UML eine Überprüfung auf Korrektheit eines Statecharts unerlässlich. Denn obwohl diese leicht automatisch durchführbar ist, existieren CASE-Werkzeuge (wie *ArgoUML*), die keine entsprechende Funktion besitzen (siehe Kapitel 4.2). Die Korrektheit von Statecharts anderer Dialekte, etwa Safe State Machine und Stateflow, wird von den entsprechenden Entwicklungsumgebungen (*Esterel Studio* und *Matlab Simulink/-Stateflow*) durch syntax-gerichtete Editoren und durch integrierte Funktionen in der Code-Generierung gewährleistet und muss daher auf solchen Statecharts nicht überprüft werden. In beiden Werkzeugen gibt es darüber hinaus bereits Funktionen zur statischen Analyse. Diese sind jedoch nicht explizit abrufbar und die mitgelieferten Analysen lassen sich weder erweitern, noch an eigene Anforderungen anpassen. Ein weiterer Nachteil der mitgelieferten Analysen ist, dass sie entweder sehr einfache oder sehr anspruchsvolle Analysen von der Art eines *Model Checking* bieten [33, 46, 64]. Analysen der Robustheit, wie sie in dieser Arbeit betrachtet werden und zwischen den beiden Arten liegen, lassen sich nicht durchführen. Um diese Lücke zu füllen, wurden einige nicht-kommerzielle und auch kommerzielle Style-Checker entwickelt. Die Werkzeuge *Mint* [81], *State Analyzer* [32, 46] und *Guideline Checker* [61] sind für den Dialekt Stateflow entwickelt worden. Statecharts anderer Dialekte lassen sich nicht analysieren. Daher sind diese Style-Checker für eine hier angestrebte dialektübergreifende Robustheitsanalyse nicht anwendbar.

Im Gegensatz zu den bereits vorgestellten Style-Checkern ist der *Regel Checker* [63, 64] dialektunabhängig. Mit dem *Regel Checker* ist es möglich 71 verschiedene Checks auf Statecharts verschiedener Dialekte auszuführen. Die mitgelieferten, auf syntaktische Abfragen beschränkten Checks, lassen sich über verschiedene Parameter an eigene Bedürfnisse anpassen. Weitere Checks können entweder in OCL oder direkt in Java programmiert werden. Der Einsatz von OCL bietet bei der Erweiterung des Regelwerks Vorteile, da eine Erweiterung leicht möglich ist. Denn für die Formulierung neuer Checks mittels OCL werden, wie erwähnt, keine detaillierten Kenntnisse einer Programmiersprache benötigt [63, Seite 126]. Jedoch werden die mit OCL formulierten Regeln im *Regel Checker* in einem interpretativen Ansatz ausgewertet. Größter Nachteil dieses Vorgehens ist die geringe Geschwindigkeit der Ausführung. Insbesondere im Umfeld kommerzieller Softwareentwicklung ist die Geschwindigkeit eingesetzter Style-Checker jedoch ein wichtiges Kriterium, das über den Einsatz eines solchen Werkzeugs entscheidet. Zudem sind Erweiterungen des

interpretierten Sprachumfangs ohne zum Teil tiefgreifende Veränderungen des zugrunde liegenden Interpreters nicht durchführbar. Ein weiterer Nachteil des *Regel Checker*s ist, dass sich keine der im letzten Abschnitt vorgestellten anspruchsvollen Analysen der semantischen Robustheit auf der Basis eines Theorembeweisers durchführen lassen [85].

Vor dem Hintergrund dieser Ergebnisse wurde der in dieser Arbeit entwickelte Style-Checker als Plug-In für das CASE-Tool KIEL mit den folgenden Zielen entwickelt:

- Der Style-Checker soll die Möglichkeit bieten, sowohl syntaktische, als auch Theorembeweiser basierte semantische Analysen durchführen zu können.
- Die Regelmenge des Style-Checker soll möglichst einfach erweiterbar sein.
- Es sollen sich sowohl dialektspezifizische als auch allgemeingültige Regeln spezifizieren lassen.
- Die tatsächlich auszuführenden Checks sollen sich durch den Benutzer vor jedem Start des Checkings auswählen lassen.

Fazit

Wie beschrieben, erfüllt auch keines der oben eingeführten Programme alle Anforderungen an einen Style-Checker, der die gesamte vorgestellte Taxonomie abdeckt und leicht erweiterbar ist. Basierend auf der hier behandelten Thematik zur automatisierten Fehlervermeidung in der Modellierung, wurde zusammen mit den vorgestellten Ansätzen zum Style-Checking auf Statecharts, ein flexibler Style-Checker für KIEL entwickelt. Die genauen Details der Implementierung – wie OCL verwendet wird und welche Werkzeugketten verwendet wurden – werden im nächsten Kapitel dargelegt.

5. Implementierung

Neben der Formulierung von Robustheitsregeln und der Untersuchung geeigneter Methoden, diese auf Statecharts auszuwerten, war ein Ziel dieser Diplomarbeit die Entwicklung eines Style-Checkers zur automatisierten Überprüfung der vorgestellten Statechart-Regeln (siehe Kapitel 4.2). Der entwickelte Style-Checker wurde in das Statechart-Modellierungswerkzeug KIEL integriert. Im folgenden Abschnitt wird eine kurze Einführung in den Aufbau von KIEL gegeben. Danach werden verschiedene Herangehensweisen an die Überprüfung von Design-Regeln auf Statecharts bewertet. Auf den Ergebnissen dieser Bewertung basierend, werden anschließend verschiedene Werkzeuge zur Umsetzung der gewählten Technik betrachtet. Anschließend werden die technischen Details der Implementierung, basierend auf dem gewählten Werkzeug, beschrieben. Zum Abschluss dieses Kapitels wird die Entwicklung eines XMI-Fileinterface für KIEL beschrieben, um den Style-Checker auch auf Modellen von Modellierungswerkzeugen, die XMI-Dateien exportieren, anwenden zu können.

5.1. KIEL

Das dialekt-übergreifende Statechart Modellierungswerkzeug *Kiel Integrated Environment for Layout* wird am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme der Christian-Albrechts-Universität zu Kiel zur Unterstützung bei der Entwicklung von komplexen reaktiven Systemen entwickelt [74]. Eine zentrale Komponente von KIEL ist eine Funktion zum automatischen Layout von Statecharts [45]. Damit können Statecharts, mit dem Ziel der Erhöhung der Lesbarkeit, automatisch layoutet werden. Eine weitere zentrale Funktion ist die Simulation von Statecharts. Im Gegensatz zur meist statischen Visualisierung während der Simulation wurde in KIEL eine Simulation mit dem dynamischem-Focus-und-Context Prinzip [74] umgesetzt. Damit werden während einer Simulation nur die Bereiche des simulierten Statecharts angezeigt, die aktiv sind. Andere Bereiche werden über geeignete Mechanismen ausgeblendet. Insgesamt soll so das Verständnis des in der Entwicklung befindlichen Systems erheblich vereinfacht werden.

KIEL wurde nach dem *Model-View-Controller* (MVC) Prinzip entwickelt und unterteilt sich wie in Abbildung 5.1 dargestellt in verschiedene Komponenten. Die Kommunikation der Komponenten untereinander erfolgt über wohldefinierte Schnittstellen. Daher ist der Austausch einzelner Komponenten relativ einfach möglich. Die Erweiterung von KIEL ist durch eben diese Voraussetzung gleichfalls leicht möglich. Daher wurde der in dieser Arbeit entwickelte Style-Checker als Plug-In für KIEL realisiert. Im Folgenden wird dieser auch als *Checking-Plug-In* bezeichnet.

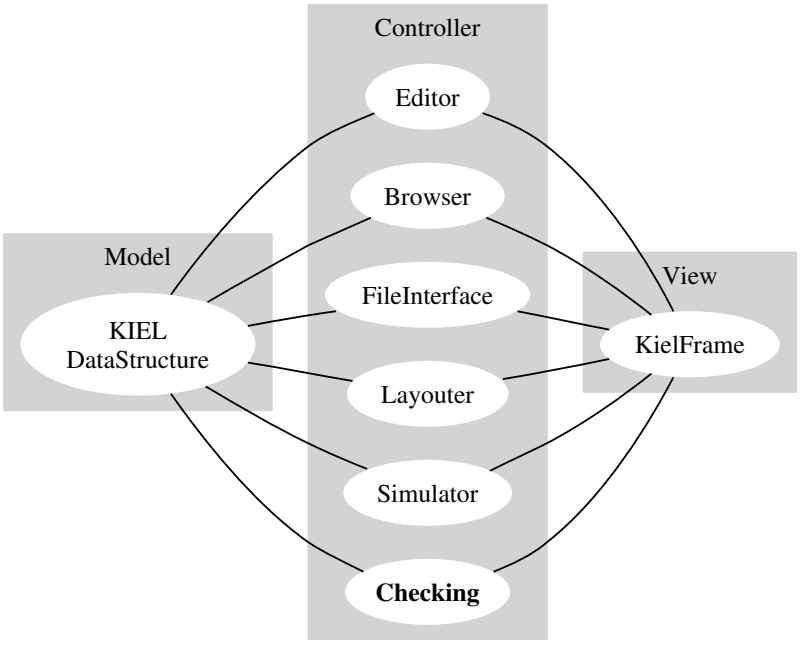


Abbildung 5.1.: Die Komponenten von KIEL nach dem MVC-Konzept geordnet

Die in Abbildung 5.1 hervorgehobene Komponente `Checking` ist im Rahmen dieser Diplomarbeit entwickelt worden. Die Komponenten `Kiel Datastructure`, `Browser` und `KielFrame` sind für die Entwicklung und Integration des Checking-Plug-In von entscheidender Bedeutung. In der Datenstruktur wird ein Statechart topologisch abgebildet. Im `Browser` wird ein Statechart visualisiert. `KielFrame` ist die grafische Oberfläche von KIEL über die der Zugriff auf die meisten Funktionen erfolgt. Der Zugriff auf den Style-Checker erfolgt über einen Menüpunkt im `KielFrame` (vergleiche Abbildung 5.6. Bevor die technischen Details der Implementierung vorgestellt werden, wird im nächsten Abschnitt eine Bewertung verschiedener Verfahren für das Style-Checking auf Statecharts vorgenommen.

5.2. Verfahren für das Style-Checking

Pap *et al.* haben vier verschiedene Verfahren zur automatisierten Überprüfung von Sicherheitskriterien auf Statecharts untersucht und bewertet [70]. Von den Autoren wurden Verfahren basierend auf der (1) Auswertung von OCL, (2) Graphtransformation, (3) problemspezifischen Programmen und (4) Erreichbarkeits-Analysen untersucht. Das im Rahmen dieser Diplomarbeit entwickelte Plug-in soll sparsam mit vorhandenen Ressourcen (Laufzeit des Programms und Speicherkapazität) umgehen und die Anforderungen aus Kapitel 4.3 erfüllen. Vor dem Hintergrund dieser Anforderungen wurden die von Pap *et al.* vorgestellten Verfahren auf Anwendbarkeit untersucht. Problemspezifische Programme und Erreichbarkeits-Analysen entsprechen nicht der Anforderung, dass das zu entwickelnde Plug-in möglichst flexibel

erweiterbar sein soll, und wurden daher für den Bereich der syntaktischen Analysen nicht weiter verfolgt.

Regeln für Graphtransformationen sind in zwei Teile unterteilt, die linke und rechte Seite. Die linke Seite einer Regel stellt die Bedingung dar, unter der eine Überführung des Ausgangsgraphen in die rechte Seite erfolgt. Die Durchführung einer Graphtransformation basiert grundsätzlich auf dem Auffinden von Übereinstimmungen der linken Seite in einem gegebenen Ausgangsgraph. Dieser Vorgang ist als *Pattern Matching* bekannt und hat eine Komplexität von $O(N^L)$, wobei N der Anzahl der Knoten im gegebenen Graphen und L der Anzahl der Knoten in der linken Seite der Regel entspricht [94]. Dies kann bei großen Statecharts, wie sie in der Industrie keine Seltenheit sind, schnell zu nicht tolerierbaren Laufzeiten führen.

Bedingt durch die Tatsache, dass das zu überprüfende in KIEL geladene Statechart nicht modifiziert werden darf, muss also für jede Regel mindestens eine weitere Kopie angelegt werden [70]. Bei Statecharts aus industriellen Projekten mit mehreren Tausend Zuständen und Transitionen stellt dies zusätzliche, nicht unerhebliche, Anforderungen an die Ressourcen (Speicherkapazität, Prozessorgeschwindigkeit) des ausführenden Computers dar und widerspricht damit dem Vorhaben, möglichst ressourcensparsam in der Ausführung zu sein.

Die Überprüfung der Regeln aus dem Katalog zur syntaktischen Robustheit wird daher aufgrund der genannten Probleme der obigen Verfahren mittels Auswertung von OCL-Ausdrücken durchgeführt. Dieses Vorgehen bietet mehrere Vorteile. Wie bereits erwähnt ist ein Vorteil der OCL, dass detaillierte Kenntnisse einer Programmiersprache nicht nötig sind, um neue Constraints zu spezifizieren. Die weiteren Vorteile sind, dass die Auswertung von OCL-Ausdrücken ressourcenschonend durchgeführt werden kann und der Regelkatalog nicht auf wenige Probleme beschränkt ist. Die Auswertung von OCL-Ausdrücken lässt sich mit zwei Techniken durchführen. Zum Einen kann ein geeigneter Interpreter verwendet und zum Anderen kann eine exekutive Auswertung durchgeführt werden. Wie einleitend erwähnt, ist die Verwendung eines Interpreters im Allgemeinen langsamer als eine exekutive Auswertung. Daher wurde in dieser Arbeit eine exekutive Auswertung der OCL umgesetzt. Die exekutive Auswertung erfolgt, unter Verwendung eines geeigneten Werkzeugs, durch eine Übersetzung der OCL-Ausdrücke in Java-Klassen. Die aus OCL generierten Java-Klassen werden kompiliert und in den entwickelten Style-Checker eingebunden. Im folgenden Abschnitt wird die Implementierung der exekutiven Auswertung von OCL beschrieben.

5.3. Style-Checking in KIEL

Für die implementierte Übersetzung von OCL-Ausdrücken in Java-Klassen wurde eine Auswahl verschiedener OCL-Werkzeuge untersucht. Die Kriterien anhand derer die Werkzeuge in der durchgeführten Untersuchung beurteilt wurden sowie die ermittelten Ergebnisse werden im Abschnitt 5.3.1 präsentiert. Unter Verwendung des gewählten *Dresden OCL Toolkit*s [19] werden, wie in Abbildung 5.2 dargestellt,

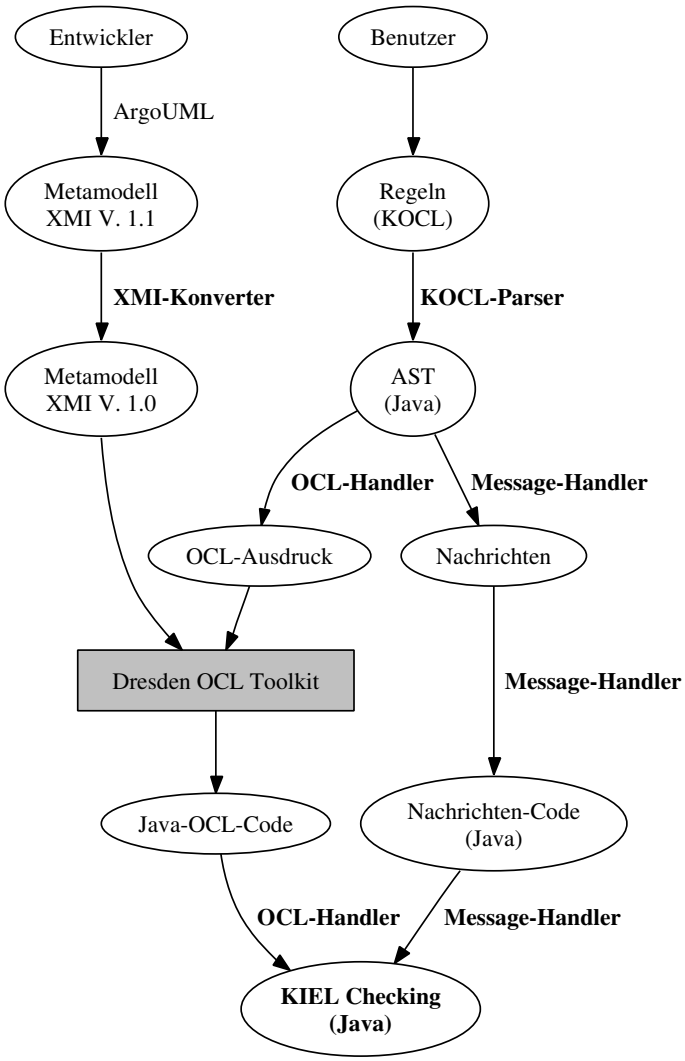


Abbildung 5.2.: Darstellung der durchgeführten Arbeitsschritte zur Übersetzung von OCL in Java-Code

OCL-Ausdrücke in entsprechenden Java-Code übersetzt.

Wie dargestellt spezifiziert der Benutzer des Style-Checkers bei Bedarf Regeln mit der eigens entwickelten Sprache *KIEL wrapped OCL* (KOCL). Diese werden mit dem entwickelten KOCL-Parser verarbeitet. Der in der KOCL-Spezifikation enthaltene OCL-Ausdruck wird vom OCL-Handler an das gewählte *Dresden OCL Toolkit* übergeben. Dieses Toolkit erzeugt aus dem OCL-Ausdruck Java-Code. Die in der KOCL-Spezifikation enthaltenen Nachrichten werden durch den Message-Handler in entsprechenden Java-Code übersetzt. Der *Regel-Generator* vereinigt die erzeugten Java-Code-Fragmente zu einer Java-Klasse.

Zur Überprüfung des OCL-Ausdrucks auf korrekte Typisierung benötigt das gewählte *Dresden OCL Toolkit* das Metamodell der Datenstruktur, auf der ein OCL-Ausdruck ausgewertet wird. Dieses Metamodell wurde im Rahmen dieser Arbeit,

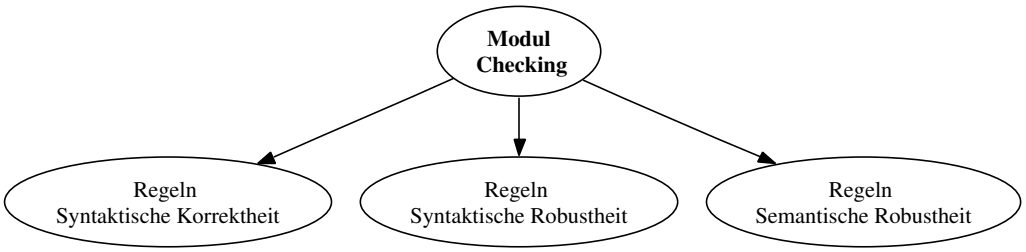


Abbildung 5.3.: Illustration der Beziehung zwischen dem Checking-Plug-In und den unterschiedlichen Regelmengen

wie dargestellt, mit *ArgoUML* erstellt und mit dem entwickelten *XMI-Konverter* für das *Dresden OCL Toolkit* vorbereitet.

Die Details der in Abbildung 5.2 dargestellten Werkzeuge werden im Abschnitt 5.3.2 beschrieben. Als erstes werden die Notwendigkeit des *XMI-Konverters* und dessen technische Details erläutert. Danach wird die Arbeitsweise des *Dresden OCL Toolkit*s beschrieben. Dieses wird von dem entwickelten *Regel-Generator* verwendet. Das Werkzeug *Regel-Generator* fasst die verschiedenen an der Verarbeitung der Regeln beteiligten Werkzeuge in einem Programm zusammen und wird im Abschnitt *KIEL wrapped OCL* beschrieben.

Wie in Abbildung 5.3 dargestellt, wurde das in dieser Arbeit implementierte Checking-Plug-In so entwickelt, dass die Auswertung von Regeln aus allen drei, in Kapitel 4 vorgestellten, Bereichen der Taxonomie möglich ist. Im Abschnitt 5.3.3 werden die technischen Details der Regelauswertung beschrieben. Abschließend werden die Integration der entwickelten Werkzeugkette in KIEL und die Erweiterungsmöglichkeiten der Regelmenge beschrieben.

5.3.1. OCL-Toolkits

Die Unterstützung von OCL in CASE-Programmen hängt von den Anforderungen der Anwender und der verfügbaren Bibliotheken ab. Bis etwa zum Jahr 2000 hatte der Großteil der Anbieter kommerzieller CASE-Programme keine OCL Unterstützung in ihre Produkte integriert [38]. In der Folgezeit wurden daher Toolkits entwickelt, um die Arbeit mit OCL zu erleichtern und den Entwicklern der CASE-Programme die Möglichkeit zu bieten, OCL-Unterstützung in ihre Programme aufzunehmen. Die entwickelten OCL-Toolkits haben bislang jedoch kaum zur Unterstützung der OCL in kommerziellen CASE-Programmen beigetragen. Für das in dieser Arbeit entwickelte Checking-Plug-In wird in diesem Abschnitt eine Evaluation ausgewählter OCL-Toolkits präsentiert. Die Kriterien anhand derer die OCL-Toolkits untersucht wurden, sind die Folgenden:

Syntaktische Analysen: Das einzusetzende Toolkit sollte Fehler in der Syntax von OCL-Ausdrücken finden können [38].

Typüberprüfung: Das Toolkit sollte die korrekte Typisierung der Ausdrücke überprüfen können [38].

OCL-Auswertung: Das Toolkit sollte Methoden bieten, um den übergebenen OCL-Ausdruck auf einer Instanz eines Modells zu überprüfen [38].

Sprache: Um die Arbeit mit dem Toolkit so einfach wie möglich zu halten, sollte das Toolkit direkt aus Java verwendet werden können. Ideal wäre es, wenn das Toolkit bereits in Java programmiert ist.

Plattform: Das KIEL-Projekt ist auf Linux- und Windows-Plattformen lauffähig. Die Einbindung eines OCL-Toolkits sollte die Plattformunabhängigkeit nicht aufheben.

Anhand dieser Kriterien wurde eine Auswahl frei verfügbarer Toolkits zur Arbeit mit OCL untersucht. Die Ergebnisse der Untersuchung sind in Tabelle 5.1 aufgelistet. Die einzelnen Toolkits werden im Folgenden kurz beschrieben. Besondere Merkmale sind entsprechend hervorgehoben.

Dresden OCL Toolkit: Das *Dresden OCL Toolkit* [19] wird an der Technischen Universität Dresden in Java entwickelt und unter der LGPL Lizenz bereitgestellt. Das Toolkit teilt sich in verschiedene Module auf, die als Bibliothek in anderen Programmen verwendet werden können. Ein für diese Arbeit wichtiger Bestandteil des Toolkits ist der OCLCompiler, der mit Hilfe eines geeigneten UML-Klassenmodells OCL-Ausdrücke auf korrekte Typisierung hin überprüft und einen Abstrakten Syntax Baum (AST) zurückliefert. Aus solch einem erzeugten AST lässt sich mit einem weiteren mitgelieferten Modul Java-Quellcode erzeugen, der einer in Java ausführbaren Version des geparsten OCL-Ausdrucks entspricht. Der Quellcode kann in anderen Programmen unter Einbindung der bereitgestellten OCL-Klassenbibliothek verwendet werden.

C#/OCL Compiler: Der *C#/OCL-Compiler* [23] ist in C# entwickelt worden. Mit Hilfe des C#/OCL-Compilers lassen sich OCL-Ausdrücke in C# Code übersetzen. Dieser erzeugte C#-Code lässt sich, wie auch der Compiler selbst, unter der Verwendung des *.NET*-Frameworks und *Visual Studio .NET 2003* kompilieren.

OSLO: Die *Open Source Library for OCL* (*OSLO*) [37] ist eine Bibliothek, die verwendet werden kann, um OCL-Unterstützung in Softwareprojekte zu integrieren. Mit Hilfe der Bibliothek kann überprüft werden, ob Instanzen von UML-Modellen die übergebenen OCL-Ausdrücke erfüllen. Die *OSLO*-Bibliothek ist in Java entwickelt worden.

PWAN: Die OCL-Bibliothek des *Project without a name* (PWAN) [77] beinhaltet einen Parser, der für syntaktisch korrekte OCL-Ausdrücke einen Syntaxbaum erzeugt. Der Parser unterstützt OCL aus Version 1.3 der UML-Spezifikation.

Tabelle 5.1.: Anforderungen an untersuchte OCL-Toolkits

Toolkit	Funktion			Sprache
	Syntaktische Analysen	Typüberprüfung	OCL Auswertung	
Dresden OCL Toolkit	•	•	•	Java
C# / OCL Compiler	-	-	-	C#
OSLO	-	-	•	Java
PWAN	•	-	-	C++

Der *C# / OCL Compiler* bietet keine Funktionen, um eingegebene OCL-Ausdrücke auf korrekte Syntax oder Typisierung hin zu überprüfen. Aufgrund der Beschränkung auf die Kombination aus *Microsoft VisualStudio .NET 2003* und *.NET*-Framework ist dieses Toolkit nicht für einen plattformübergreifenden Einsatz geeignet. Die vorliegende Version von *OSLO* wertet OCL-Ausrücke auf geladenen Modellinstanzen aus, bietet aber keine Überprüfung der Syntax und Typisierung. Die *PWAN*-OCL-Bibliothek liefert für einen korrekt formulierten OCL-Ausdruck einen AST. Eine Typüberprüfung oder sogar eine Auswertung des OCL-Ausdrucks ist jedoch nicht möglich. Die Weiterverarbeitung des AST bleibt somit dem Anwender überlassen.

Wie Tabelle 5.1 zeigt, erfüllt nur das *Dresden OCL Toolkit* alle gestellten Anforderungen. Mit Hilfe des *Dresden OCL Toolkits* lassen sich OCL-Ausdrücke auf korrekte Syntax und Typisierung hin überprüfen. Da die Implementationssprache Java ist, kann somit die Plattformunabhängigkeit weiterhin gewährleistet werden.

Weitere Kriterien, die die Auswahl beeinflusst haben, sind die Aktualität, die Einsätze in anderer Software und die Dokumentation der Bibliotheken. Die *PWAN*-Bibliothek steht auch bei diesen Kriterien hinten an. Die letzte Aktualisierung der Bibliothek war 1999 und die Dokumentation beschränkt sich auf einige Kommentare im Quellcode. Einsätze in anderen Programmen sind genau wie für den *C# / OCL Compiler* nicht bekannt. Die Internetpräsenz von *OSLO* weist *Modelware* als ein Projekt auf, in dem die Bibliothek eingesetzt wird. Die bisher verfügbare Dokumentation beschränkt sich auf Hinweise zur Bedienung einer Demo-Applikation. Jegliche benötigte Information muss dem Quellcode entnommen werden.

Die Dokumentation des *Dresden OCL Toolkits* von Finger [25] hat sich nach kurzer Einarbeitungszeit als hinreichend gut erwiesen. Daneben weist das *Dresden OCL Toolkit* bereits eine Reihe bekannter Projekte als Einsatzgebiete auf, so dass die Entscheidung auf das *Dresden OCL Toolkit* fiel. Die zur Anbindung des gewählten *Dresden OCL Toolkit*s an KIEL durchgeführten Arbeiten werden im Folgenden beschrieben.

5.3.2. Verwendung des Dresden OCL Toolkits

Im Folgenden werden die im Rahmen der Implementierung durchgeführten Arbeiten vorgestellt, um einen Überblick über die Funktionsweise des entwickelten Plug-Ins zu geben.

XMI-Konverter

Eine Voraussetzung für das verwendete OCL-Toolkit ist, dass die Typsicherheit von OCL-Ausdrücken überprüft werden kann. Für diese Funktion benötigt das gewählte *Dresden OCL Toolkit* Informationen über das Meta-Modell (Abbildung D.1), auf dem ein OCL-Ausdruck formuliert wurde. Diese Informationen können dem *Dresden OCL Toolkit* auf verschiedene Arten zur Verfügung gestellt werden. Es ist beispielsweise möglich, dass das Toolkit die benötigten Daten mittels *Reflection* [27] aus vorhandenen Java-Klassen selbstständig ausliest. Mit dieser Möglichkeit werden jedoch Einschränkungen bezüglich der verfügbaren OCL-Funktionen gemacht, die ein effektives Arbeiten behindern. Es ist unter Anderem nicht mehr möglich Mengen-Funktionen auf Assoziationen aufzurufen, da die Multiplizität einer Assoziation per *Reflection* nicht bestimmt werden kann. Ein weiterer Mechanismus, die Typinformationen an das Toolkit zu übergeben, ist, das Metamodell in einem CASE-Programm zu modellieren und als XMI-Datei zu exportieren. Wird das Metamodell als XMI-Datei bereitgestellt, stehen alle OCL-Funktionen zur Verfügung. Daher wurde das Metamodell der in KIEL verwendeten Statechart Datenstruktur mit *ArgoUML* [4] modelliert und als XMI-Datei exportiert. Ein Nebeneffekt dieser Vorgehensweise ist, dass Änderungen an der Datenstruktur auch in das Metamodell eingepflegt werden müssen. Der Arbeitsaufwand ist also etwas höher, als wenn man die Meta-Daten mittels *Reflection* auslesen würde.

Obwohl bereits Version 2.0 des gewählten *Dresden OCL Toolkit* existiert, wurde für diese Implementierung die Version 1.3 verwendet, da die neue Version noch keinen Java-Codegenerator bietet. Die ältere Version des Toolkits setzt als Metamodell eine mit *ArgoUML* Version 0.0.7 erzeugte XMI-Datei voraus. Diese Version von *ArgoUML* ist jedoch nicht mehr verfügbar. Der XMI-Export der Version 0.20 von *ArgoUML* führt den Export mit XMI Version 1.1 durch, die von der gewählten Version des Toolkits nicht verarbeitet werden kann. Um dieses Problem zu lösen, wurde ein Programm entwickelt, das eine Transformation der XMI-Dateien, die durch die neue *ArgoUML* Version erzeugt wurden, in XMI Version 1.0 automatisiert durchführt.

Der hierzu entwickelte *XMI-Konverter* (Anhang G.1) wurde als Kommandozeilenprogramm entworfen (siehe Auflistung 5.1). Als Eingabe erwartet das Programm den Dateinamen des Quell-Dokuments und den des Ausgabe-Dokuments. Für die Transformation wird das *Extended Markup Language* (XML)-Dokument mit Methoden aus der *Simple API for XMI* (SAX)-Bibliothek sequentiell durchlaufen. Über die generierten *Call-back*-Ereignisse des XMI-Parsers wird mit Methoden des *Document Object Model* (DOM) ein neues Dokument erzeugt. Die Arbeitsweise des *XMI-Konverter*s ist in Abbildung 5.4 dargestellt.

Auflistung 5.1: Kommandozeilenausgabe des XMI-Konverters

```
KIEL xmi-file converter.
Used for converting xmi-files generated by argoUML v. 0.2.x
into the format readable by the Dresden OCL Toolkit v. 1.3.
Usage: xmiConverter <Inputfile> <Outputfile>
```

Abbildung 5.4.: Übersicht über die Funktionsweise des XMI-Konverters

Code-Generator

Das *Dresden OCL Toolkit* bietet, wie erwähnt, die Möglichkeit OCL-Ausdrücke in ausführbaren Java-Code zu übersetzen. Der im Toolkit enthaltene Parser erzeugt aus einem übergebenen OCL-Ausdruck einen *OCLTree* [25]. Der Typchecker des *Dresden OCL Toolkits* überprüft anschließend anhand des Metamodells auf dem OCLTree, ob der eingelesene OCL-Ausdruck korrekt typisiert ist. Ist der Ausdruck korrekt typisiert, kann der OCLTree an den Java-Codegenerator übergeben werden. Der Codegenerator erzeugt, unter Anwendung des *Visitor-Design-Pattern*s [30], aus dem OCLTree den Java-Code [25]. Der erzeugte Java-Code lässt sich vom Codegenerator abfragen und steht zur weiteren Verwendung bereit. Abbildung 5.5 visualisiert die an der Übersetzung beteiligten Module des *Dresden OCL Toolkit*s.

Hauptbestandteil des erzeugten Java-Codes ist die im Toolkit vorhandene Java-Implementierung der OCL-Bibliothek [24]. Die OCL-Bibliothek wird benötigt, um Methoden der Sprache in Java ausführen zu können und ist Kapselung für die zugrunde liegenden Klassen des Metamodells. Damit der erzeugte Java-Code ausgeführt werden kann, muss die OCL-Bibliothek also in das Programm aufgenommen werden, das den erzeugten Java-Code ausführt. Der OCL-Ausdruck in Auflistung 5.2 ist in den Java-Code in Auflistung 5.3 übersetzt worden.

Auflistung 5.2 zeigt den an das *Dresden OCL Toolkit* übergebene OCL-Ausdruck für die bereits in Abschnitt 4.2 beschriebene *Well-formedness*-Regel CompositeState-1 [66]. In Zeile eins ist – ausgezeichnet durch das Schlüsselwort `context` – die Klasse angegeben, die den nachfolgenden Ausdruck erfüllen muss. In diesem Fall soll der Ausdruck auf allen Instanzen der Klasse `CompositeState` ausgewertet werden. Diese Klasse repräsentiert hierarchische Zustände. In diesem Beispiel ist eine Invariante – ausgezeichnet durch das Schlüsselwort `inv` – spezifiziert. Der Bezeichner `self` dient zur Referenzierung des Kontextes, ähnlich wie der Bezeichner `this` in Java. Das Attribut `subnodes` der referenzierten Klasse enthält alle Zustände die in einem hierarchischen Zustand enthalten sind. Der Pfeil-Operator ist ein Operator der OCL mit dem nachfolgende Funktionen auf einer Menge ausgeführt werden. In diesem Fall wird – ausgezeichnet durch `select` – eine Teilmenge gebildet, die alle initialen Zustände enthält. Abschließend wird der Vergleich durchgeführt, ob die ermittelte

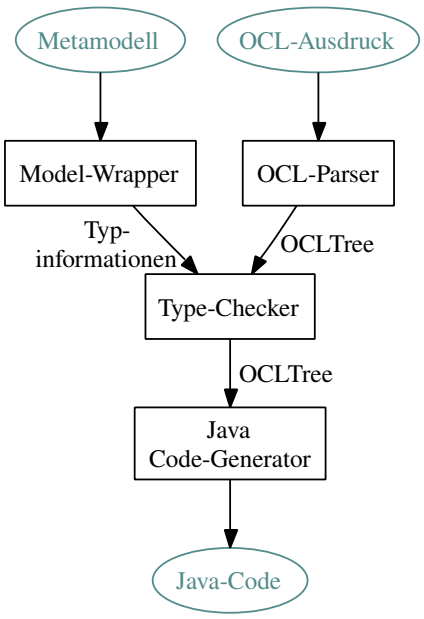


Abbildung 5.5.: An der OCL-Transformation beteiligte Module des *Dresden OCL Toolkit*s

Auflistung (5.2) OCL-Ausdruck der *Well-formedness*-Regel CompositeState-1

```
1 context CompositeState;
2 inv: self.subnodes->select (v |v.oclIsTypeOf(InitialState))->size <= 1
```

Auflistung (5.3) Erzeugter Java-Code

```
1  final OclAnyImpl tudOclNode0=Ocl.toOclAnyImpl( Ocl.getFor(o) );
2  final OclSequence tudOclNode1=Ocl.toOclSequence(
3    tudOclNode0.getFeature("subnodes"));
4
5  final OclIterator tudOclIter0=tudOclNode1.getIterator();
6  final OclBooleanEvaluatable tudOclEval0=new OclBooleanEvaluatable() {
7    public OclBoolean evaluate() {
8      final OclType tudOclNode2=OclType.getOclTypeFor(o, "InitialState");
9      final OclBoolean tudOclNode3=Ocl.toOclAnyImpl(tudOclIter0.getValue()).
           oclIsTypeOf(tudOclNode2);
10     return tudOclNode3;
11   }
12 };
13
14 final OclSequence tudOclNode4=Ocl.toOclSequence(tudOclNode1.select(tudOclIter0,
       tudOclEval0));
15
16 final OclInteger tudOclNode5=tudOclNode4.size();
17 final OclInteger tudOclNode6=new OclInteger(1);
18 final OclBoolean tudOclNode7=tudOclNode5.isLessEqual(tudOclNode6);
```

Teilmenge höchstens einen initialen Zustand enthält.

Der mit dem Java-Codegenerator erzeugte Java-Code ist in Auflistung 5.3 enthalten. In Zeile eins wird die Instanz des zu überprüfenden Objekts von der OCL-Basisbibliothek des Toolkits gekapselt. Alle weiteren Zugriffe auf das Objekt erfolgen über die Kapselung. In Zeile zwei wird die Menge `subnodes` in einer Variable vom Typ `OclSequence` gekapselt. In den Zeilen sechs bis zwölf wird die Filterbedingung des eingegebenen OCL Ausdrucks als Java-Funktion deklariert. Dieser Filter wird in Zeile zwölf auf der Kapselung der Menge `subnodes` angewendet. In den Zeilen 16 bis 18 wird verglichen, ob die Anzahl der selektierten initialen Zustände kleiner oder gleich eins ist.

Kiel Wrapped OCL

Damit das Ergebnis aussagekräftig an den Benutzer des entwickelten Checking-Plug-in zurückgeliefert werden kann, wird ein geeigneter Hinweistext benötigt. Dieser Hinweistext lässt sich nicht aus dem OCL-Ausdruck herauslesen und auch nicht mit ihm spezifizieren. Damit das entwickelte Plug-in dennoch entsprechende Meldungen zurückliefern kann, müssen diese mit dem OCL-Ausdruck in geeigneter Weise kombiniert werden.

Zu diesem Zweck wurde die Spezifikationssprache KOCL entwickelt. Mit dieser Sprache werden Hinweismeldungen und OCL-Ausdrücke kombiniert. Damit ist es möglich mehrere Hinweistexte zu deklarieren und je nach Context eine unterschiedliche Meldung zurückzuliefern. In Auflistung 5.4 ist der OCL-Ausdruck aus Auflistung 5.2 in KOCL mit einer zugehörigen aussagekräftigen Meldung präsentiert. In den Zeilen zwei bis vier ist die Nachrichten Deklaration – eingeleitet durch den Identifier `declarations` – angegeben. Die Regel ist in den Zeilen fünf bis acht – ausgezeichnet durch den Identifier `constraint` – spezifiziert. Zeile sieben enthält den OCL Ausdruck – angepasst an KIELs Metamodell. In Zeilen neun bis elf ist – ausgezeichnet durch den Identifier `fails` – die Deklaration enthalten, welche Nachricht zurückgeliefert wird, wenn das Constraint nicht erfüllt ist.

Die Verarbeitung der KOCL-Dateien geschieht durch einen mit dem Parser Gene-

Auflistung 5.4: Ein KOCL-Beispiel für das KIEL-Metamodell

```
1  rule UML13CompositeStateRule1 {
2      declarations {
3        message "A composite state can have at most one initial vertex.";
4      }
5      constraint {
6        context ORState or Region;
7        "self.subnodes->select (v | v.oclIsTypeOf(InitialState))->size <= 1";
8      }
9      fails {
10       message;
11     }
12 }
```

Auflistung 5.5: Kommandozeilenausgabe des *RuleParser*

```
KIEL Rule to Java Converter
Usage: RuleParser <ModelFile> <RuleDir> <Base Output Directory> <BasePackage>
```

rator SableCC [29] erzeugten Parser. Die EBNF der Grammatik ist in Abbildung C.1 angegeben. Die Eingabe für den SableCC befindet sich im Anhang (Abschnhitt E.1).

Basierend auf dem *Visitor-Design-Pattern* wird aus KOCL-Spezifikationen pro Regel (`rule`) eine Java-Datei erzeugt. Damit die Verwendung der erzeugten Java-Dateien im Vorgang der Auswertung einheitlich durchgeführt werden kann, wurde eine gemeinsame Basisklasse (`BaseCheck`, Anhang G.3) für alle Regeln implementiert. Die Basisklasse bietet alle wichtigen Funktionen, die zur reibungslosen Integration in das Checking-Plug-In benötigt werden. Von der Basisklasse werden die übersetzten KOCL-Spezifikationen abgeleitet. Die Transformation wird durch den *RuleParser* (Anhang G.2) durchgeführt. Er ist, wie der *XMI-Konverter*, als Kommandozeilenprogramm ausgelegt und in den *Build*-Prozess von KIEL integriert. Auflistung 5.5 zeigt die Ausgabe eines Aufrufs ohne Parameter.

5.3.3. Arbeitsweise des Style-Checkers

Im vorigen Abschnitt wurde die Verwendung des *Dresden OCL Toolkit*s beschrieben. Die erzeugten Java-Klassen kommen durch den Style-Checker zur Anwendung auf Statecharts. Die Arbeitsweise des entwickelten Style-Checkers und die Integration in KIEL wird im Folgenden beschrieben.

Regelanwendung

Die Anwendung der Regeln auf ein Statechart erfolgt über den *Call-back*-Mechanismus des *Visitor-Design-Patterns*. Das zu überprüfende Statechart ist in der Statechart Datenstruktur von KIEL baumartig abgebildet. Ein Knoten eines Baums ist ein Zustand des Statecharts. Transitionen sind Zuständen zugeordnet. Dieser Baum wird mittels Tiefensuche traversiert. Für jeden besuchten Zustand wird überprüft, ob eine Regel basierend auf dem `context` anwendbar ist. Ist der Typ des Zustands gültig, wird die Bedingung ausgewertet und im Bedarfsfall eine definierte Nachricht zurückgeliefert.

Die flexible Arbeitsweise der Regelauswertung ermöglicht es, verschiedene Arten von Regeln auszuwerten. Dies ist insbesondere hilfreich, da die Auswertung der Regeln semantischer Robustheit, wie oben erwähnt, einen externen Theorembeweiser benötigt [85]. Auch diese Regeln sind von der gemeinsamen Basisklasse abgeleitet. Die erste Anforderung an das entwickelte Checking-Plug-in – sowohl syntaktische als auch Theorembeweiser-basierte semantische Analysen sollen durchgeführt werden können (vergleiche Kapitel 4.3) – wurde also erfüllt. Die in dieser Arbeit implementierten Regeln und die Ergebnisse eine Laufzeitanalyse werden in Kapitel 6 präsentiert.

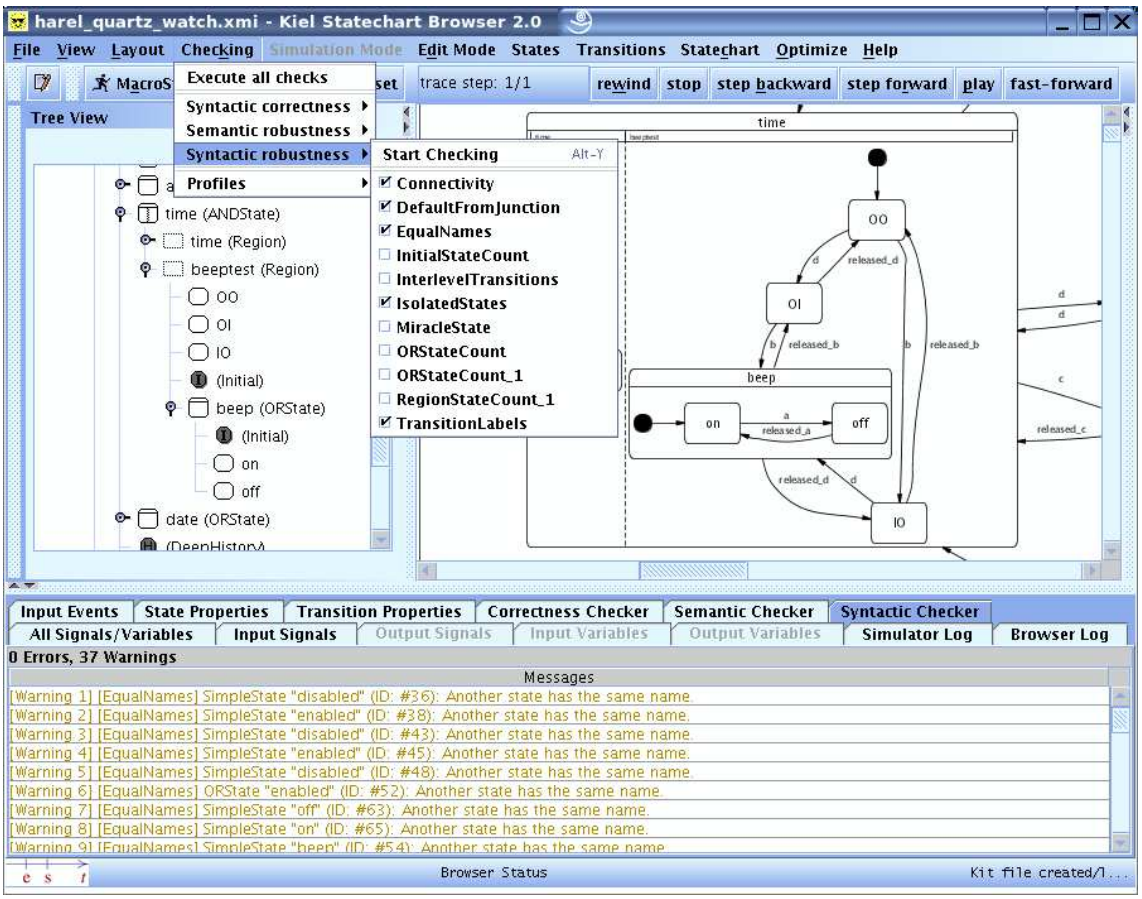
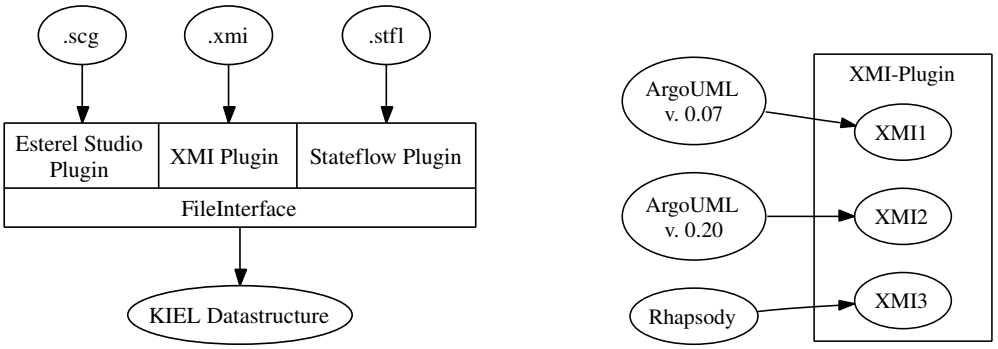


Abbildung 5.6.: Screenshot von KIEL mit ausgeklappten Checking-Menüpunkt

Integration in KIEL

Eine Erweiterung der Regelmenge erfolgt durch das Hinzufügen einer neuen `rule`-Deklaration in die Regeldateien mit KOCL. Die implementierte Werkzeugkette wurde so in den *Build*-Prozess von KIEL integriert, dass keine Änderungen benötigt werden, wenn neue Regeln an den bisher vorgesehenen Stellen im Dateisystem hinzugefügt werden. Regeln, die sich nicht in KOCL formulieren lassen, wie etwa solche die externe Programme ansteuern, werden von der Basisklasse abgeleitet und an der entsprechenden Stelle im Dateisystem des KIEL-Projektes abgelegt. Diese Regeln werden ebenfalls automatisch im *Build*-Prozess übersetzt und sind beim nächsten Start von KIEL verfügbar. Das Style-Checker-Plug-In bestimmt bei jedem Start von KIEL welche Regeln zur Verfügung stehen und stellt entsprechende Menüpunkte zur Aktivierung automatisch bereit. Der Benutzer kann über diese erzeugten Menüpunkte vor jedem Checking-Vorgang bestimmen, welche Regeln ausgewertet werden sollen (siehe Abbildung 5.6).

Wird ein Statechart in KIEL importiert, verarbeitet das Checking-Plug-In außerdem ein Regel-Profil, passend zum Dialekt in dem das Statechart entworfen wurde. In solch einem Profil werden all die Regeln angegeben, die standardmä-



(a) Generelle Arbeitsweise des Fileinterfaces.

(b) Modulsicht des XMI-Fileinterface.

Abbildung 5.7.: Das *Fileinterface*-Modul

ßig auf Statecharts eines Dialekts ausgewertet werden sollen. So kann auch für die Kommandozeilen-Variante von KIEL angegeben werden, welche Regeln ausgewertet werden sollen und welche nicht. Ist kein Profil vorhanden werden standardmäßig alle Regeln ausgewertet. Die bisher definierten Profile sind im Anhang B enthalten.

5.4. XMI-Import

KIEL bietet über die Komponente `FileInterface` bislang unter anderem Importfunktionen für Statecharts, die mit den Modellierungswerkzeugen *Esterel Studio*, *Matlab Simulink/Stateflow* und mit KIEL selbst entworfen wurden. Das File-Interface von KIEL ist, wie in Abbildung 5.7a dargestellt, modular konzipiert. Für jedes Dateiformat, das in KIEL verarbeitet werden soll, wird ein Plug-In entwickelt. Eine zu öffnende Datei wird an das entsprechende zum Dateiformat gehörende Plug-In übergeben und dieses liefert eine Instanz eines Statecharts in der Datenstruktur zurück [47, 73]. Damit der Einsatz des Checking-Plug-Ins nicht auf Statecharts aus diesen Werkzeugen beschränkt bleibt, und um der zunehmenden Verbreitung der UML gerecht zu werden, ist das File-Interface um ein Modul zum Im- und Export von Statecharts aus und in das XMI-Format erweitert worden.

Bei der Entwicklung des XMI-File-Interface (Anhang G.4) musste besonderes Augenmerk auf die Möglichkeit gelegt werden, dass sich die Namen der Tags des XMI-Formats sowohl zwischen einzelnen XMI- und UML-Versionen, als auch zwischen Modellierungswerkzeugen unterscheiden können. Informationen darüber, in welcher XMI- und UML-Version und von welchem Programm eine zu importierende XMI-Datei erzeugt wurde, sind in den *META-Tag*-Informationen am Anfang einer XMI-Datei enthalten.

Um dem Umstand verschiedener Versionen gerecht zu werden und möglichst flexibel auf Änderungen reagieren zu können, wurde das entwickelte XMI-Fileinterface modular konzipiert. Für jede Variation des XMI Dateiformats wird ein eigenes Mo-

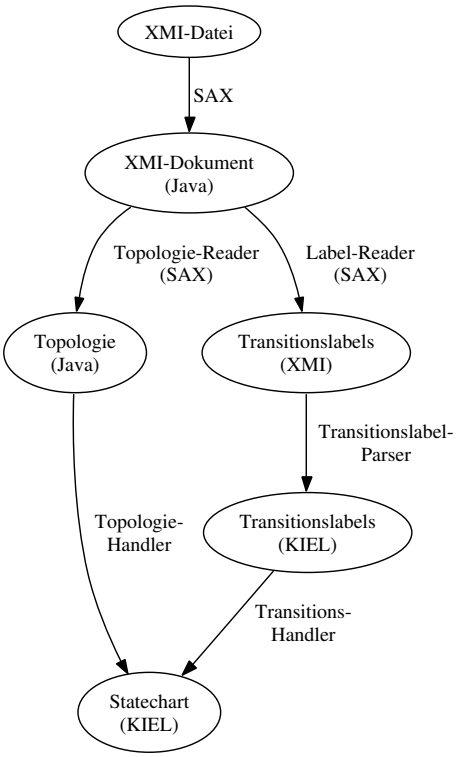


Abbildung 5.8.: Visualisierung der Zwischenschritte des implementierten Reader-Moduls

dul, das Statecharts aus den zu importierenden Dateien in die Datenstruktur von KIEL einliest, angelegt. Die Verwaltung der einzelnen Module geschieht durch das XMI-Plug-In (Vergleiche Abbildung 5.7b).

Der Import einer XMI-Datei erfolgt grundsätzlich in einem mehrteiligen Prozess. Im ersten Schritt überprüft ein SAX-Parser, ob es sich bei der ihm übergebenen Datei um eine syntaktisch gültige XMI-Datei handelt. Ist dies der Fall, wird im nächsten Schritt anhand der *META-Tag*-Informationen der übergebenen Datei bestimmt, welches *Reader*-Modul die *State Machine* in die KIEL-Datenstruktur einliest. Ist kein Modul im XMI-File-Interface für das Format der XMI-Datei geladen, wird der Vorgang abgebrochen.

Bisher wurde ein *Reader*-Modul implementiert, das mit *ArgoUML* erzeugte XMI-Dateien verarbeitet (Anhang G.4). Dieses Modul liest das erste in der zu importierenden Datei gefundene Statechart in die topologische Statechart Datenstruktur von KIEL ein. Das Einlesen erfolgt mittels SAX-Methoden. Die Topologie des Statecharts wird ohne Veränderungen in die Statechart Datenstruktur überführt, Beschriftungen von Transitionen werden jedoch gesondert behandelt. Diese werden mit einem Transitionslabel-Parser (Anhang G.4) in die einzelnen Bestandteile einer Beschriftung zerlegt und in die dafür vorgesehenen Klassen in der Datenstruktur überführt. Mit den so erzeugten Instanzen von Transitionslabels lässt sich eine Simulation eines importierten Statecharts in KIEL durchführen. Der Transitionslabel-

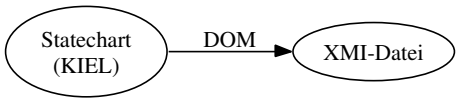


Abbildung 5.9.: Arbeitsweise des XMI-Exports

Parser wurde mit *SableCC* erzeugt. Als Eingabe für *SableCC* dient eine an die Java-Syntax angelehnte Grammatik (Anhang E.2) von Etienne Gagnon [29]. In Abbildung 5.8 ist die Reihenfolge der Arbeitsschritte zur Übersicht visualisiert worden.

Der Export von Statecharts erfolgt ebenfalls über das File-Interface. Die Module zum Import von Statecharts enthalten die Routinen, die den Export eines Statecharts in das gewählte Dateiformat vornehmen. Statecharts sind in der UML Teil einer Klasse oder Methode. Der Export erzeugt daher ein generisches UML-Modell und legt in diesem Modell eine Klasse an. Dieser erzeugten Klasse wird das zu exportierende Statechart zugeordnet. Die XMI-Datei wird mit Methoden des DOM angelegt. Die Arbeitsweise des XMI-Exports (Anhang G.4) ist in Abbildung 5.9 dargestellt.

Fazit

Alle in Kapitel 4.3 formulierten Anforderungen an das Checking-Plug-in wurden erfüllt. Es ist leicht erweiterbar, flexibel in der Anwendung und bietet Möglichkeiten zur Überprüfung von Regeln die sowohl mit OCL formuliert, als auch direkt in Java implementiert wurden. Durch die Erweiterung um das XMI-Fileinterface ist zudem der Zugriff auf weitere Statecharts ermöglicht worden. Die in KOCL formulierten Regeln aus Kapitel 4.2 sind im Anhang (Abschnitt F) enthalten. Im folgenden Kapitel werden die Ergebnisse einer *Usability*-Analyse vorgestellt.

6. Ergebnisse

Zur Beurteilung des entwickelten Checking-Plug-ins wurden verschiedene Untersuchungen vorgenommen. Als erstes wurden die in Kapitel 4.2 vorgestellten Regeln umgesetzt, um den Nutzen von *KOCL* zu beurteilen. Anschließend wurden die erstellten Regeln zur Beurteilung der Geschwindigkeit des Checking-Plug-ins auf verschiedene Statecharts angewendet. Zum Abschluss wurde der entwickelte Style-Checker auf Beispiele aus der Industrie angewendet. Im Folgenden werden die gesammelten Ergebnisse präsentiert.

6.1. KOCL Usability-Analyse

Die *Well-formedness*-Regeln sind ein bekanntes Beispiel für die Verwendung von OCL. Daher wurden als erste Anwendung 26 der *Well-formedness*-Regeln mit *KOCL* spezifiziert. Einige in der UML formulierten *Well-formedness*-Regeln wurden nicht umgesetzt, da diese Regeln Eigenschaften von UML-Modellen selbst behandeln und daher keine Relevanz für Statecharts haben. Tabelle 6.1 listet alle umgesetzten *Well-formedness*-Regeln und die jeweils benötigte Zeit für deren Umsetzung auf. Durchschnittlich werden zur Spezifikation einer Regel in *KOCL* zwei Minuten benötigt. Die benötigte Zeit hängt von der Anzahl der getippten Zeichen ab. Man kann also nachvollziehen, dass es also schneller ist mit *KOCL* zu arbeiten, als eine Regel in Java zu programmieren. Die vorgestellten Regeln zur syntaktischen Robustheit wurden ebenfalls mit *KOCL* umgesetzt. Die Spezifikation einer solchen Regel hat bis zu elf Minuten gedauert (vergleiche Tabelle 6.2). Die Differenzen bezüglich der Dauer hängen direkt von der jeweiligen Komplexität des Problems ab.

Tabelle 6.1.: Übersicht der umgesetzten *Well-formedness*-Regeln der UML 1.3 und 2.0; aufgelistet sind die Hinweise und Namen aus der UML und die zur Spezifikation in *KOCL* benötigte Zeit

Regel	UML 1.3	UML 2.0	Zeit (min:sec)
A composite state can have at most one initial vertex.	CompositeState 1	Region 1	2:15
A composite state can have at most one deep history vertex.	CompositeState 2	Region 2	1:59
A composite state can have at most one shallow history vertex.	CompositeState 3	Region 3	2:01

Fortsetzung auf der nächsten Seite

Regel	UML 1.3	UML 2.0	Zeit (min:sec)
There have to be at least two composite substates in a concurrent composite state.	CompositeState 4	State 1	2:02
A concurrent state can only have composite states as substates.	Composite State 5	-	1:51
The substates of a composite state are part of only that composite state.	Composite State 6	-	1:45
A final state cannot have any outgoing transitions.	FinalState 1	FinalState 1	1:40
An initial vertex can have at most one outgoing transition and no incoming transition.	PseudoState 1	-	2:08
An initial vertex can have at most one outgoing transition.	-	PseudoState 1	1:59
History vertices can have at most one outgoing transition.	PseudoState 2	PseudoState 2	1:58
A join vertex must have at least two incoming transitions and exactly one outgoing transition.	PseudoState 3	PseudoState 3	2:02
A fork vertex must have at least two outgoing transitions and exactly one incoming transition.	PseudoState 4	PseudoState 5	2:01
A junction vertex must have at least one incoming and one outgoing transition.	PseudoState 5	PseudoState 7	2:00
A choice vertex must have at least one incoming and one outgoing transition.	PseudoState 6	PseudoState 8	1:55
A top state is always a composite.	StateMachine 2	-	2:15
A top state cannot have any containing states.	StateMachine 3	-	2:07
The top state cannot be the source of a transition	Statemachine 4	-	2:07
The value of a bound value must be a positive integer, or unlimited.	SynchState 1	-	1:11
All incoming transitions to a SynchState must come from the same region and all outgoing transitions from a SynchState must go to the same region.	SynchState 2	-	2:21
A fork segment should not have guards or triggers.	Transition 1	Transition 1	2:23
A join segment should not have guards or triggers.	Transition 2	Transition 2	1:56
A fork segment should always target a state.	Transition 3	Transition 3	1:53
A join segment must always originate from a state.	Transition 4	Transition 4	1:53
Transitions outgoing pseudostates may not have a trigger.	Transition 5	Transition 5	2:13
Join segments should originate from orthogonal states.	Transition 6	-	2:05
Fork segments should target orthogonal states.	Transition 7	-	2:07

Tabelle 6.2.: Umgesetzte Regeln der syntaktischen Robustheit mit Hinweistext und benötigter Zeit zur Spezifikation

Regel	Hinweis	Zeit (min:sec)
InitialStateCount	An ORState or a Region can have at most one initial vertex.	2:45
ORStateCount	An Orstate should contain at least one state.	2:17
RegionStateCount	A region should contain at least one state.	2:21
TransitionLabels	The transition has no label or no trigger.	4:01
IsolatedStates	The state is isolated.	2:36
InterlevelTransitions	The source and the target of the transition have different parent states.	4:46
EqualNames	All states should have different names.	10:45
DefaultFromJunction	There should be a default transition from each connective junction.	6:37
Connectivity	There is no path from an initial state to this state.	1:45

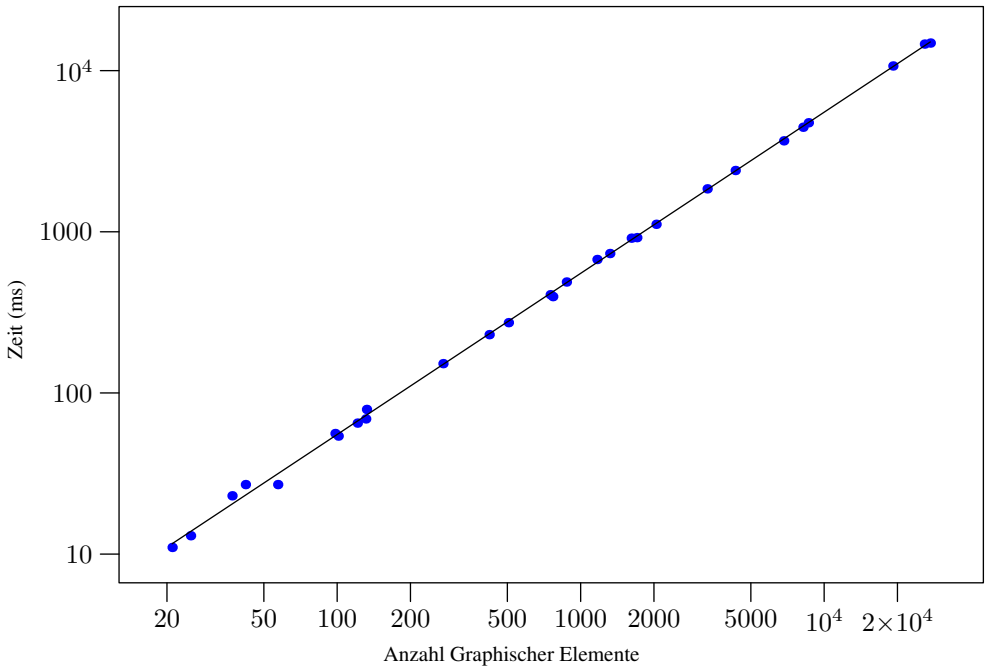
6.2. Laufzeitanalyse

Damit ein Style-Checker von Entwicklern in der täglichen Arbeit akzeptiert wird, muss solch ein Programm Hinweise auf mögliche Problemstellen im Modell ausreichend schnell liefern. Zur Beurteilung der Geschwindigkeit des entwickelten Checking-Plug-Ins und der Regeln wurde daher eine Laufzeitanalyse für Beispiele unterschiedlicher Größe durchgeführt. Die zur Laufzeitanalyse verwendeten Beispiele stammen aus der *Estbench* (Esterel benchmark) Sammlung [15] und der Columbia Esterel Compiler Distribution [20]. Die gewählten Beispiele aus beiden Sammlungen wurden mit KIELs integrierter Esterel Transformation [76] in equivalente SSMs überführt, bevor das Checking-Plug-In angewendet wurde. Die Analyse wurde auf den nicht-optimierten Varianten der importierten Statecharts ausgeführt, d.h. sie enthalten eine große Anzahl an graphischen Elementen (Zustände, Pseudozustände, Transitionen, etc.) – perfekt für eine quantitative Analyse der Laufzeit. Tabelle 6.3 und Abbildung 6.1 zeigen die Ergebnisse der Zeitmessungen. Die Zeiten wurden auf einem PC mit Linux OS, einem 2,6 GHz AMD Athlon 64 Prozessor und 2 GB RAM gemessen.

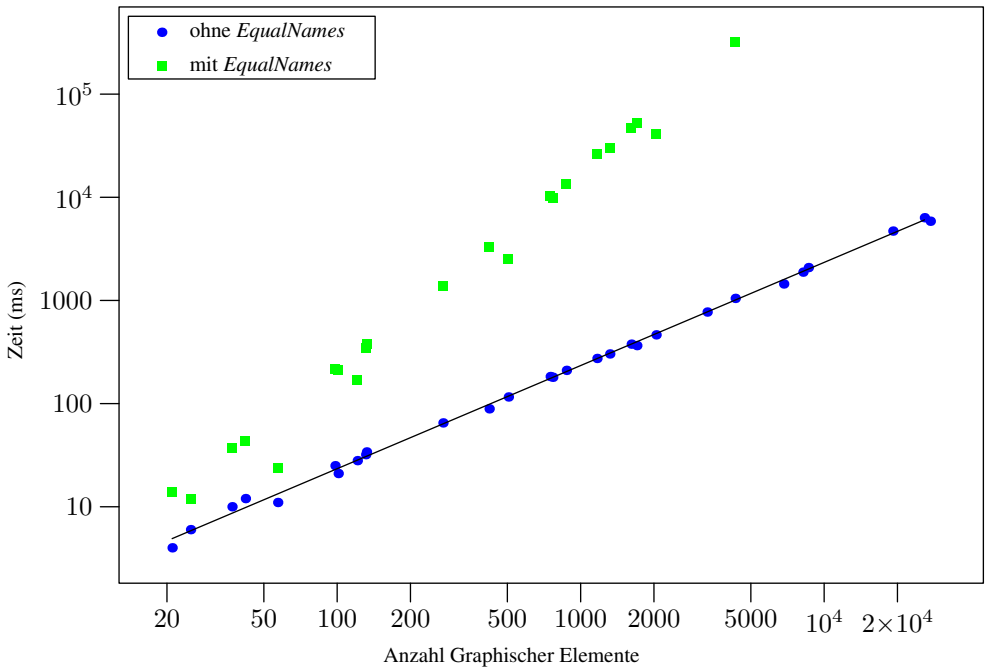
Die *Well-formedness*-Regeln überprüfen, wie erwähnt, ob das untersuchte Statechart syntaktisch korrekt gemäß der UML Spezifikation ist. Bis auf das Beispiel STOPWATCH sind alle untersuchten Statecharts gemäß den ausgewerteten Regeln syntaktisch korrekt. Die für die Regeln der syntaktischen Korrektheit ermittelten Laufzeiten spiegeln also einzig die Auswertung der Regelmenge auf dem untersuchten Statechart wieder, da die Nachrichtenerstellung einen deutlich messbaren Anteil an der Laufzeit hätte [85, Seite 62]. Die Laufzeit für die syntaktischen Analysen skaliert, wie in Abbildung 6.1a dargestellt, direkt mit der Anzahl der Graphischen Elemente eines Charts. Die gesammelten Daten lassen den Schluss zu, dass die Laufzeit der Analysen der syntaktischen Korrektheit in $O(n)$ liegt, mit n der Anzahl graphischer

Tabelle 6.3.: Ergebnisse experimenteller Laufzeitanalysen der syntaktischen Korrektheit und Robustheits Regeln; angegeben sind die Anzahl der graphischen Elemente, die Laufzeiten und die Anzahl der erzeugten Warnungen

Modell	Graphische Elemente	Syntaktische Korrektheit	Syntaktische Robustheit			
			ohne *EqualNames*		mit *EqualNames*	
		Zeit (ms)	Zeit (ms)	Warnungen	Zeit (ms)	Warnungen
ABRO	37	23	10	4	37	11
STOPWATCH	57	27	11	3	24	6
SCHIZOPHRENIA	42	27	12	8	43	13
JACKY1	98	56	25	18	216	45
RUNNER	131	69	32	16	343	32
REINCARANATION	132	79	34	16	381	34
ABCD	750	407	183	134	10387	254
GREYCOUNTER	768	396	180	99	9925	211
WW	1167	673	274	181	26005	355
MEJIA	1317	733	303	169	30325	367
TCINT	1614	913	377	206	46587	482
ATDS-100	3308	1847	772	432	-	967
MCA200	19156	10690	4705	2954	-	-
BINTREE-2	25	13	6	0	12	6
BINTREE-4	121	65	28	0	168	30
BINTREE-6	505	273	116	0	2509	126
BINTREE-8	2041	1113	464	0	40988	510
BINTREE-10	8185	4455	1882	0	-	2046
QUADTREE-1	21	11	4	1	14	8
QUADTREE-2	101	54	21	5	210	40
QUADTREE-3	421	230	89	21	3296	168
QUADTREE-4	1701	920	365	85	52650	680
QUADTREE-5	6821	3668	1445	341	-	2728
QUADTREE-6	27285	14833	5860	1381	-	-
TOKENRING-3	272	152	65	30	1389	68
TOKENRING-10	874	488	210	93	13341	215
TOKENRING-50	4314	2401	1047	453	317614	1055
TOKENRING-100	8614	4748	2083	903	-	-
TOKENRING-300	25814	14618	6338	2703	-	-



(a) Laufzeiten der Analysen der syntaktischen Korrektheit



(b) Laufzeiten der syntaktischen Robustheitsanalysen

Abbildung 6.1.: Graphische Darstellung der Laufzeiten des entwickelten Style-Checkers im Bezug zur Anzahl der Elemente untersuchter Statecharts

Elemente (Zustände, Pseudozustände, Transitionen, etc.) des untersuchten Statecharts.

Im Gegensatz zur Auswertung der syntaktischen Korrektheitsregeln liefern die syntaktischen Robustheitsregeln für jedes untersuchte Statechart Nachrichten zurück, d. h. es werden Verletzungen der vorgestellten Regeln in den betrachteten Statecharts gefunden. Viele der Nachrichten werden von den Regeln *ORStateCount* und *TransitionenLabels* generiert. Die Laufzeiten für die Robustheitsregeln wurde in zwei Kategorien aufgeteilt. Zum Einen, die Anwendung des Checking-Plug-Ins mit allen Regeln, und zum Anderen die Anwendung des Plug-Ins ohne die Regel *EqualNames*. Wie in Abbildung 6.1b dargestellt, liegt die Laufzeit der syntaktischen Robustheitsanalysen ohne *EqualNames* ebenfalls in $O(n)$, mit n wie oben.

Durch die technischen Beschränkungen der OCL ermittelt die Regel *EqualNames* für jeden einzelnen Zustand eines Statecharts, ob die Menge aller Zustände des Charts mindestens einen anderen Zustand mit gleichem Namen enthält. Sei n die Anzahl der Zustände eines untersuchten Charts, dann führt die Anwendung der Regel zu $n * n$ Vergleichen. Die Laufzeit für die Regel *EqualNames* liegt also in $O(n^2)$. Aufgrund dieser Tatsache wurden die Laufzeiten auf den für diese Laufzeitanalyse verwendeten Statecharts MCA200, BINTREE-10, QUADTREE-5 und -6 sowie TOKENRING-100 und -300 für die Regel *EqualNames* nicht gemessen, da diese jeweils mehrere tausend Zustände enthalten (vergleiche Tabelle 6.3).

Es konnte beobachtet werden, dass die Laufzeiten der syntaktischen Robustheitsanalyse mit *Equalnames* auf den Beispielen STOPWATCH und den Statecharts der BINTREE-Reihe kürzer sind, als auf anderen Statecharts mit vergleichbarer Anzahl an graphischen Elementen. Diese Abweichungen sind darin begründet, dass das Verhältnis Transitionen/Zustände der Statecharts STOPWATCH und BINTREE-2 bis -8 größer ist als bei den übrigen Beispielen. Daher muss die Regel *EqualNames* auf den Statecharts STOPWATCH, BINTREE-2, BINTREE-4, BINTREE-6, BINTREE-8 nicht so oft ausgewertet werden.

6.3. Ein reales Beispiel

In Kooperation mit der Firma b+m Informatik AG (im Folgenden: bmiag) [11] wurde das im Rahmen dieser Arbeit entwickelte Framework zum Style-Checking auf industriellen Statecharts angewendet; die bmiag verwendet zur Modellierung das UML-CASE-Werkzeug *Rational XDE*. Zur Entwicklung von Web-basierten Applikation setzt die bmiag Aktivitäts-, Use-Case-, Klassen- und Prozesssteuerungs-Diagramme ein. Aktivitätsdiagramme beschreiben die mögliche Navigation in einem System; Aktivitäten repräsentieren Web-Seiten. Das Verhalten der Web-Seiten wird mit Statecharts spezifiziert.

Bei der Arbeit mit Statecharts wird in dem Unternehmen ein eigens entwickelter, stringenter Style-Guide verwendet. Der aufgestellte Style-Guide lässt nur die Verwendung der Statechart-Elemente Zustand, initialer Zustand, finaler Zustand, Choice-Zustand und Transition mit Signal, Guard und Action zu. Darüber hinaus

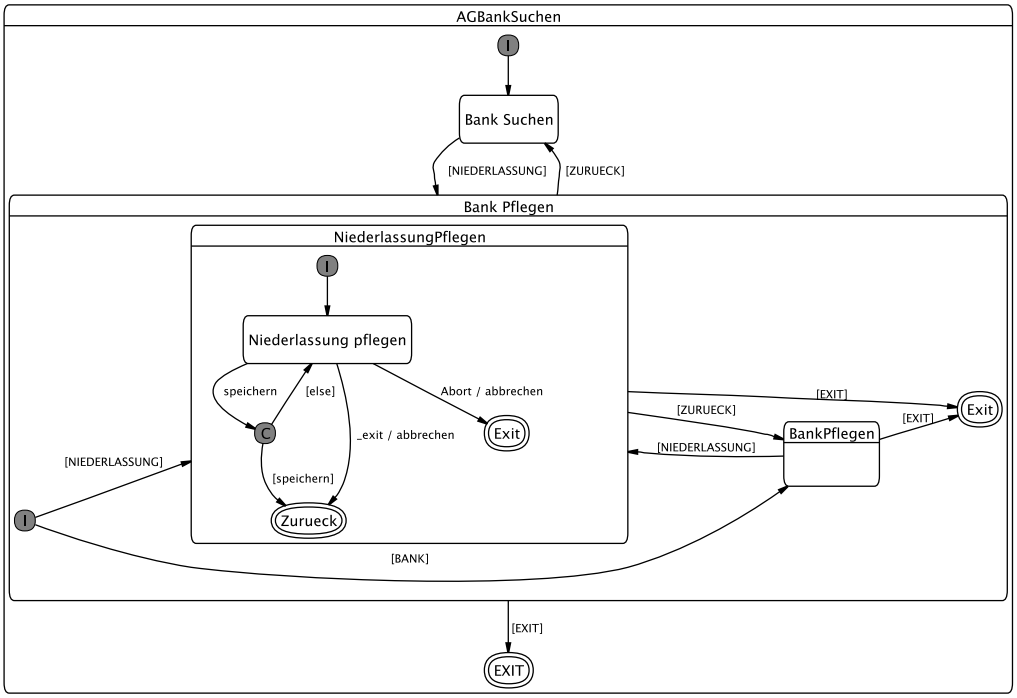


Abbildung 6.2.: Ein transformiertes Aktivitätsdiagramm

werden einfache Zustände mit zwei unterschiedlichen Stereotypen (DoActionState und ActorActionState) weiter unterteilt. Die insgesamt 34 aufgestellten Constraints schränken die Verwendung dieser Elemente zusätzlich ein.

Von den 34 aufgestellten Constraints schränken zehn die Verknüpfung mit Aktivitäten ein und wurden nicht berücksichtigt. Die 24 übrigen Constraints befassen sich ausschließlich mit den Statechart-Elementen. Die eingeführten Stereotypen von Zuständen sind Gegenstand von fünf Regeln und wurden nicht weiter berücksichtigt. Acht Regeln sind identisch mit in der vorliegenden und Schaefers Arbeit aufgestellten Regeln. Die übrigen Regeln des Style-Guides von der bmiag wurden im Rahmen dieser Untersuchung mit der hier entwickelten Sprache KOCL spezifiziert, um eine automatische Analyse der Regeln durchzuführen. Dabei hat sich die Verwendung von KOCL bewährt. Pro Regel aus dem Style-Guide der bmiag wurden weniger als 3 Minuten zur Spezifikation mit KOCL benötigt.

Wie erwähnt, werden Statecharts bei der bmiag zur Verhaltensbeschreibung von Aktivitäten verwendet. Zur Anwendung des entwickelten Statechart Style-Checkers wurden Aktivitätsdiagramme in Statecharts umgewandelt. Im Rahmen der Umwandlung wurden die Aktivitäten durch hierarchische Zustände ersetzt und das zugehörige Statechart wurde als Sub-Statechart des so erzeugten Zustands modelliert. Abbildung 6.2 zeigt stark reduziert den Ausschnitt eines Aktivitätsdiagramms mit eingebetteten Statecharts. Signal-Events sind mit S, Methodenaufrufe mit M als Präfix gekennzeichnet.

In diesem Statechart wurden mit dem in dieser Arbeit entwickelten Style-Checker

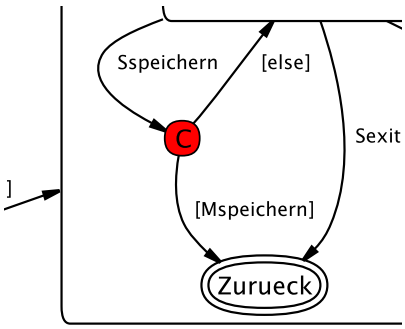


Abbildung 6.3.: Verletztes Constraint der bmiag

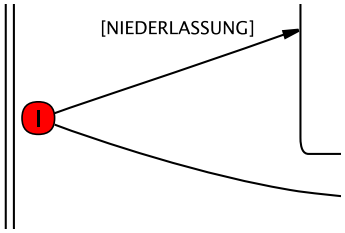


Abbildung 6.4.: Verletzte *Well-formedness*-Regel *PseudoState 1*

Inkonsistenzen sowohl zu dem in dieser Arbeit aufgestellten, als auch zu dem von der bmiag bereitgestellten Style-Guide, aufgezeigt. Aus dem Statechart-Style-Guide der bmiag wurde das Constraint „Als mögliche Zielzustände [von Choice-Zuständen] ausgehender Transitionen sind derzeit nur DoAction- und ActorActionStates erlaubt“ verletzt (vergleiche Abbildung 6.3). Diese Inkonsistenz ist darin begründet, dass ein finaler Zustand kein Actor- oder DoActionState ist. Eine Anpassung des verletzten Constraints, indem man auch finale Zustände als Zielzustände zulässt, würde dieses Problem beseitigen.

In Abbildung 6.4 ist eine Verletzung der *Well-formedness*-Regel *PseudoState 1* (vergleiche Tabelle 6.1) dargestellt. Dieser Fehler resultiert aus der durchgeführten Umwandlung von Aktivitätsdiagrammen in Statecharts. In Statecharts dürfen initiale Zustände nur eine ausgehende Transition besitzen. Demgegenüber dürfen Aktivitätsdiagramme auch mehrere ausgehende Transitionen aufweisen.

Die letzte aufgezeigte Inkonsistenz zu dem in dieser und Schaefers Arbeit aufgestellten Style-Guide ist eine Verletzung der Regel *TransitionOverlap* [85]. In Abbildung 6.5 sind Transitionen mit überlappenden Prädikaten markiert. Die Prädikate der Transitionen überlappen, da für keine der beiden das Komplement der jeweils anderen modelliert wurde.

Der von der bmiag aufgestellte Style-Guide für die Modellierung schränkt den verfügbaren Funktionsumfang, wie erwähnt, stark ein. Daher werden bereits bei der Modellierung viele fehleranfällige Konstruktionen ausgeschlossen. Die Anwendung des in dieser Arbeit entwickelten Style-Checkers hat nur wenige Inkonsistenzen aufgedeckt. Diese Inkonsistenzen zeigen jedoch keine Fehler innerhalb des von der bmiag

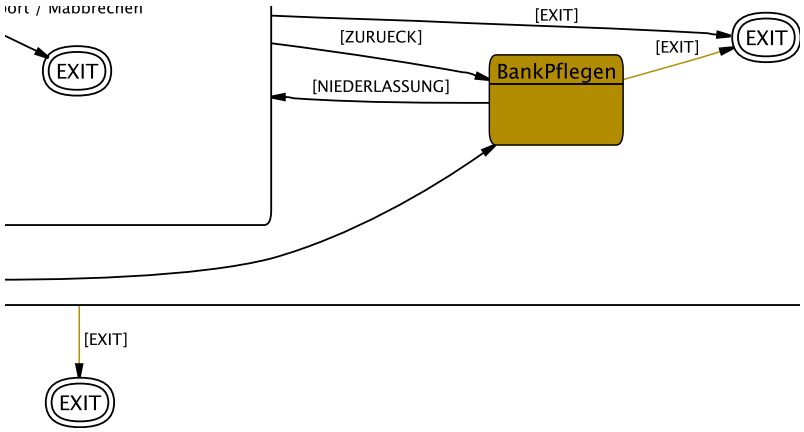


Abbildung 6.5.: Überlappende Transitionen

entwickelten Programms auf, sondern sind als Hinweise auf vermeintlich fehleranfällige Konstruktionen zu verstehen oder sind in der durchgeführten Einbettung von Statecharts begründet. Modifikationen an den zur Verfügung gestellten Statecharts sind daher nicht zwingend notwendig. Die Aufarbeitung aller Hinweise, die durch die Auswertung von *TransitionOverlap* erzeugt werden, wirkt beispielsweise dem *Write-Things-Once*-Prinzip entgegen [85]. Der Entwickler muss also für jeden vom Style-Checker gelieferten Hinweis mit der Intention des Systems abstimmen, ob eine Änderung des Modells notwendig ist.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde zur Fehlervermeidung in der Softwareentwicklung die Anwendung von Style-Guides auf Statecharts untersucht. Die Anwendung von Style-Guides erfolgt klassischerweise mittels Style-Checkern. Eine Betrachtung verschiedener Style-Checker zur Durchführung dieser Aufgabe lieferte das Ergebnis, das die untersuchten Werkzeuge verschiedene Schwächen aufweisen. Sei es, dass der überprüfte Regelkatalog entweder einen zu geringen Umfang hat und nicht erweitert werden kann oder Erweiterungsmöglichkeiten gegeben sind, dafür aber keine semantischen Analysen durchgeführt werden können. Daher wurde im Rahmen dieser Diplomarbeit als erstes ein Style-Guide aufgestellt, der Regeln sowohl die syntaktische Korrektheit, als auch die syntaktische und semantische Robustheit betreffend enthält. Die in den Style-Guide aufgenommenen Regeln wurden verschiedenen Quellen entnommen und in dieser Arbeit anhand einiger Beispiele erläutert.

Die syntaktische Korrektheit betreffende Regeln wurden aufgenommen, da es beobachtet werden konnte, dass die Überprüfung der syntaktischen Korrektheit bei der Arbeit mit UML-Statecharts notwendig ist. Die die syntaktische Korrektheit betreffenden, aufgenommenen Regeln wurden der UML-Spezifikation entnommen. Verletzungen dieser Regeln zeigen Fehler in Statecharts auf, die eine Simulation des modellierten Systems verhindern.

Eine Überprüfung der syntaktischen Robustheit von Statecharts liefert im Gegensatz zur Überprüfung der syntaktischen Korrektheit keine Fehler, die eine Weiterverarbeitung von Statecharts verhindern. Ein Ziel dieser Regeln ist es, den Sprachumfang der Statechart-Syntax auf eine Teilmenge einzuschränken, von der man annimmt, dass sie weniger fehleranfällig ist. Darüberhinaus sollen einige der in dieser Arbeit vorgestellten Regeln das Verständnis eines Statecharts erhöhen und evtl anfallende Wartungsarbeiten erleichtern.

In dieser Arbeit wurde insbesondere der Aspekt der syntaktischen Robustheit eingehend betrachtet. Die semantische Robustheit von Statecharts wurde von Schaefer in der engverwandten Diplomarbeit untersucht und entsprechende Regeln wurden dort implementiert [85].

Zur automatischen Überprüfung der Regeln des aufgestellten Style-Guides wurde im Rahmen dieser Arbeit anschließend ein Framework entwickelt, das die Regeln auf verschiedenen Statechart Dialekten auswerten kann. Diese Checking-Framework ist modular konzipiert, damit es sich einfach erweitern lässt. Dafür wurde eine Werkzeugkette entwickelt mit der sich Regeln in der OCL spezifizieren lassen. Dies hat den Vorteil, dass weitere Regeln zur Untersuchung der syntaktischen Korrektheit

und Robustheit ohne Kenntnisse einer Programmiersprache werden können.

Die OCL bietet keine Möglichkeit, Nachrichten zu formulieren und zurückzuliefern. Daher wurde das textuelle Dateiformat KOCL zur Kombination von Nachrichten und OCL-Ausdrücken entwickelt.

Die abschließende *usability*-Analyse hat gezeigt, dass sich Regeln in dem entwickelten Framework schnell formulieren lassen. Die Analyse der Laufzeiten zeigt, dass der gewählte Ansatz zur Auswertung von OCL für die meisten Regeln eine sehr gute Geschwindigkeit bietet. Die Regel *EqualNames* ist als einziges nicht effizient mit der derzeitigen OCL-Bibliothek umsetzbar. Eine geeignete Implementierung direkt in Java führt in diesem Fall sicherlich zu besseren Ergebnissen, ist aber für die Zukunft noch offen. Die Anwendung des entwickelten Frameworks auf einem Beispiel aus industriellem Umfeld hat Inkonsistenzen zu dem bereitgestellten Style-Guide aufgezeigt. Ein entsprechend konfigurierter Style-Checker für die modellbasierte Entwicklung kann also im Sinne der eingeführten Taxonomie bei der Fehlervermeidung helfen.

Für die Zukunft ist eine Erweiterung der Regelmenge um weitere Regeln, etwa aus der Literatur, wünschenswert. Unter anderem Scaife *et al.* [84] haben sieben Kriterien vorgestellt, anhand derer fehleranfällige Stateflow Modelle identifiziert werden können. Der in dieser Arbeit entwickelte Style-Checker bietet alle technischen Voraussetzungen, um diese zu überprüfen. Jedoch gehen alle von Scaife *et al.* vorgestellten Kriterien über den Funktionsumfang der OCL heraus. Daher müssen die entsprechenden Checks direkt in Java programmiert werden und wurden in dieser Arbeit nicht berücksichtigt. Drei der Kriterien weisen Ähnlichkeiten zu den von Schaefer implementierten Regeln *TransitionOverlap* und *RaceCondition* auf, so dass eine Implementierung entsprechender Checks keine Schwierigkeiten bereiten sollte. Die vier übrigen Kriterien überprüfen Beziehungen von Statechart-Elementen. Diese Beziehungen sind syntaktischer Natur aber bisher nicht behandelt worden. Eine Implementierung entsprechender Java-Klassen zur automatischen Überprüfung dieser Kriterien sollte jedoch auch keine Schwierigkeiten verursachen.

A. Literaturverzeichnis

[1] *IST-2001-33522 OMEGA Correct Development of Real-Time Embedded Systems*, 2001. http://www-omega.imag.fr.

[2] André, Charles: *SyncCharts: A Visual Representation of Reactive Behaviors*. Technischer Bericht RR 95–52, rev. RR (96–56), I3S, Sophia-Antipolis, France, Rev. April 1996. http://www.i3s.unice.fr/~andre/CAPublis/SYNCCHARTS/SyncCharts.pdf.

[3] André, Charles: *Semantics of S.S.M (Safe State Machine)*. Technischer Bericht, Esterel Technologies, Sophia-Antipolis, France, April 2003. available at http://www.esterel-technologies.com, in the download section.

[4] ArgoUML: *Tigris.org: Open Source Software Engineering Tools*. http://argouml.tigris.org/.

[5] Artisan Software: *UML Modeling Tools for Real-time Embedded Systems Modeling and Systems Engineering*. http://www.artisansw.com/.

[6] "BABES-BOLYAI" University of Cluj-Napoca: *Object Constraint Language Environment*. http://lci.cs.ubbcluj.ro/ocle/index.htm.

[7] Barrett, Geoff: *Formal Methods Applied to a Floating-Point Number System*. IEEE Transactions on Software Engineering, 15(5):611–621, 1989.

[8] Beeck, Michael von der: *A Comparison of Statecharts Variants*. In: Langmaack, H., W. P. de Roever und J. Vytopil (Herausgeber): *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Band 863 der Reihe *Lecture Notes in Computer Science*, Seiten 128–148. Springer-Verlag, 1994.

[9] Berry, Gérard: *The Esterel v5 Language Primer, Version v5_91*. Centre de Mathématiques Appliquées Ecole des Mines and INRIA, 06565 Sophia-Antipolis, 2000.

[10] Blair, Michael, Sally Obenski und Paula Bridickas: *PATRIOT MISSILE DEFENSE: Software Problem Led to System Failure at Dhahran, Saudi Arabia*, Februar 1992. http://161.203.16.4/t2pbat6/145960.pdf.

[11] B+M Informatik AG: *B+M Informatik AG Homepage*, 2006. http://www.bmiag.de.

[12] Booch, Grady: *Object-oriented Analysis and Design with Applications*. Benjamin Cummings, Redwood City, 2nd Auflage, 1993.

[13] Borland: *Borland Homepage*. http://www.borland.com.

[14] Bowen, Jonathan: *Formal Specification And Documentation Using Z – A Case Study Approach*. Thomson Publishing, 1996.

[15] *Estbench Esterel Benchmark Suite*. http://www1.cs.columbia.edu/~sedwards/software/estbench-1.0.tar.gz.

[16] Collins, B. P., J. E. Nicholls und I. H. Sorensen: *Introducing formal methods: The CICS experience with Z*. Technischer Bericht TR12.260, IBM, IBM UK Labs Ltd., Hursley Park, UK, 1987.

[17] Cook, Steve und John Daniels: *Designing Object Systems: Object-oriented modelling with Syntropy*. Prentice-Hall, 1994.

[18] Demuth, Birgit, Heinrich Hussmann und Sten Loecher: *OCL as a Specification Language for Business Rules in Database Applications*. Lecture Notes in Computer Science, 2185:104–117, 2001. http://link.springer-ny.com/link/service/series/0558/papers/2185/21850104.pdf.

[19] Dresden OCL Toolkit, 2006. http://dresden-ocl.sourceforge.net/.

[20] Edwards, Stephen A.: *CEC: The Columbia Esterel Compiler*. http://www1.cs.columbia.edu/~sedwards/cec/.

[21] EmPowerTec: *Oclarity*. http://www.empowertec.de/products/rational-rose-ocl.htm.

[22] Esterel Technologies: *Esterel Studio User Manual*, 5.2 Auflage, Juli 2004.

[23] eWebSimplex: *C#/OCL Compiler*, 2004. http://www.ewebsimplex.net/csocl/.

[24] Finger, Frank: *Java-Implementierung der OCL-Basisbibliothek*. Technischer Bericht, Technische Universität Dresden, Fakultät Informatik, Lehrstuhl Softwaretechnologie, 1999.

[25] Finger, Frank: *Design and Implementation of a Modular OCL Compiler*. Diploma Thesis, Dresden University of Technology - Department of Computer Science - Software Engineering Group, März 2000.

[26] Finney, Kate: *Mathematical Notation in Formal Specification: Too Difficult for the Masses?* IEEE Transactions on Software Engineering, 22(2):158–159, Februar 1996.

[27] Flanagan, David: *Java*. in a Nutshell. O'Reilly, 5. Auflage, März 2005.

[28] Ford Motor Company: *Structured Analysis Using Matlab/Simulink/-Stateflow Modeling Style Guidelines*, 1999. `http://vehicle.berkeley.edu/mobies/papers/stylev242.pdf`.

[29] Gagnon, Etienne: *SableCC: Java parser generator*. `http://sablecc.org/`.

[30] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns*. Addison-Wesley, 1995.

[31] Gernert, Christiane: *Agiles Projektmanagement*. Hanser, 2003.

[32] Hanxleden, Reinhard von, Ritwik Bhattacharya und Kay Kossowan: *The State Analyzer, Version 0.1*. Project Report FT3/AS-1999-002, DaimlerChrysler, Februar 1999.

[33] Hanxleden, Reinhard von, Kay Kossowan und Jörg Donandt: *Robustheitskriterien für Statecharts*. Project Report FT3/AS-2000-003, DaimlerChrysler, März 2000.

[34] Harel, David: *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming, 8(3):231–274, Juni 1987.

[35] Harel, David: *On visual formalisms*. Communications of the ACM, 31(5):514–530, 1988.

[36] Harel, David und Michal Politi: *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, New York, 1998.

[37] Hein, Christian, Tom Ritter und Michael Wagner: *Open Source Library for OCL*, 2005. `http://oslo-project.berlios.de/`.

[38] Hussmann, Heinrich, Birgit Demuth und Frank Finger: *Modular architecture for a toolset supporting OCL*. Lecture Notes in Computer Science, 1939:278–293, 2000.

[39] Huuck, Ralf: *Sanity Checks for Stateflow Diagrams*. In: Edwards, Stephen A., Nicolas Halbwachs, Reinhard v. Hanxleden und Thomas Stauner (Herausgeber): *Synchronous Programming – SYNCHRON'04*, Nummer 04491 in *Dagstuhl Seminar Proceedings*, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.

[40] IRISA/CNRS: *Unified Modeling Language All pUrposes Transformer*, 2004. `http://www.irisa.fr/UMLAUT/`.

[41] ISO/IEC: *Information technology – Z formal specification notation – Syntax, type system and semantics*, Juli 2002.

[42] Karlsruhe, Universität von: *KeY*, 1998. http://key-project.org/.

[43] KIEL Project, The: *Project Homepage*, 2004. Kiel Integrated Environment for Layout.

[44] Kleene, Stephen Cole: *Representation of Events in Nerve Nets and Finite Automata*. In: Shannon, C. E. und J. McCarthy (Herausgeber): *Automata Studies*, Seiten 3–41. Princeton University Press, Princeton, New Jersey, 1956.

[45] Kloss, Tobias: *Flexibles und Automatisiertes Layout von Statecharts*. Student resarch project, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Juli 2003.

[46] Kossowan, Kay: *Automatisierte Überprüfung semantischer Modellierungsrichtlinien für Statecharts*. Diplomarbeit, Technische Universität Berlin, 2000.

[47] Kühl, Lars: *Transformation von Esterel nach Esterel Studio*. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, September 2005.

[48] Kyas, Marcel: *Object Constraint Language Verification Platform*.

[49] Kyas, Marcel, Harald Fecher, Frank S. de Boer, Joost Jacob, Jozef Hooman, Mark van der Zwaag, Tamarah Arons und Hillel Kugler: *Formalizing UML Models and OCL Constraints in PVS*. In: *Electronic Notes in Theoretical Computer Science*. Elsevier, April 2004.

[50] Ladkin, Peter B.: *Causal Reasoning about Aircraft Accidents*. In: Koornneef, Floor und Meine van der Meulen (Herausgeber): *Computer Safety, Reliability and Security, 19th International Conference, SAFECOMP 2000, Rotterdam, The Netherlands, October 24-27, 2000*, Band 1943 der Reihe *Lecture Notes in Computer Science*, Seiten 344–360. Springer, 2000.

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

[52] Lions, Jacques-Louis: *Ariane 5: Flight 501 failure – report by the inquiry board*. Press Release 33, Juli 1996. http://www.esa.int/esaCP/Pr_33_1996_p_EN.html.

[53] Löcher, Sten: *UML/OCL für die Integritätssicherung in Datenbankanwendungen*. Diplomarbeit, Technische Universität Dresden, 2001.

[54] MathWorks Automotive Advisory Board (MAAB): *Controller Style Guidelines for Production Intent Using MATLAB, Simulink and Stateflow*, April 2001. http://www.mathworks.com/industries/auto/maab.html.

[55] MATHWORKS INC.: *Stateflow and Stateflow Coder for use with Simulink — User's Guide*. Mathworks Inc., 6 Auflage, 2004.

[56] MCCULLOCH, W. S. und W. PITTS: *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, 5:115–133, 1943.

[57] MEALY, G. H.: *A Method for Synthesizing Sequential Circuits*. Bell System Technical Journal, 34:1045–1079, September 1955.

[58] MONNINGER, FRIEDER: *Eiffel. Objektorientiertes Programmieren in der Praxis*. Heise Heinz, 1993.

[59] MOORE, EDWARD F.: *Gedanken-Experiments on Sequential Machines*. In: SHANNON, C. E. und J. MCCARTHY (Herausgeber): *Automata Studies*, Nummer 34 in *Annals of Mathematical Studies*, Seiten 129–153. Princeton University Press, Princeton, NJ, 1956.

[60] MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION (MISRA): *MISRA-C:2004. Guidelines for the Use of the C Language in Critical Systems*. Motor Industry Research Association (MIRA), Nuneaton CV10 0TU, UK, 2004.

[61] MOUTOS, MILTIADIS, ALBRECHT KORN und CARSTEN FISEL: *Guideline-Checker*. Studienarbeit, University of Applied Sciences in Esslingen, Juni 2000.

[62] MUTZ, MARTIN: *Ein Regel Checker zur Verbesserung des modellbasierten Softwareentwurfs*. Toolpräsentation, Technische Universität Braunschweig, 2003. `http://www.cs.tu-bs.de/ips/mutz/papers/EKA2003.pdf`.

[63] MUTZ, MARTIN: *Eine durchgängige modellbasierte Entwurfsmethodik für eingebettete Systeme im Automobilbereich*. Dissertation, Technische Universität Braunschweig, 2005.

[64] MUTZ, MARTIN und MICHAELA HUHN: *Automated Statechart Analysis for User-defined Design Rules*. Technischer Bericht, Technische Universität Braunschweig, 2003. `http://www.cs.tu-bs.de/ips/mutz/papers/TR2003-10.pdf`.

[65] OBJECT MANAGEMENT GROUP: *Unified Modeling Lanugage—UML Resource Page*. `http://www.uml.org`.

[66] OBJECT MANAGEMENT GROUP: *Unified Modeling Language (UML) 1.3 specification*, Februar 2000. `http://www.omg.org/cgi-bin/apps/doc?formal/00-03-01.pdf`.

[67] OBJECT MANAGEMENT GROUP: *OMG Unified Modeling Language Specification, Version 1.5*, März 2003. `http://www.omg.org/cgi-bin/doc?formal/03-03-01`.

[68] Oxford Univerity Computing Laboratory: *Programming Research Group Homepage*, 2004. http://web.comlab.ox.ac.uk/oucl/about/prg/.

[69] Paderborn Software Engineering Group, University of: *Fujaba*, 2006. http://www.fujaba.de/.

[70] Pap, Zsigmond, Istvan Majzik und Andras Pataricza: *Checking General Safety Criteria on UML Statecharts*. Lecture Notes in Computer Science, 2187, 2001.

[71] Parnas, David L.: *On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System*. In: *Proceedings of the 24th National ACM Conference*, Seiten 379–385, 1969.

[72] Parnas, David L.: *Some Theorems We Should Prove*. In: *HUG '93: Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*, Seiten 155–162, London, UK, 1994. Springer-Verlag.

[73] Posor, Adrian: *Extension of KIEL by Stateflow Charts*. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Dezember 2005.

[74] Prochnow, Steffen und Reinhard von Hanxleden: *Comfortable Modeling of Complex Reactive Systems*. In: *Proceedings of Design, Automation and Test in Europe (DATE'06)*, Munich, März 2006.

[75] Prochnow, Steffen, Gunnar Schaefer, Ken Bell und Reinhard von Hanxleden: *Analyzing Robustness of UML State Machines*. In: *Proceedings of the Workshop on Modeling and Analysis of Real-Time and Embedded Systems (MARTES'06), held in conjunction with the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS/UML 2006*, Genua, Oktober 2006.

[76] Prochnow, Steffen, Claus Traulsen und Reinhard von Hanxleden: *Synthesizing Safe State Machines from Esterel*. In: *Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, Ottawa, Canada, Juni 2006.

[77] Project without a name: *PWAN OCL*, 1999. http://pwan.sourceforge.net/index.html.

[78] Rabin, M. O. und D. Scott: *Finite Automata and their Decision Problems*. IBM Journal of Research and Development, 3:114–125, 1959.

[79] Rational Software: *Rational Rose Technical Developer*. http://www-306.ibm.com/software/awdtools/developer/technical/.

[80] Rhapsody: *I-Logix, Inc.* http://www.ilogix.com/products/.

[81] Ricardo Company, The: *Mint – A Style checker for Simulink and Stateflow*, 2006. http://www.ricardo.com/engineeringservices/controlelectronics.aspx?page=mint.

[82] Richters, Mark: *UML-based Specification Environment*, 2001. http://www.db.informatik.uni-bremen.de/projects/USE/.

[83] Rumbaugh, James, Michael Blaha und Wiliam Premerlani: *Objektorientiertes Modellieren und Entwerfen*. Hanser Fachbuch, 1993.

[84] Scaife, N., C. Sofronis, P. Caspi, S. Tripakis und F. Maraninchi: *Defining and translating a "safe" subset of Simulink/Stateflow into Lustre*. Technischer Bericht 2004-16, Verimag, Centre Équation, 38610 Gières, Juli 2004. http://www-verimag.imag.fr/index.php?page=techrep-list.

[85] Schaefer, Gunnar: *Statechart Style Checking – Automated Semantic Robustness Analysis of Statecharts*. Diploma Thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Juni 2006.

[86] Spivey, J. M.: *The Z notation: a reference manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[87] Stölzel, Mirko, Steffen Zschaler und Leif Geiger: *Integrating OCL and Model Transformations in Fujaba*. Technischer Bericht, Dresden University of Technology, Department of Computer Science, 2006.

[88] Sun Microsystems, Inc.: *Code Conventions for the Java Programming Language*, 1997. http://java.sun.com/docs/codeconv/.

[89] The MathWorks Inc.: *Stateflow and Stateflow Coder User's Guide, Version 6*. Natick, MA, September 2005. http://www.mathworks.com/access/helpdesk/help/pdf_doc/stateflow/sf_ug.pdf.

[90] The Object Management Group: *UML Homepage*.

[91] Warmer, Jos und Anneke Kleppe: *OCL Tool for Precise UML Specifications*. http://www.klasse.nl/octopus/index.html.

[92] Warmer, Jos B. und Anneke G. Kleppe: *The object constraint language: precise modeling with UML*. Addison-Wesley, Reading, Massachusetts, 1998.

[93] Zheng, Jiang, Laurie Williams, Nachiappan Nagappan, Will Snipes, John Hudepohl und Mladen Vouk: *A Study of Static Analysis for Fault Detection in Software*. Technischer Bericht, Department of Computer Science, North Carolina State University, Microsoft Research, Nortel Networks Software Dependability Design, 2005.

[94] Zündorf, Albert: *Graph Pattern Matching in PROGRES*. In: *Selected papers from the 5th International Workshop on Graph Gramars and Their Application to Computer Science*, Seiten 454–468, London, UK, 1996. Springer-Verlag.

B. Regelprofile

Die auf einem Statechart ausgewerteten Regeln können abhängig vom Dialekt über Profile ausgewählt werden. Diese Profile werden in den Einstellungen von KIEL gespeichert. Ein Profil wird geladen, sobald ein Statechart in KIEL geladen wird. In der visuellen Variante des Programms können die zu überprüfenden Regeln über den entsprechenden Menüpunkt vor dem Vorgang an- und abgewählt werden (Vergleiche Abbildung 5.6). Folgende Profile wurden bisher für die Regeln der syntaktischen Robustheit (Kapitel 4.2) angelegt:

Esterel Studio V. 5.0

1. IsolatedStates
2. EqualNames
3. OrStateCount
4. RegionStateCount
5. DefaultFromJunction
6. TransitionLabels

MatlabStateflow V. 6.1

1. IsolatedStates
2. EqualNames
3. OrStateCount
4. RegionStateCount
5. DefaultFromJunction
6. TransitionLabels
7. Connectivity
8. InterlevelTransition
9. InitialState

ArgoUML V. 0.2.0.x

1. IsolatedStates
2. EqualNames
3. OrStateCount
4. RegionStateCount
5. DefaultFromJunction
6. TransitionLabels
7. Connectivity
8. InterlevelTransition
9. InitialState

Für die Überprüfungen aus dem Bereich der syntaktischen Korrektheit wurden bisher folgende Profile angelegt (Die Regeln bezeichnen die entsprechende KOCL-Spezifikation):

Esterel Studio V. 5.0

1. UMLCompositeStateRule1
2. UML13CompositeStateRule3
3. UML13FinalStateRule1
4. UML13PseudoStateRule1
5. UML13PseudoStateRule5
6. UML13PseudoStateRule6
7. UMLFinalStateConstraint1
8. UMLPseudostateConstraint1
9. UMLTransitionConstraint5

MatlabStateflow V. 6.1

1. UML13CompositeStateRule1
2. UML13CompositeStateRule2
3. UML13CompositeStateRule3
4. UML13CompositeStateRule4

5. UML13CompositeStateRule5
6. UML13CompositeStateRule6
7. UML13FinalStateRule1
8. UML13PseudoStateRule1
9. UML13PseudoStateRule2
10. UML13PseudoStateRule6
11. UML13SynchStateRule1
12. UML13SynchStateRule2
13. UMLStateMachineRule4
14. UMLFinalStateConstraint1
15. UMLPseudostateConstraint1
16. UMLTransitionConstraint2
17. UMLTransitionConstraint5

ArgoUML V. 0.2.0.x

1. UML13CompositeStateRule1
2. UML13CompositeStateRule2
3. UML13CompositeStateRule3
4. UML13CompositeStateRule4
5. UML13CompositeStateRule5
6. UML13CompositeStateRule6
7. UML13FinalStateRule1
8. UML13PseudoStateRule1
9. UML13PseudoStateRule2
10. UML13PseudoStateRule3
11. UML13PseudoStateRule4
12. UML13PseudoStateRule5

13. UML13PseudoStateRule6
14. UML13SynchStateRule1
15. UML13SynchStateRule2
16. UMLFinalStateConstraint1
17. UMLPseudostateConstraint1
18. UMLTransitionConstraint1
19. UMLTransitionConstraint2
20. UMLTransitionConstraint3
21. UMLTransitionConstraint4
22. UMLTransitionConstraint5
23. UMLTransitionConstraint6
24. UMLTransitionConstraint7

C. Die EBNF von KOCL

In Abbildung C.1 ist die EBNF der KOCL-Grammatik angegeben. Mit KOCL (siehe Kapitel 5.3.2) werden OCL-Ausdrücke mit Hinweistexten kombiniert. Es gibt die Möglichkeit mehrere Hinweisetexte in einer Regel zu spezifizieren. Der Kontext des OCL-Ausdrucks wurde erweitert, so dass die spezifizierte OCL-Bedingung auf mehreren Klassen ausgewertet werden kann. Dadurch wird vermieden, die gleiche Regel für unterschiedliche Klassen mehrfach spezifizieren zu müssen. Der Umstand, dass eine Regel auf verschiedenen Klassen ausgewertet werden kann, wurde in der Definition der Nachrichtenbehandlung berücksichtigt. Abhängig von der Klasse, auf der die OCL-Bedingung ausgewertet wird, lässt sich eine der deklarierten Nachrichten zurückliefern.

⟨*booleanops*⟩ ::= '`and`' | '`or`'

⟨*identifier*⟩ ::= ⟨*nondigit*⟩ (⟨*digit*⟩ | ⟨*nondigit*⟩)*

⟨*string*⟩ ::= '`"`' (⟨*digit*⟩ | ⟨*nondigit*⟩)* '`"`'

⟨*rules*⟩ ::= ⟨*rule*⟩+

⟨*rule*⟩ ::= ⟨*rule-heading*⟩ '`{`' ⟨*error-part*⟩? ⟨*declaration-part*⟩?
⟨*constraint-definition*⟩ ⟨*conforms-part*⟩? ⟨*fails-part*⟩? '`}`'

⟨*rule-heading*⟩ ::= '`rule`' ⟨*identifier*⟩

⟨*error-part*⟩ ::= '`error`' '`;`'

⟨*declaration-part*⟩ ::= '`declarations`' '`{`' ⟨*message-definition*⟩+ '`}`'

⟨*message-definition*⟩ ::= ⟨*identifier*⟩ ⟨*string*⟩ '`;`'

⟨*constraint-definition*⟩ ::= '`constraint`' '`{`' ⟨*context-part*⟩ ⟨*constraint-part*⟩ '`}`'

⟨*context-part*⟩ ::= '`context`' ⟨*identifiert-list*⟩ '`;`'

⟨*identifier-expression*⟩ ::= '`not`'* ⟨*identifier*⟩

⟨*identifier-list*⟩ ::= ⟨*identifier-expression*⟩ ⟨*identifier-list-tail*⟩?

⟨*identifier-list-tail*⟩ ::= ⟨*booleanops*⟩ ⟨*identifier-list*⟩

⟨*constraint-part*⟩ ::= ⟨*string*⟩ '`;`'

⟨*conforms-part*⟩ ::= '`conforms`' '`{`' ⟨*eval-statement-part*⟩? '`}`'

⟨*fails-part*⟩ ::= '`fails`' '`{`' ⟨*eval-statement-part*⟩? '`}`'

⟨*eval-statement-part*⟩ ::= ⟨*if-then-statement*⟩
| ⟨*if-then-else-statement*⟩
| ⟨*expression-semicolon*⟩

⟨*if-then-statement*⟩ ::= '`if`' ⟨*expression*⟩ '`then`' ⟨*identifier*⟩ '`;`'

⟨*if-then-else-statement*⟩ :== ⟨*if-then-statement*⟩ '`else`' ⟨*identifier*⟩ '`;`'

⟨*expression*⟩ ::= ⟨*identifier*⟩
| '`{`' ⟨*identifier*⟩ ⟨*booleanops*⟩ ⟨*expression*⟩ '`}`'

Abbildung C.1.: Die KOCL-Grammatik in EBNF.

D. Das Metamodell

Abbildung D.1 enthält eine vereinfachte Version des Metamodells der in KIEL verwendeten Statechart Datenstruktur. Die vollständige Version des Metamodells wird als XMI-Datei an das im Regel-Generator verwendete *Dresden OCL Toolkit* übergeben (siehe Kapitel 5.3.2). Das Metamodell wurde mit *ArgoUML* angelegt und aus dem Werkzeug ebenfalls als XMI-Datei exportiert. Alle KOCL-Regeln wurden über diesem Metamodell definiert.

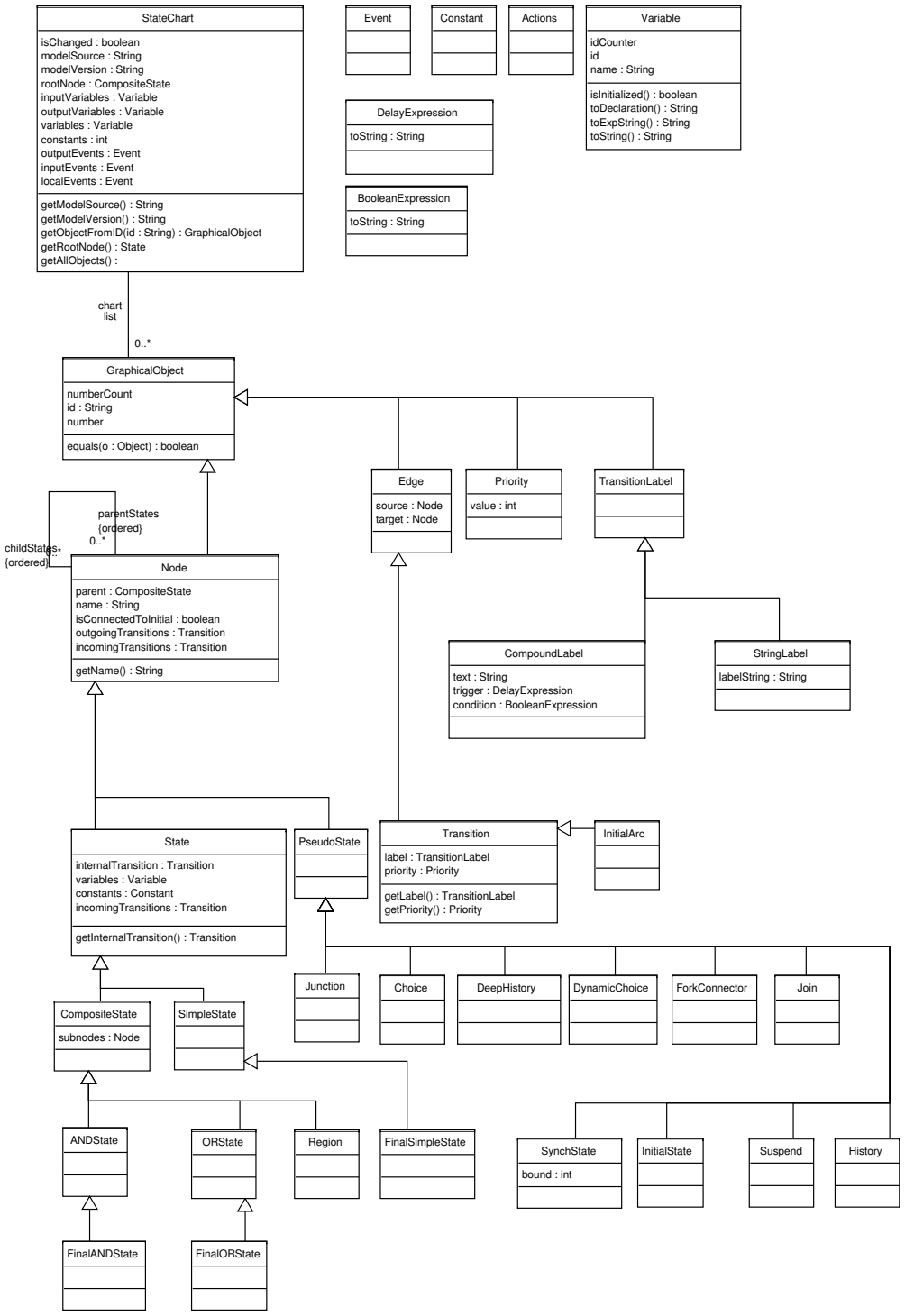


Abbildung D.1.: Das vereinfachte Metamodell der topologischen Statechart Datenstruktur.

E. Parserspezifikationen im SableCC Format

Dieser Abschnitt enthält die Grammatiken für KOCL und den Transitionslabel-Parser. Beide Grammatiken sind im SableCC-Format angegeben. Der mit SableCC erzeugte KOCL-Parser, Abschnitt E.1, ist Bestandteil des Regel-Generators. Für den Transitionslabel-Parser (siehe Kapitel 5.3.2), Abschnitt E.2, wurde eine Java 1.4 Grammatik von Etienne Gragon [29] modifiziert.

E.1. KOCL-Grammatik

```
Package kiel.util.checkingHelpers.synrule;

/***********************************************************************
 * Helpers *
 ***********************************************************************/
Helpers
   all = [0 .. 127];
   digit = ['0' .. '9'];
   nondigit = ['_' + ['#' + [['a' .. 'z'] + ['A' .. 'Z']]]];
   cr = 13;
   lf = 10;

   c_char = [all - ['"' + [10 + 13]]];
   c_char_sequence = c_char+;


   tab = 9;
   not_cr_lf = [[all - cr] - lf];

   and = 'and';
   or = 'or';

   not_star  =  [ all - '*' ];
   not_star_slash  =  [ not_star - '/' ];


/***********************************************************************
 * Tokens *
 ***********************************************************************/
Tokens
   semicolon = ';';
   endofruledot = '.';
   l_scope = '{';
   r_scope = '}';
   l_brace = '(';
   r_brace = ')';

   rule = 'rule';
   error = 'error';
   declarations = 'declarations';
   constraint = 'constraint';
   context = 'context';
   conforms = 'conforms';
   fails = 'fails';
   if = 'if';
   then = 'then';
   else = 'else';
   this = 'this';
   not = 'not';

   booleanops = (and | or);
   identifier = nondigit (nondigit | digit)*;
   string = '"' c_char_sequence? '"';

   blank = (cr | lf | cr lf |tab | ' ')+;

   comment = '/*' not_star* '*' + ( not_star_slash not_star* '*' + )* '/';

/***********************************************************************
 * Ignored Tokens *
 ***********************************************************************/
Ignored Tokens
   blank,
   comment;


/***********************************************************************
 * Productions *
 ***********************************************************************/
Productions


   // More than one rule per file
   rules =
        P.rule+;

   // rule syntax
   rule =
     rule_heading l_scope
```

```
    is_error_part?
    decl_part?
    constraint_definition_part
    conforms_return_part?
    fails_return_part?
    r_scope;

  // Head of the rule
  rule_heading =
    T.rule identifier;

  // optional declaration of error type
  is_error_part =
    T.error semicolon;

  // message declarations
  decl_part =
    declarations l_scope message_definition+ r_scope;
  message_definition =
    identifier string semicolon;

  // constraint declaration
  constraint_definition_part =
    constraint l_scope context_part constraint_part r_scope;
  context_part =
    context identifier_list semicolon;
  identifier_expr =
     T.not* identifier;
  identifier_list =
        identifier_expr identifier_list_tail?;
  identifier_list_tail =
     booleanops identifier_list;

// {single} T.not* identifier |
// {brace} l_brace identifier_list booleanops identifier_list r_brace;

  constraint_part =
    string semicolon;

  // message return part if the chart conforms to the constraint
  conforms_return_part =
    conforms l_scope eval_statement_part? r_scope;

  // message return part if the chart does not conform to the constraint
  fails_return_part =
    fails l_scope eval_statement_part? r_scope;

  // optional if then else statement within both message return parts
  eval_statement_part =
    {onlyif} if_then_statement
    | {else} if_then_else_statement
    | {direct} expression semicolon;
  if_then_statement =
    if expression then identifier semicolon;
  if_then_else_statement =
    if_then_statement else identifier semicolon;
  expression =
    {simple} identifier
    | l_brace identifier booleanops expression r_brace;
```

E.2. JavaTransitionLabels-Grammatik

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * This file is part of J11. *
 * See the file "J11-LICENSE" for Copyright information and the *
 * terms and conditions for copying, distribution and *
 * modification of J11. *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/***************************************************************
 * Etienne Gagnon: I have built this grammar based to the *
 * information on pages 19-23 of the "Inner Classes Specification *
 * document. *
 * Sun has not released yet an official grammar for Java 1.1 and I *
 * suspect that the definition of Java identifiers has changed and *
 * is different from the Java 1.02 version. This intuition comes *
 * from noticing the deprecation of Character.isJavaLetter() and *
 * isJavaLetterOrDigit() methods. *
 * In this grammar, I have not changed the lexer. So I scan *
 * Java 1.02 identifiers. This is the only documented definition *
 * of identifiers that I have. If somebody has better information, *
 * please let me know. *
 *
 * Changes including inner class updates, more robust semicolon
 * processing and strictfp keyword by:
 * Thomas Leonhardt
 *
 ***************************************************************/


Package kiel.fileInterface.JavaTransitionLabels;

/***************************************************************
 * Helpers *
 ***************************************************************/
Helpers

   unicode_input_character = [0..0xffff];
   ht = 0x0009;
   lf = 0x000a;
   ff = 0x000c;
   cr = 0x000d;
   sp = ' ';

   line_terminator = lf | cr | cr lf;
   input_character = [unicode_input_character - [cr + lf]];

   not_star = [input_character - '*'] | line_terminator;
   not_star_not_slash = [input_character - ['*' + '/']] | line_terminator;

   unicode_letter =
       [0x0041..0x005a] | [0x0061..0x007a] | [0x00aa..0x00aa] |
[0x00b5..0x00b5] |
       [0x00ba..0x00ba] | [0x00c0..0x00d6] | [0x00d8..0x00f6] |
```

```
    [0x00f8..0x01f5] |
           [0x01fa..0x0217] | [0x0250..0x02a8] | [0x02b0..0x02b8] |
    [0x02bb..0x02c1] |
           [0x02d0..0x02d1] | [0x02e0..0x02e4] | [0x037a..0x037a] |
    [0x0386..0x0386] |
           [0x0388..0x038a] | [0x038c..0x038c] | [0x038e..0x03a1] |
    [0x03a3..0x03ce] |
           [0x03d0..0x03d6] | [0x03da..0x03da] | [0x03dc..0x03dc] |
    [0x03de..0x03de] |
           [0x03e0..0x03e0] | [0x03e2..0x03f3] | [0x0401..0x040c] |
    [0x040e..0x044f] |
           [0x0451..0x045c] | [0x045e..0x0481] | [0x0490..0x04c4] |
    [0x04c7..0x04c8] |
           [0x04cb..0x04cc] | [0x04d0..0x04eb] | [0x04ee..0x04f5] |
    [0x04f8..0x04f9] |
           [0x0531..0x0556] | [0x0559..0x0559] | [0x0561..0x0587] |
    [0x05d0..0x05ea] |
           [0x05f0..0x05f2] | [0x0621..0x063a] | [0x0640..0x064a] |
    [0x0671..0x06b7] |
           [0x06ba..0x06be] | [0x06c0..0x06ce] | [0x06d0..0x06d3] |
    [0x06d5..0x06d5] |
           [0x06e5..0x06e6] | [0x0905..0x0939] | [0x093d..0x093d] |
    [0x0958..0x0961] |
           [0x0985..0x098c] | [0x098f..0x0990] | [0x0993..0x09a8] |
    [0x09aa..0x09b0] |
           [0x09b2..0x09b2] | [0x09b6..0x09b9] | [0x09dc..0x09dd] |
    [0x09df..0x09e1] |
           [0x09f0..0x09f1] | [0x0a05..0x0a0a] | [0x0a0f..0x0a10] |
    [0x0a13..0x0a28] |
           [0x0a2a..0x0a30] | [0x0a32..0x0a33] | [0x0a35..0x0a36] |
    [0x0a38..0x0a39] |
           [0x0a59..0x0a5c] | [0x0a5e..0x0a5e] | [0x0a72..0x0a74] |
    [0x0a85..0x0a8b] |
           [0x0a8d..0x0a8d] | [0x0a8f..0x0a91] | [0x0a93..0x0aa8] |
    [0x0aaa..0x0ab0] |
           [0x0ab2..0x0ab3] | [0x0ab5..0x0ab9] | [0x0abd..0x0abd] |
    [0x0ae0..0x0ae0] |
           [0x0b05..0x0b0c] | [0x0b0f..0x0b10] | [0x0b13..0x0b28] |
    [0x0b2a..0x0b30] |
           [0x0b32..0x0b33] | [0x0b36..0x0b39] | [0x0b3d..0x0b3d] |
    [0x0b5c..0x0b5d] |
           [0x0b5f..0x0b61] | [0x0b85..0x0b8a] | [0x0b8e..0x0b90] |
    [0x0b92..0x0b95] |
           [0x0b99..0x0b9a] | [0x0b9c..0x0b9c] | [0x0b9e..0x0b9f] |
    [0x0ba3..0x0ba4] |
           [0x0ba8..0x0baa] | [0x0bae..0x0bb5] | [0x0bb7..0x0bb9] |
    [0x0c05..0x0c0c] |
           [0x0c0e..0x0c10] | [0x0c12..0x0c28] | [0x0c2a..0x0c33] |
    [0x0c35..0x0c39] |
           [0x0c60..0x0c61] | [0x0c85..0x0c8c] | [0x0c8e..0x0c90] |
    [0x0c92..0x0ca8] |
           [0x0caa..0x0cb3] | [0x0cb5..0x0cb9] | [0x0cde..0x0cde] |
    [0x0ce0..0x0ce1] |
           [0x0d05..0x0d0c] | [0x0d0e..0x0d10] | [0x0d12..0x0d28] |
    [0x0d2a..0x0d39] |
           [0x0d60..0x0d61] | [0x0e01..0x0e2e] | [0x0e30..0x0e30] |
    [0x0e32..0x0e33] |
           [0x0e40..0x0e46] | [0x0e81..0x0e82] | [0x0e84..0x0e84] |
    [0x0e87..0x0e88] |
           [0x0e8a..0x0e8a] | [0x0e8d..0x0e8d] | [0x0e94..0x0e97] |
    [0x0e99..0x0e9f] |
           [0x0ea1..0x0ea3] | [0x0ea5..0x0ea5] | [0x0ea7..0x0ea7] |
    [0x0eaa..0x0eab] |
           [0x0ead..0x0eae] | [0x0eb0..0x0eb0] | [0x0eb2..0x0eb3] |
    [0x0ebd..0x0ebd] |
           [0x0ec0..0x0ec4] | [0x0ec6..0x0ec6] | [0x0edc..0x0edd] |
    [0x0f40..0x0f47] |
           [0x0f49..0x0f69] | [0x10a0..0x10c5] | [0x10d0..0x10f6] |
    [0x1100..0x1159] |
           [0x115f..0x11a2] | [0x11a8..0x11f9] | [0x1e00..0x1e9b] |
    [0x1ea0..0x1ef9] |
           [0x1f00..0x1f15] | [0x1f18..0x1f1d] | [0x1f20..0x1f45] |
    [0x1f48..0x1f4d] |
           [0x1f50..0x1f57] | [0x1f59..0x1f59] | [0x1f5b..0x1f5b] |
    [0x1f5d..0x1f5d] |
           [0x1f5f..0x1f7d] | [0x1f80..0x1fb4] | [0x1fb6..0x1fbc] |
    [0x1fbe..0x1fbe] |
           [0x1fc2..0x1fc4] | [0x1fc6..0x1fcc] | [0x1fd0..0x1fd3] |
    [0x1fd6..0x1fdb] |
           [0x1fe0..0x1fec] | [0x1ff2..0x1ff4] | [0x1ff6..0x1ffc] |
    [0x207f..0x207f] |
           [0x2102..0x2102] | [0x2107..0x2107] | [0x210a..0x2113] |
    [0x2115..0x2115] |
           [0x2118..0x211d] | [0x2124..0x2124] | [0x2126..0x2126] |
    [0x2128..0x2128] |
           [0x212a..0x2131] | [0x2133..0x2138] | [0x3005..0x3005] |
    [0x3031..0x3035] |
           [0x3041..0x3094] | [0x309b..0x309e] | [0x30a1..0x30fa] |
    [0x30fc..0x30fe] |
           [0x3105..0x312c] | [0x3131..0x318e] | [0x4e00..0x9fa5] |
    [0xac00..0xd7a3] |
           [0xf900..0xfa2d] | [0xfb00..0xfb06] | [0xfb13..0xfb17] |
    [0xfb1f..0xfb28] |
           [0xfb2a..0xfb36] | [0xfb38..0xfb3c] | [0xfb3e..0xfb3e] |
    [0xfb40..0xfb41] |
           [0xfb43..0xfb44] | [0xfb46..0xfbb1] | [0xfbd3..0xfd3d] |
    [0xfd50..0xfd8f] |
           [0xfd92..0xfdc7] | [0xfdf0..0xfdfb] | [0xfe70..0xfe72] |
    [0xfe74..0xfe74] |
           [0xfe76..0xfefc] | [0xff21..0xff3a] | [0xff41..0xff5a] |
    [0xff66..0xffbe] |
           [0xffc2..0xffc7] | [0xffca..0xffcf] | [0xffd2..0xffd7] |
    [0xffda..0xffdc];

       unicode_digit =
           [0x0030..0x0039] | [0x0660..0x0669] | [0x06f0..0x06f9] |
    [0x0966..0x096f] |
           [0x09e6..0x09ef] | [0x0a66..0x0a6f] | [0x0ae6..0x0aef] |
    [0x0b66..0x0b6f] |
           [0x0be7..0x0bef] | [0x0c66..0x0c6f] | [0x0ce6..0x0cef] |
    [0x0d66..0x0d6f] |
           [0x0e50..0x0e59] | [0x0ed0..0x0ed9] | [0x0f20..0x0f29] |
    [0xff10..0xff19];

       java_letter = unicode_letter | '$' | '_';
       java_letter_or_digit = unicode_letter | unicode_digit | '$' | '_';

       non_zero_digit = ['1'..'9'];
       digit = ['0'..'9'];
       hex_digit = ['0'..'9'] | ['a'..'f'] | ['A'..'F'];
```

```
    octal_digit = ['0'..'7'];
    zero_to_three = ['0'..'3'];

    decimal_numeral = '0' | non_zero_digit digit*;
    hex_numeral = '0' ('x' | 'X') hex_digit+;
    octal_numeral = '0' octal_digit+;

    integer_type_suffix = 'l' | 'L';

    exponent_part = ('e' | 'E') ('+' | '-')? digit+;

    float_type_suffix = 'f' | 'F' | 'd' | 'D';

    single_character = [input_character - ['' + '\']];
    octal_escape = '\' (octal_digit octal_digit? | zero_to_three octal_digit
octal_digit);
    escape_sequence = '\b' | '\t' | '\n' | '\f' | '\r' | '\"' | '\' '' |
'\\' | octal_escape;
    string_character = [input_character - ['"' + '\']] | escape_sequence;

/*****************************************************************
 * Tokens *
 *****************************************************************/
Tokens

    white_space = (sp | ht | ff | line_terminator)*;

    traditional_comment = '/*' not_star+ '*'+ (not_star_not_slash not_star*
'*'+)* '/';
    documentation_comment = '/**' '*'* (not_star_not_slash not_star*
'*'+)* '/';


/*************************************************************************
************
    * Here, we take into account the possibilily that the line terminator
might be missing *
    * at the end of the last line of a file. This is imprecise in the Java
Language *
    * Specification, because it is not clear if a line terminator should be
added to the *
    * last line, or not. "javac", the reference compiler, accepts and
end-of-line-comment, *
    * even if the line terminator is missing from the last line.
*

*************************************************************************
***********/
    end_of_line_comment = '//' input_character* line_terminator?;


    true = 'true';
    false = 'false';
    null = 'null';

    l_parenthese = '(';
    r_parenthese = ')';
    l_brace = '{';
    r_brace = '}';
    l_bracket = '[';
    r_bracket = ']';
    semicolon = ';';
    comma = ',';
    dot = '.';

    assign = '=';
    lt = '<';
    gt = '>';
    complement = '!';
    bit_complement = '~';
    question = '?';
    colon = ':';

    eq = '==';
    lteq = '<=';
    gteq ='>=';
    neq = '!=';
    and = '&&';
    or = '||';
    plus_plus = '++';
    minus_minus = '--';

    plus = '+';
    minus = '-';
    star = '*';
    div = '/';
    bit_and = '&';
    bit_or = '|';
    bit_xor = '^';
    mod = '%';
    shift_left = '<<';
    signed_shift_right = '>>';
    unsigned_shift_right = '>>>';

    plus_assign = '+=';
    minus_assign = '-=';
    star_assign = '*=';
    div_assign = '/=';
    bit_and_assign = '&=';
    bit_or_assign = '|=';
    bit_xor_assign = '^=';
    mod_assign = '%=';
    shift_left_assign = '<<=';
    signed_shift_right_assign = '>>=';
    unsigned_shift_right_assign = '>>>=';

    decimal_integer_literal = decimal_numeral integer_type_suffix?;
    hex_integer_literal = hex_numeral integer_type_suffix?;
    octal_integer_literal = octal_numeral integer_type_suffix?;

    floating_point_literal =
        digit+ '.' digit* exponent_part? float_type_suffix? |
        '.' digit+ exponent_part? float_type_suffix? |
        digit+ exponent_part float_type_suffix? |
        digit+ exponent_part? float_type_suffix;

    character_literal = ''' (single_character | escape_sequence) ''';
    string_literal = '"' string_character* '"';

    identifier = java_letter java_letter_or_digit*;
```

```
/*****************************************************************
 * Ignored Tokens *
 *****************************************************************/
Ignored Tokens

  white_space,
  traditional_comment,
  documentation_comment,
  end_of_line_comment;

/*****************************************************************
 * Productions *
 *****************************************************************/
Productions

/*****************************************************************
19.2 Grammar from Â§2.3: The Syntactic Grammar Â§2.3
*****************************************************************/

goal =
    transition_label;


transition_label =
    trigger? guard? effect?;

trigger = simple_name;

guard = l_bracket expression r_bracket;

effect =
       {single}
          div statement_expression |

       {multiple}
       div expression_statement+;

/*****************************************************************
19.3 Grammar from Â§3: Lexical Structure Â§3
*****************************************************************/

literal =
    {integer_literal}
        integer_literal |

    {floating_point_literal}
        floating_point_literal |

    {boolean_literal}
        boolean_literal |

    {character_literal}
        character_literal |

    {string_literal}
        string_literal |

    {null_literal}
        null_literal;
```

```
/*****************************************************************
19.4 Grammar from Â§4: Types, Values, and Variables Â§4

type =
    {primitive_type}
        primitive_type |

    {reference_type}
        reference_type;

primitive_type =
    {numeric_type}
        numeric_type |

    {boolean}
        boolean;

numeric_type =
    {integral_type}
        integral_type |

    {floating_point_type}
        floating_point_type;

integral_type =
    {byte}
        byte |

    {short}
        short |

    {int}
        int |

    {long}
        long |

    {char}
        char;

floating_point_type =
    {float}
        float |

    {double}
        double;

reference_type =
    {class_or_interface_type}
        class_or_interface_type |

    {array_type}
        array_type;

class_or_interface_type =
    name;

class_type =
    class_or_interface_type;
```

```
interface_type =
    class_or_interface_type;

array_type =
    {primitive_type}
        primitive_type dims |

    {name}
        name dims;

**********************************************************************/


/**********************************************************************
19.5 Grammar from Â§6: Names Â§6
**********************************************************************/

name =
    {simple_name}
        simple_name |

    {qualified_name}
        qualified_name;

simple_name =
    identifier;

qualified_name =
    name dot identifier;


/**********************************************************************
19.11 Grammar from Â§14: Blocks and Statements Â§14
**********************************************************************/

expression_statement =
    statement_expression semicolon;

statement_expression =
    {assignment}
        assignment |

    {single_identifier}
        simple_name ;


/**********************************************************************
19.12 Grammar from Â§15: Expressions Â§15
**********************************************************************/

primary =
    {primary_no_new_array}
        primary_no_new_array;

primary_no_new_array =
    {literal}
        literal |

    {l_parenthese}
        l_parenthese expression r_parenthese ;


postfix_expression =
    {primary}
        primary |

    {name}
        name |

    {post_increment_expression}
        post_increment_expression |

    {post_decrement_expression}
        post_decrement_expression;

post_increment_expression =
    postfix_expression plus_plus;

post_decrement_expression =
    postfix_expression minus_minus;

unary_expression =
    {pre_increment_expression}
        pre_increment_expression |

    {pre_decrement_expression}
        pre_decrement_expression |

    {plus}
        plus unary_expression |

    {minus}
        minus unary_expression |

    {unary_expression_not_plus_minus}
        unary_expression_not_plus_minus;

pre_increment_expression =
    plus_plus unary_expression;

pre_decrement_expression =
    minus_minus unary_expression;

unary_expression_not_plus_minus =
    {postfix_expression}
        postfix_expression |

    {bit_complement}
        bit_complement unary_expression |

    {complement}
        complement unary_expression;


multiplicative_expression =
    {unary_expression}
        unary_expression |
```

```
        {star}
            multiplicative_expression star unary_expression |

        {div}
            multiplicative_expression div unary_expression |

        {mod}
            multiplicative_expression mod unary_expression;

additive_expression =
    {multiplicative_expression}
        multiplicative_expression |

    {plus}
        additive_expression plus multiplicative_expression |

    {minus}
        additive_expression minus multiplicative_expression;

shift_expression =
    {additive_expression}
        additive_expression |

    {shift_left}
        shift_expression shift_left additive_expression |

    {signed_shift_right}
        shift_expression signed_shift_right additive_expression |

    {unsigned_shift_right}
        shift_expression unsigned_shift_right additive_expression;

relational_expression =
    {shift_expression}
        shift_expression |

    {lt}
        relational_expression lt shift_expression |

    {gt}
        relational_expression gt shift_expression |

    {lteq}
        relational_expression lteq shift_expression |

    {gteq}
        relational_expression gteq shift_expression;

equality_expression =
    {relational_expression}
        relational_expression |

    {eq}
        equality_expression eq relational_expression |

    {neq}
        equality_expression neq relational_expression;

and_expression =
    {equality_expression}
        equality_expression |

    {and_expression}
        and_expression bit_and equality_expression;

exclusive_or_expression =
    {and_expression}
        and_expression |

    {exclusive_or_expression}
        exclusive_or_expression bit_xor and_expression;

inclusive_or_expression =
    {exclusive_or_expression}
        exclusive_or_expression |

    {inclusive_or_expression}
        inclusive_or_expression bit_or exclusive_or_expression;

conditional_and_expression =
    {inclusive_or_expression}
        inclusive_or_expression |

    {conditional_and_expression}
        conditional_and_expression and inclusive_or_expression;

conditional_or_expression =
    {conditional_and_expression}
        conditional_and_expression |

    {conditional_or_expression}
        conditional_or_expression or conditional_and_expression;

conditional_expression =
    {conditional_or_expression}
        conditional_or_expression |

    {question}
        conditional_or_expression question expression colon
conditional_expression;

assignment_expression =
    {conditional_expression}
        conditional_expression |

    {assignment}
        assignment;

assignment =
    left_hand_side assignment_operator assignment_expression;

left_hand_side =
    {name}
        name;

assignment_operator =
    {assign}
        assign |

    {star_assign}
```

```
    star_assign |
    {div_assign}
        div_assign |
    {mod_assign}
        mod_assign |
    {plus_assign}
        plus_assign |
    {minus_assign}
        minus_assign |
    {shift_left_assign}
        shift_left_assign |
    {signed_shift_right_assign}
        signed_shift_right_assign |
    {unsigned_shift_right_assign}
        unsigned_shift_right_assign |
    {bit_and_assign}
        bit_and_assign |
    {bit_xor_assign}
        bit_xor_assign |
    {bit_or_assign}
        bit_or_assign;

expression =
    assignment_expression;


constant_expression =
    expression;

/**********************************************************************
Literals
**********************************************************************/

boolean_literal =
    {true} true |
    {false} false;

null_literal =
    null;

integer_literal =
    {decimal} decimal_integer_literal |
    {hex} hex_integer_literal |
    {octal} octal_integer_literal;
```

F. KOCL-Spezifikationen

Im folgenden sind die KOCL-Spezifikationen aller umgesetzten Regeln angegeben. Diese Regeln werden im Build-Prozess von KIEL automatisch in Java-Klassen übersetzt.

F.1. *Well-formedness*-Regeln

F.1.1. UML 1.3

```
rule UML13CompositeStateRule1 {
    error;
    declarations {
      message "A composite state can have at most one initial vertex.";
    }

    constraint {
      context ORState or Region;
      "self.subnodes->select (v | v.oclIsTypeOf(InitialState))->size <= 1";
    }

    fails {
      message;
    }
}

rule UML13CompositeStateRule2 {
    error;

    declarations {
      message "A composite state can have at most one deep history vertex.";
    }

    constraint {
      context ORState or Region;
      "self.subnodes->select (v | v.oclIsTypeOf(DeepHistory))->size <= 1";
    }

    fails {
      message;
    }
}

rule UML13CompositeStateRule3 {
    error;
    declarations {
      message "A composite state can have at most one shallow history vertex.";
    }

    constraint {
      context ORState or Region;
      "self.subnodes->select (v | v.oclIsTypeOf(History))->size <= 1";
    }

    fails {
      message;
    }
}

rule UML13CompositeStateRule4 {
    error;
    declarations {
      message "There have to be at least two composite substates in a concurrent composite state
      .";
    }

    constraint {
      context ANDState;
      "subnodes->select (v | v.oclIsTypeOf(Region))->size >=2";
    }

    fails {
      message;
    }
}

rule UML13CompositeStateRule5 {
    error;
    declarations {
      message "A concurrent state can only have composite states as substates.";
    }

    constraint {
      context ANDState;
      "subnodes->forAll(s | (s.oclIsTypeOf(Region)))";
    }

    fails {
      message;
    }
}


rule UML13CompositeStateRule6 {
```

```
    error;
    declarations {
      message "The substates of a composite state are part of only that composite state.";
    }

    constraint {
      context Region or ORState;
      "subnodes->forAll(s|s.parent.id = self.id)";
    }

    fails {
      message;
    }
}


rule UML13FinalStateRule1 {
    error;
    declarations {
      message "A final state cannot have any outgoing transitions";
    }

    constraint {
      context FinalSimpleState or FinalANDState or FinalORState;
      "outgoingTransitions->size = 0";
    }

    fails {
      message;
    }
}

rule UML13PseudoStateRule1 {

    error;
    declarations {
      message "An initial vertex can have at most one outgoing transition and no incoming
      transition.";
    }

    constraint {
      context InitialState;
      "(outgoingTransitions->size <= 1) and (incomingTransitions->size = 0)";
    }

    fails {
      message;
    }

}


rule UML13PseudoStateRule2 {

    error;
    declarations {
      message "History vertices can have at most one outgoing transition.";
    }

    constraint {
      context History or DeepHistory;
      "outgoingTransitions->size <= 1";
    }

    fails {
      message;
    }
}


rule UML13PseudoStateRule3 {

    error;
    declarations {
      message "A join vertex must have at least two incoming transitions and exactly one outgoing
       transition.";
    }

    constraint {
      context Join;
      "(incomingTransitions->size >= 2) and (outgoingTransitions->size = 1)";
    }

    fails {
      message;
    }
}

rule UML13PseudoStateRule4 {

    error;
    declarations {
      message "A fork vertex must have at least two outgoing transitions and exactly one incoming
       transition.";
    }

    constraint {
      context ForkConnector;
      "(incomingTransitions->size = 1) and (outgoingTransitions->size >= 2)";
    }

    fails {
      message;
    }
}


rule UML13PseudoStateRule5 {

    error;
    declarations {
      message "A junction vertex must have at least one incoming and one outgoing transition.";
    }

    constraint {
      context Junction;
      "(incomingTransitions->size >= 1) and (outgoingTransitions->size >= 1)";
    }
```

```
    fails {
      message;
    }
}

rule UML13PseudoStateRule6 {

    error;
    declarations {
      message "A choice vertex must have at least one incoming and one outgoing transition.";
    }

    constraint {
      context Choice;
      "(incomingTransitions->size >= 1) and (outgoingTransitions->size >= 1)";
    }

    fails {
      message;
    }
}


rule UML13StateMachineRule2 {

    error;
    declarations {
      message "A top state is always a composite.";
    }

    constraint {
      context StateChart;
      "self.rootNode.oclIsKindOf(CompositeState)";
    }

    fails {
      message;
    }

}

rule UML13StateMachineRule3 {

    error;
    declarations {
      message "A top state cannot have any containing states.";
    }

    constraint {
      context StateChart;
      "self.rootNode.parentStates->size=0";
    }

    fails {
      message;
    }
}


rule UMLStateMachineRule4 {

    error;
    declarations {
      message "The top state cannot be the source of a transition.";
    }

    constraint {
      context StateChart;
      "self.rootNode.outgoingTransitions->size=0";
    }

    fails {
     message;
    }
}


rule UML13SynchStateRule1 {

    error;
    declarations {
      message "The value of the bound attribute must be a positive integer, or unlimited.";
}

    constraint {
      context SynchState;
      "bound>0";
    }

    fails {
      message;
    }
}


rule UML13SynchStateRule2 {

    error;
    declarations {
      message "All incoming transitions to a SynchState must come from the same region and all
      outgoing transitions from a SynchState must go to the same region.";
    }

    constraint {
      context SynchState;
      "(incomingTransitions->forAll(t1,t2 | t1.source.parent = t2.source.parent)) and (
      outgoingTransitions->forAll(t1,t2 | t1.target.parent = t2.target.parent))";
    }

    fails {
      message;
    }
}
```

F.1.2. UML 2.0

```
rule UMLPseudostateConstraint1 {
    error;

    declarations {
      message "An initial vertex can have at most one outgoing transition.";
    }

    constraint {
       context InitialState;
       "self.outgoingTransitions->size <= 1";
    }

    fails {
      message;
    }
}


rule UMLTransitionConstraint1 {

    error;
    declarations {
      message "A fork segment must not have guards or triggers.";
    }

    constraint {
      context Transition;
      "(source.oclIsTypeOf(ForkConnector) and self.label.oclIsTypeOf(CompoundLabel)) implies ((
      self.label.oclAsType(CompoundLabel).trigger.toString = '') and (self.label.oclAsType(
      CompoundLabel).condition.toString = ''))";
    }

    fails {
      message;
    }
}

rule UMLTransitionConstraint2 {

    error;
    declarations {
      message "A join segment must not have guards or triggers.";
    }

    constraint {
      context Transition;
      "(target.oclIsTypeOf(Join) and self.label.oclIsTypeOf(CompoundLabel)) implies ((self.label.
      oclAsType(CompoundLabel).trigger.toString = '') and (self.label.oclAsType(CompoundLabel).
      condition.toString = ''))";
    }

    fails {
      message;
    }
}

rule UMLTransitionConstraint3 {

    error;
    declarations {
      message "A fork segment must always target a state.";
    }

    constraint {
      context Transition;
      "source.oclIsTypeOf(ForkConnector) implies target.oclIsTypeOf(State)";
    }

    fails {
      message;
    }
}


rule UMLTransitionConstraint4 {

    error;
    declarations {
      message "A join segment must always originate from a state.";
    }

    constraint {
      context Transition;
      "target.oclIsTypeOf(Join) implies source.oclIsTypeOf(State)";
    }

    fails {
      message;
    }
}


rule UMLTransitionConstraint5 {

    error;
    declarations {
      message "Transitions outgoing pseudostates may not have a trigger.";
    }

    constraint {
      context Transition;
      "source.oclIsKindOf(PseudoState) implies if (label.oclIsTypeOf(StringLabel)) then (label.
      oclAsType(StringLabel).labelString = '') else (label.oclAsType(CompoundLabel).trigger ='')
      endif";
    }

    fails {
      message;
    }
}


rule UMLTransitionConstraint6 {

    error;
    declarations {
```

```
    message "Join segments should originate from orthogonal states.";
  }

  constraint {
    context Transition;
    "target.oclIsTypeOf(Join) implies target.parent.parent.oclIsTypeOf(ANDState)";
  }

  fails {
    message;
  }
}

rule UMLTransitionConstraint7 {
  error;

  declarations {
    message "Fork segments should target orthogonal states.";
  }

  constraint {
    context Transition;
    "source.oclIsTypeOf(ForkConnector) implies target.parent.parent.oclIsTypeOf(ANDState)";
  }

  fails {
    message;
  }
}
```

F.2. Robustheitsregeln

```
rule InitialStateCount {

       declarations {
          failor "An orstate should contain exactly one initial vertex.";
          failregion "A region should contain exactly one initial vertex.";
       }

       constraint {
         context Region or ORState;
          "subnodes->select(oclIsTypeOf(InitialState))->size = 1";
       }

       fails {
            if ORState then
                      failor;
         else
                      failregion;
       }

}


rule ORStateCount_1 {
     declarations{
       message "An Orstate should contain at least one state.";
     }

     constraint {
       context ORState;
       "self.subnodes->size>0";
     }

     fails {
       message;
     }
}

rule RegionStateCount_1 {
     declarations{
       message "The region contains no states.";
     }

     constraint {
       context Region;
       "self.subnodes->size>0";
     }

     fails {
       message;
     }
}

rule ORStateCount {
     declarations{
       failmessage "An Orstate should contain at least two states.";
     }

     constraint {
       context ORState;
       "subnodes->select(oclIsKindOf(SimpleState) or oclIsKindOf(ANDState) or oclIsKindOf(ORState
       ))->size >= 2";
     }

       fails {
              failmessage;
       }

}

rule MiracleState {

       declarations {
          message "The state has no incoming transition.";
       }

       constraint {
         context SimpleState and not InitialState and not DeepHistory and not History;
          "not (parentStates->size = 0) implies incomingTransitions->size > 0";
       }

       fails {
              message;
       }

}

rule TransitionLabels {
    declarations {
        message "The transition has no label or no Trigger.";
    }

    constraint {
      context Transition;
      "(not (self.source.oclIsKindOf(PseudoState))) implies if (label.oclIsTypeOf(StringLabel))
       then (label.oclAsType(StringLabel).labelString <> '') else (label.oclAsType(CompoundLabel).
       text <> '') endif ";
   }

   fails{
     message;
   }
}

rule IsolatedStates {
  declarations {
       message "Either the state is isolated or one of its children is entered by an interlevel
      transition.";
  }

  constraint {
       context Node and not Region;
       "(parentStates->size = 0) or ((incomingTransitions->size + outgoingTransitions->size) > 0)
      ";
  }
```

```
  fails{
    message;
  }
}


rule InterlevelTransitions {

       declarations {
              failsmessage "The transition is interlevel.";
       }
       constraint {
              context Transition;
              "self.target.parent.id = self.source.parent.id";
       }
       fails{
              failsmessage;
       }
}


rule Connectivity {

    declarations {
      message "The node is not on a path from an initial state.";
    }

    constraint {
      context State and not Region;
      "(self.incomingTransitions->size > 0) implies self.isConnectedToInitial";
    }

    fails {
      message;
    }
}


rule EqualNames{
    declarations {
      message "Another state has the same name.";
    }

    constraint {
      context State and not Region;
      "(self.chart.list->select(s| s.oclIsKindOf(Node)))->select(s|s.oclAsType(Node).name = self.
      name)->size <= 1";
    }

    fails {
      message;
    }
}

rule DefaultFromJunction {

    declarations {
      message "There should be a default transition from each connective junction.";
    }

    constraint {
       context Choice;
       "self.outgoingTransitions->select(t|if(t.label.oclIsTypeOf(StringLabel)) then t.label.
      oclAsType(StringLabel).labelString = '' else t.label.oclAsType(CompoundLabel).text = ''
      endif)->size = 1";
    }

    fails {
      message;
    }
}
```

G. Java Code

In diesem Abschnitt wird der Quellcode des in Kapitel 5.3 beschriebenen Checking-Plug-Ins präsentiert. Abschnitt G.1 enthält den entwickelten XMI-Konverter. Dieser konvertiert das mit ArgoUML angelegt Metamodell so, dass es vom *Dresden OCL Toolkit* eingelesen werden kann. Anschließend, im Abschnitt G.2, ist der Quellcode des Regel-Generators enthalten. Dieser erzeugt aus KOCL-Spezifikation unter Verwendung des *Dresden OCL Toolkit*s Java-Klassen für das Checking-Plug-In. Der Quellcode des Checking-Plug-Ins ist in Abschnitt G.3 enthalten. Abschließend (Abschnitt G.4) wird der Quellcode des entwickelten XMI-Fileinterfaces präsentiert. Abbildung G.1 zeigt die Struktur des Checking-Plug-Ins.

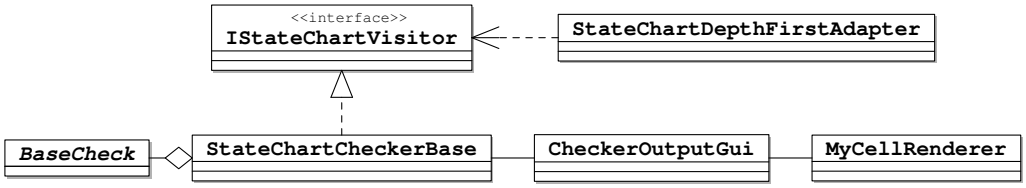


Abbildung G.1.: Übersicht über die Abhängigkeiten des Checking-Plug-Ins.

G.1. XMI-Konverter

G.1.1. XMIConverter.java

```
package kiel.util.checkingHelpers.XMIConverter;

import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

/**
 * <p>Description: The xmiConverter class is the main class of the tool
 * used for backporting XMI metadata files generated
 * by argoUML 0.20 to the format generated by argoUML 0.0.7.
 * This is mandatory due to limitations of the used version
 * of the Dresden OCL Toolkit. It can process the older version only.
 * It uses the XMISaxConverter to perform the conversion.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/10/31 09:37:53 $
 */
public final class XMIConverter {


    /**
     * The constructor of the class. It is private to prevent instantation.
     *
     */
    private XMIConverter() {

    }


    /**
     * The field containing the output file.
     */
    private static String output;


    /**
     * Internal method, that loads the xmi meta modell file to an instance of
     * XMISaxConverter.
     * @param input The name of the input file.
     * @return True, if no error occured.
     */
    private static boolean convertFile(final String input) {

        boolean result = true;

        try {
            DefaultHandler handler = new XMISaxConverter(output);


            XMLReader parser = (XMLReader) (Class.forName(
                    XMISaxConverter.DEFAULT_PARSER_NAME).newInstance());

            parser.setFeature("http://xml.org/sax/features/validation",
                    false);
            parser.setFeature("http://xml.org/sax/features/namespaces",
                    false);
            parser.setFeature(
                    "http://apache.org/xml/features/validation/schema",
                    false);
            parser.setFeature(
                    "http://apache.org/xml/features/validation/"
                    + "schema-full-checking", false);

            parser.setContentHandler(handler);
            parser.setErrorHandler(handler);

            parser.parse(input);
        } catch (Exception e) {
            e.printStackTrace();
            result = false;
        }

        return result;

    }

    /**
     * The main method of the tool.
     * It needs two arguments. The first argument has to be the input file.
     * The second argument is the name of the output file.
     * @param args The input parameters.
     */
    public static void main(final String[] args) {

        if (args.length != 2) {
            System.out.println(
                    "\nKIEL xmi-file converter.\n"
                    + "Used for converting xmi-files generated by "
                    + "argoUML v. 0.2.0\n"
                    + "into the format readable by the Dresden OCL "
                    + "Toolkit v. 1.3.\n\n"
                    + "Usage: xmiConverter <Inputfile> <Outputfile>\n");
            System.exit(1);
        }


        output = args[1];
        if (convertFile(args[0])) {
            System.out.println(args[0] + " converted successful.");
        } else {
```

```
            }
        }
    System.out.println(args[0] + " not converted. "
        + "See output for more information.");
    }
```

G.1.2. XMISaxConverter.java

```
package kiel.util.checkingHelpers.XMIConverter;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;


/**
 * <p>Description: The translator class. It uses a SaxParser to traverse the
 * XML tree.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.4 $ last modified $Date: 2006/10/31 09:37:54 $
 */
public class XMISaxConverter extends DefaultHandler {


    /** Default parser name. */
    public static final String
        DEFAULT_PARSER_NAME = "org.apache.xerces.parsers.SAXParser";



    /**
     * The name of the file to write the changes to.
     */
    private String outputfile;

    /**
     * The XMLDocument that gets created.
     */
    private Document output;

    /**
     * The current node.
     */
    private Node current;

    /**
     * Internally used field. Needed in certain situations to decide if a node
     * has to be handled or not.
     */
    private boolean handleElement = true;

    /**
     * Internally used.
     */
    private boolean isTypeElement = false;

    /**
     * Internally used field to denote if the current node is a multiplicity
     * node.
     */
    private boolean isMultiplicity = false;

    /**
     * Counter for the header nodes.
     */
    private int headerNr = 0;

    /**
     * The constructor of the class.
     * Initializes a DocumentBuilder and creates the output document.
     * @param o The name of the outputfile.
     */
    public XMISaxConverter(final String o) {
        super();
        this.outputfile = o;
        current = null;

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

        output = null;
        DocumentBuilder db;
        try {
            db = dbf.newDocumentBuilder();
            output = db.newDocument();
        } catch (ParserConfigurationException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }


    /**
     * Internally used method to remove a string from a name of a node.
     * @param name The string, where the part is removed from.
     * @param toRemove The part to be removed.
     * @return The input string where the part was removed from.
     */
    private static String removeFromName(final String name,
            final String toRemove) {
        int index = name.indexOf(toRemove);

        if (index > -1) {
```

```
            return name.substring(0, index)
            + name.substring(index + toRemove.length());
        } else {
            return name;
        }
    }

    /**
     * This method transforms a name of the currently handled node to the old
     * format.
     * @param raw The read name of the node.
     * @return The transformed node name.
     */
    private static String createValidNodeName(final String raw) {

        String nodeName = raw;

        nodeName = removeFromName(nodeName, "UML:");
        nodeName = removeFromName(nodeName, "Namespace.");
        nodeName = removeFromName(nodeName, "Classifier.");
        nodeName = removeFromName(nodeName, "BehavioralFeature.");
        nodeName = removeFromName(nodeName, "StructuralFeature.");
        nodeName = removeFromName(nodeName, "Parameter.");
        nodeName = removeFromName(nodeName, "Association.");
        nodeName = removeFromName(nodeName, "AssociationEnd.");

        if (nodeName.equalsIgnoreCase("generalization.child")) {
            nodeName = "subtype";
        }

        if (nodeName.equalsIgnoreCase("generalization.parent")) {
            nodeName = "supertype";
        }

        if (nodeName.equals("participant")) {
            nodeName = "type";
        }
        return nodeName;
    }


    /**
     * Message handler for processing instrunctions.
     * Writes the target and data to the prompt.
     * @param target A string.
     * @param data A data string.
     */
    public final void processingInstruction(final String target,
            final String data) {
        System.out.println(target + " ** " + data);
    }

    /**
     * Message handler for the startDocument event.
     */
    public final void startDocument() {
        System.out.println("");
    }

    /**
     * Internally used method for correct transformation of the multiplicity
     * tag.
     * @param raw The raw string of the node.
     * @param attrs The list of attributes of the node.
     */
    private void handleMultiplicity(final String raw, final Attributes attrs) {
        String nodename = createValidNodeName(raw);

        if (nodename.indexOf("Range") != -1) {

            String endval = attrs.getValue("upper");

            if (Integer.parseInt(endval) == -1) {
                endval = "*";
            }

            current.appendChild(output.createTextNode(
                    attrs.getValue("lower") + ".." + endval));
        }

    }

    /**
     * Method to create a Type Element.
     * @param raw The string of the handled node.
     * @param attrs The list of the attributes of the node.
     */
    private void handleTypeElement(final String raw, final Attributes attrs) {
        Element newEle = output.createElement("XMI.reference");
        newEle.setAttribute("target", attrs.getValue("xmi.idref"));
        current.appendChild(newEle);
    }

    /**
     * Add a new element to the document.
     * @param raw The string of the current node.
     * @param attrs The list of the attributes.
     */
    private void createNewElement(final String raw, final Attributes attrs) {

        Element newEle = null;

        if (isTypeElement || (attrs.getIndex("xmi.idref") > -1)) {
            handleTypeElement(raw, attrs);
            return;
        }

        if (isMultiplicity) {
            handleMultiplicity(raw, attrs);
            return;
        }

          newEle = output.createElement(createValidNodeName(raw));

        if (current == null) {
            output.appendChild(newEle);
            current = newEle;
        } else {
            current.appendChild(newEle);
            current = newEle;
```

```
        }

        if (isTypeElement) {
            return;
        }

        isTypeElement = newEle.getNodeName().equalsIgnoreCase("type");
        isMultiplicity = newEle.getNodeName().indexOf("multiplicity") != -1;

        if (isTypeElement || isMultiplicity) {

        } else {


            // no subnodes for the first XMI node
            if (newEle.getNodeName().equalsIgnoreCase("xmi")) {
                return;
            }

            // XMI metamodel information have to be parameterised
            if (newEle.getNodeName().equalsIgnoreCase("XMI.metamodel")) {

                for (int index = 0; index < attrs.getLength(); index++) {
                    newEle.setAttribute(attrs.getQName(index),
                            attrs.getValue(index));
                }

                return;
            }

            for (int index = 0; index < attrs.getLength(); index++) {

                // only add xmi.id as attribute.
                // all others become children
                if (attrs.getQName(index).toLowerCase().indexOf(
                        "xmi.id") > -1) {
                    Attr a = output.createAttribute("XMI.id");
                    a.setNodeValue(attrs.getValue(index));
                    newEle.setAttributeNode(a);
                } else {
                        // all attributes except "name", get their values as params
                    Element child;
                    if (attrs.getQName(index).equalsIgnoreCase("ordering")) {
                        child = output.createElement("isOrdered");
                    } else {
                        child = output.createElement(attrs.getQName(index));
                    }

                    newEle.appendChild(child);
                    if (child.getNodeName().equalsIgnoreCase("name")) {

                        child.appendChild(output.createTextNode(
                                attrs.getValue(index)));
                    } else {
                        if (child.getNodeName().equalsIgnoreCase("isOrdered")) {
                            /*
                             * - the value is set to false per default.
                             */
                            if (attrs.getValue(index).equals("ordered")) {
                                child.setAttribute("XMI.value", "true");
                            } else {
                                child.setAttribute("XMI.value", "false");
                            }
                        } else {
                            child.setAttribute("XMI.value",
                                    attrs.getValue(index));
                        }
                    }

                }

            }

            if (current.getNodeName().equalsIgnoreCase("AssociationEnd")
                    && (attrs.getIndex("name") == -1)) {
                Element namechild = output.createElement("name");
                current.appendChild(namechild);
            }

        }

    }


    /**
     * Start a new element according to the paramenters.
     * @param uri The uri of the node.
     * @param local .
     * @param raw The string representation of the node.
     * @param attrs The list of the attributes of the node.
     */
    public final void startElement(final String uri, final String local,
            final String raw, final Attributes attrs) {

        if (raw.toLowerCase().indexOf("header") > -1) {
            headerNr++;
        }

        //handleElement = handleElement
        // && raw.toLowerCase().indexOf("multiplicity") == -1;
        if (headerNr == 1) {
            return;
        }

        if (!handleElement) {
            return;
        }

        createNewElement(raw, attrs);
    }


    /**
     * React to the endElement event.
     * @param uri The uri of the node.
     * @param local .
     * @param raw The string representation of the node.
     */
    public final void endElement(final String uri, final String local,
            final String raw) {
```

```
        String node = createValidNodeName(raw);

        if (raw.toLowerCase().indexOf("header") > -1) {
          headerNr--;
        }

        if (node.indexOf("multiplicity") != -1) {
          isMultiplicity = false;
        }

360     if (node.equalsIgnoreCase("type")) {
          isTypeElement = false;
        }


        if ((current instanceof Element)
            && (((Element) current).getTagName().equalsIgnoreCase(node))) {
          current = current.getParentNode();
        }
370   }
```

```
      /**
       * React to the endDocument event and write the generated document.
       */
      public final void endDocument() {
        XMLSerializer xmls;
        try {
          OutputFormat format = new OutputFormat(output);
          format.setIndenting(true);
          format.setLineWidth(80);
          xmls = new XMLSerializer(new FileOutputStream(
              new File(this.outputfile)), format);
          xmls.serialize(output);
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
380
390
    }
```

G.2. Regel-Generator

G.2.1. RuleParser.java

```
package kiel.util.checkingHelpers.translator;

import java.io.File;
import java.io.FileReader;
import java.io.FilenameFilter;
import java.io.PushbackReader;

import kiel.util.checkingHelpers.synrule.lexer.Lexer;
import kiel.util.checkingHelpers.synrule.node.Start;
import kiel.util.checkingHelpers.synrule.parser.Parser;
import kiel.util.checkingHelpers.synrule.parser.ParserException;

/**
 * <p>Description: This is the main part of the rule translator.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.5 $ last modified $Date: 2006/10/31 09:34:51 $
 */
public final class RuleParser {

    /**
     * The total count of needed params.
     */
    private static final int MAXVALIDPARAMCOUNT = 4;

    /**
     * The index of the parameter for the package name.
     */
    private static final int PARAMIDXPACKAGENAME = 3;


    /**
     * empty constructor to prevent instantiation and to make checkstyle happy.
     *
     */
    private RuleParser() {

    }


    /**
     * The main method of the rule translator.
     * @param args The user input.
     */
    public static void main(final String[] args) {

        if ((args.length == 0) || (args.length != MAXVALIDPARAMCOUNT)) {
            System.out.println("KIEL Rule to Java Converter");
            System.out.println("Usage: RuleParser <ModelFile> <RuleDir>"
                    + " <Base Output Directory> <BasePackage>");
            System.exit(1);
        }

        File basedir = new File(args[1]);
        String nameOfFailedRule = "";

        File[] rules = basedir.listFiles(
                new FilenameFilter() {
                    public boolean accept(final File f, final String s) {
                        return s.toLowerCase().endsWith(".rules");
                    }
                }
            );

        File f = null;
        Parser p = null;

        Translator t = null;

        for (int i = 0; i < rules.length; i++) {
            f = rules[i];

            boolean result = false;
            final int fileBufferSize = 1024;
            try {
                p = new Parser(new Lexer(new PushbackReader(new FileReader(f),
                                fileBufferSize)));
                Start tree = p.parse();
                RuleSemanticAnalyzer rsa = new RuleSemanticAnalyzer();
                System.out.println("Starting semantic analysis...");
                tree.apply(rsa);
                System.out.println("...finished");

                String dir = args[2];
                String packagename = args[PARAMIDXPACKAGENAME];
                // create the name of the package
                if (packagename.charAt(packagename.length() - 1) != '.') {
                    packagename += ".";
                }

                packagename += f.getName().substring(0,
                        f.getName().lastIndexOf('.'));

                // check if the directory for the package exists in
                // the output dir and create it, if it doesn't
                if (dir.charAt(dir.length() - 1) != File.separatorChar) {
                    dir += File.separatorChar;
                }
                dir += f.getName().substring(0, f.getName().lastIndexOf('.'))
```

```
        + File.separator;

    File fDir = new File(dir);
    if (!fDir.exists()) {
        fDir.mkdir();
    }

    if (rsa.getProblemCount() == 0) {
        t = new Translator(
                args[0], // modell file
                dir, // Output directory
                packagename
                );

        tree.apply(t);
        //newfile = t.getFilename();
        result = !t.hasErrorOccured();

    }
} catch (ParserException e) {
    System.out.println(f.getName() + " is not valid.");
    System.out.println("Error at: " + e.getMessage());

    result = false;
} catch (Exception e) {
    System.out.println(f.getName() + " is not valid.\n"
            + "Message: " + e.getMessage() + "\n"
            + e.getCause());
    result = false;
}

nameOfFailedRule = t.getCurrentRuleName();

if (result) {
    System.out.println("The rule definition file " + f.getName()
            + " is valid.");
} else {
    System.out.println("At least one rule in "
            + f.getName()
            + " is not valid.\n"
            + "The last rule handled was: " + nameOfFailedRule);
    System.exit(1);
}
}
}
}
```

G.2.2. RuleSemanticAnalyzer.java

```
package kiel.util.checkingHelpers.translator;

import kiel.util.checkingHelpers.synrule.node.*;
import kiel.util.checkingHelpers.synrule.analysis.*;
import java.util.Hashtable;

/**
 * <p>Description: This is a semantic analyzer which is used when reading
 * the rules. It checks, wether the used identifiers have been correctly
 * declared, or not.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/10/31 09:34:51 $
 */
public class RuleSemanticAnalyzer extends DepthFirstAdapter {

    /**
     * The hashtable containing the declared identifier of a rule.
     */
    private Hashtable symbolTable = new Hashtable();

    /**
     * The field containing the count of the problems in one rule.
     */
    private int problemCount = 0;

    /**
     * The event handler for nodes of type AMessageDefinition.
     * @param node The current instance of a message definition.
     */
    public final void inAMessageDefinition(final AMessageDefinition node) {
        TIdentifier id = node.getIdentifier();

        String key = id.getText();
        if (symbolTable.containsKey(key.toUpperCase())) {
            System.out.println("Error: [" + id.getLine() + "," + id.getPos()
                    + "] Identifier '" + key + "' already defined");
            problemCount++;
        } else {
            key = key.toUpperCase();
            symbolTable.put(key, key);
        }

    }

    /**
     * The event handler that is called, when an if then statement in the
     * rule is visited.
     * @param node The if then statement.
     */
    public final void inAIfThenStatement(final AIfThenStatement node) {
        TIdentifier id = node.getIdentifier();

        String key = id.getText();

        if (!symbolTable.containsKey(key.toUpperCase())) {
            System.out.println("Error: [" + id.getLine() + "," + id.getPos()
                    + "] undeclared Identifier: " + key);
            problemCount++;
        }

    }

    /**
     * The event handler, which is called, when a rule is left.
     * @param node The node of the rule.
     */
    public final void outARule(final ARule node) {

        symbolTable.clear();

    }

    /**
     * Getter for the problem count.
     * @return The count of found problems.
     */
    public final int getProblemCount() {
        return problemCount;
    }

    /**
     * Setter for the field problem count.
     * @param value The actual value
     */
    public final void setProblemCount(final int value) {
        this.problemCount = value;
    }

}
```

G.2.3. Translator.java

```
package kiel.util.checkingHelpers.translator;

import kiel.util.checkingHelpers.synrule.node.*;
import kiel.util.checkingHelpers.synrule.analysis.*;
//import kiel.dataStructure.Node;
//import kiel.dataStructure.State;

import java.io.*;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.LinkedList;

import org.xml.sax.SAXException;

import tudresden.ocl.OclTree;
import tudresden.ocl.check.OclTypeException;
import tudresden.ocl.check.types.*;
import tudresden.ocl.check.types.xmifacade.XmiParser;
import tudresden.ocl.codegen.CodeFragment;
import tudresden.ocl.codegen.JavaCodeGenerator;


/**
 * <p>Description: This is the class, that does the java code generation and
 * calls the ocl to java code compiler of the Dresden OCL Toolkit.
 * It extends the DepthFirstAdapter class generated by SableCC which implements
 * a Visitor Design Pattern to handle all nodes of the parsed abstract syntax
 * tree of a rule.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.11 $ last modified $Date: 2006/10/31 09:34:52 $
 */
public class Translator extends DepthFirstAdapter {

    /**
     * The currently created file.
     */
    private String filename;

    /**
     * The meta modell file used for the ocl compiler.
     */
    private String model;

    /**
     * The name of the package for all created classes.
     */
    private String packagename;

    /**
     * The directory where all java classes will be saved into.
     */
    private String outputDir;

    /**
     * Internal field for correct indentation.
     */
    private int tabcounter = 0;

    /**
     * The writer used to create the class files.
     */
    private BufferedWriter out = null;

    /**
     * If the currently parsed rule is an error, this field is true.
     * It is used for correct internal handling of the message generation
     * in the generated test.
     */
    private boolean isError;

    /**
     * Internal handler needed for correct translation of found
     * if then else expressions.
     */
    private boolean isOnlyExpression;

    /**
     * This list contains all those identifiers, that are declared
     * as valid context.
     */
    private ArrayList context;

    /**
     * True, if an error occured.
     */
    private boolean errorOccured;


    /**
     * Contains the name of the currently handled rule.
     * Used for error tracing.
     */
    private String currentRuleName;


    /**
     * The constructor for the Translator class.
     * @param modelfile The file denoting the xmi model.
     * @param outputdir The directory to write the generated file to.
     * @param pname The name of the package where the generated java classes
     * will be inserted into.
     */
    public Translator(final String modelfile, final String outputdir,
            final String pname) {
        super();
        this.model = modelfile;
        this.packagename = pname;
```

```
        this.outputDir = outputdir;
        isError = false;
        context = new ArrayList();
        isOnlyExpression = false;
        errorOccured = false;
    }

    /**
     * Setter for the name of the java source file to be created.
     * @param file The filename.
     */
    public final void setFilename(final String file) {
        this.filename = file;
    }

    /**
     * The name of the currently created java class file.
     * @return The name of the class file.
     * Equal to the name of the rule in progress..
     */
    public final String getFilename() {
        return filename;
    }

    /**
     * Internal handler for pretty printing the source code.
     * @return A String containing of a number of tab characters.
     */
    private String getTabs() {
        String result = "";
        for (int i = 0; i < tabcounter; i++) {
            result = result + "\t";
        }
        return result;
    }

    /**
     * Internal handler for writing a passed line of text to a generated file.
     * It uses the getTabs method to supply the correct indentation.
     * @param line The line of text to be written.
     */
    private void writelnToFile(final String line) {
        try {
            out.write(getTabs() + line);
            out.newLine();
        } catch (IOException e) {
            errorOccured = true;
            e.printStackTrace();
        }
    }

    /**
     * Internal handler for writing a string to a generated file.
     * No layouting is performed.
     * @param s The text to be written.
     */
    private void writeToFile(final String s) {
        try {
            out.write(s);
        } catch (IOException e) {
            errorOccured = true;
            e.printStackTrace();
        }

    }

    /**
     * Opens and prepares a file to be written to.
     * @param fn The name of the file to get opened.
     */
    private void openFile(final String fn) {
        try {
            out = new BufferedWriter(
            new OutputStreamWriter(
                new FileOutputStream(outputDir + fn + ".java")));

        } catch (IOException e) {
            errorOccured = true;
            e.printStackTrace();
        }
    }

    /**
     * This method closes the current file and writes all performed changes
     * to the disc.
     */
    private void closeFile() {
        try {
            out.flush();
            out.close();
        } catch (IOException e) {
            errorOccured = true;
            e.printStackTrace();
        }
    }

    /**
     * Internal handler for writing the source code header.
     */
    private void writeHeader() {
        // packagename
        writelnToFile("package " + packagename + ";");
        writelnToFile("");

        // import for the ocl library
        writelnToFile("import tudresden.ocl.lib.*;");

        // import for the kiel dataStructure
        writelnToFile("import kiel.dataStructure.*;");
        // import for the checker framework
        writelnToFile("import kiel.checking.*;");
        writelnToFile("");
    }

    /**
     * Event handler for the visitor, which is called,
     * when a rule heading is visited.
     * It opens the file given by the name of the rule,
     * writes the default header and inserts the class header.
     * @param node The current rule heading.
```

```
     */
    public final void inARuleHeading(final ARuleHeading node) {
        filename = node.getIdentifier().getText();
        currentRuleName = filename;
        openFile(filename);
        System.out.println("Translating rule: " + filename);
        writeHeader();
        writelnToFile("public class " + filename + " extends BaseCheck {");
        tabcounter++;
        // add the field for the chart
        writelnToFile("private StateChart privateChart = null;");
    }

    /**
     * Event handler for the visitor, that is called, when the
     * handling of a rule is finished.
     * It calls <code>closeFile</code> to write the file to the disc.
     * @param node The rule.
     */
    public final void outARule(final ARule node) {
        tabcounter--;
        writelnToFile("}");
        closeFile();
        context.clear();
    }

    /**
     * Event handler for the visitor, that is called, when a declaration part
     * is found.
     * @param node The found declaration part of the current rule.
     */
    public final void inADeclPart(final ADeclPart node) {
        writelnToFile("");
    }

    /**
     * Event handler for a found definition of a message.
     * It adds the declaration of the message to the generated file.
     * @param node The handled message definition node.
     */
    public final void inAMessageDefinition(final AMessageDefinition node) {
        String message = node.getString().getText();
        writelnToFile("private String " + node.getIdentifier().getText()
                + " = " + message + ";");
    }

    /**
     * Handler for nodes that represent the beginning of expressions containing
     * at least one identifier.
     * @param node The identifier expression.
     */
    public final void inAIdentifierExpr(final AIdentifierExpr node) {
        String code = "";
        LinkedList list = node.getNot();
        for (int i = 0; i < list.size(); i++) {
            code += "!";
        }

        if (list.size() == 0) {
            context.add(node.getIdentifier().getText());
        }

        code += "(o instanceof " + node.getIdentifier().getText() + ")";
        writeToFile(code);
    }

    /**
     * Handler for succeding parts of expressions containing at least two
     * identifier. This handler contains the correct mapping of the logical
     * <b>or</b> and <b>and</b> operators to the equivalent java operators.
     * @param node The tail of an identifier expression.
     */
    public final void inAIdentifierListTail(final AIdentifierListTail node) {
        TBooleanops b = node.getBooleanops();
        String op = null;

        if (b.getText().equalsIgnoreCase("or")) {
            op = "||";
        }
        if (b.getText().equalsIgnoreCase("and")) {
            op = "&&";
        }

        writeToFile(" " + op + " ");
    }

    /**
     * The message handler for entering the context part of a rule.
     * It begins to write the method <code>isValidTarget()</code> of
     * a check to the file.
     * @param node The context part.
     */
    public final void inAContextPart(final AContextPart node) {
        writelnToFile("");
        writelnToFile("public final boolean "
                + "isValidTarget(final Object o) {");
        tabcounter++;
        writeToFile(getTabs() + "return ");
    }

    /**
     * The message handler, that is called when the context part is left.
     * It finishes the method <code>isValidTarget</code>, that was begun
     * by inAContextPart.
     * @param node The left context part.
     */
    public final void outAContextPart(final AContextPart node) {
        writeToFile(";");
        writelnToFile("");
        tabcounter--;
        writelnToFile("}");
    }

    /**
     * Internally used handler for mapping the method parentStates
     * to the correct method supplied by the check itself and not by the
     * GraphicalObject.
     * The method <code>getParentStates</code>was introduced by the
     * Meta modell of kiel.dataStructure and is used to access all parents of
     * a node.
```

```
     * @param code The code snippet generated by the Dresden OCL Toolkit
     * @return A modified version of the input.
     */
    private String handleIteratingCode(final String code) {

        String result = code;
        int index = code.indexOf(".getFeature(\"parentStates\"));");

        if (index >= 0) {
            String part1 = code.substring(0, index);
            part1 = part1.substring(0, part1.lastIndexOf('=') + 1);
            String part2 = code.substring(index);
            part2 = part2.substring(part2.indexOf(';'));

            result = part1
            + "Ocl.getOclSequenceFor(getParentStates(o))"
            + part2;
        }

        return result;
    }

    private String handleChartCode(final String code) {
        String result = code;
        int index = code.indexOf(".getFeature(\"chart\"));");

        if (index >= 0) {
            String part1 = code.substring(0, index);
            part1 = part1.substring(0, part1.lastIndexOf('=') + 1);
            String part2 = code.substring(index);
            part2 = part2.substring(part2.indexOf(';'));

            result = part1
            + "Ocl.toOclAnyImpl(Ocl.getFor(privateChart))"
            + part2;
        }

        return result;
    }


    /**
     * Internally used method to map the property isConnectedToInitial to the
     * method of the Basecheck.
     * The method <code>isConnectedToInitial</code>was introduced by the
     * Meta modell of kiel.dataStructure and is used to check, if a node a
     * path from the initial state to the node exists.
     * @param code The code snippet generated by the Dresden OCL Toolkit.
     * @return A modified version of the input.
     */
    private String handleConnectingCode(final String code){

        String result = code;

        int index = code.indexOf(".getFeature(\"isConnectedToInitial\"));");

        if (index >= 0) {
            String part1 = code.substring(0, index);
            part1 = part1.substring(0, part1.lastIndexOf('=') + 1);
            String part2 = code.substring(index);
            part2 = part2.substring(part2.indexOf(';'));

            result = part1
            + "Ocl.getFor(isConnectedToInitial((Node) o))"
            + part2;
        }

        return result;
    }

    /**
     * Internally used method for handling a constraint definition.
     * This method actually uses the Dresden OCL Toolkit compiler and code
     * generator to translate the given ocl constraint into java source code.
     * @param constraint The extracted OCL constraint.
     */
    private void handleConstraint(final String constraint) {
        ModelFacade mf;
        OclTree tree;

        File f = new File(model);

        final URL xmi;
        try {
            xmi = f.toURL();

            mf = XmiParser.getModel(xmi, "");
            // needed modifications for the constraint itself
            String cons = "context ";
            cons += (String) context.get(0);
            cons += " inv: ";
            cons += constraint.substring(1, constraint.length() - 1);
            // calls the ocl parser of the Dresden OCL toolkit to generate
            // the abstract syntax tree representing the input constraint.
            tree = OclTree.createTree(cons, mf);
            tree.applyDefaultNormalizations();
            JavaCodeGenerator jcg = new JavaCodeGenerator();

            CodeFragment[] frags = tree.getCode(jcg);
            String code;
            // because only invariants are handled, only one
            // fragment is returned
            for (int i = 0; i < frags.length; i++) {
                //frags[i].getResultVariable()
                code = frags[i].getCode();

                // replace the occurences of this by o
                code = code.replaceAll("this", "o");
                // insert tabs at the beginning of each line
                code = code.replaceAll("\n", "\n" + getTabs());

                // replace iterating method stubs
                int index = code.indexOf(".getFeature(\"parentStates\"));");
                while (index > 0) {
                    code = handleIteratingCode(code);
                    index = code.indexOf(".getFeature(\"parentStates\"));");
                }

                // replace other method stubs
                index = code.indexOf(".getFeature(\"isConnectedToInitial\"));");
```

```
                while (index > 0) {
                    code = handleConnectingCode(code);
                    index = code.indexOf(".getFeature(\"isConnectedToInitial\"));");
                }

                index = code.indexOf(".getFeature(\"chart\"));");
                while (index > 0) {
                    code = handleChartCode(code);
                    index = code.indexOf(".getFeature(\"chart\"));");
                }

                writelnToFile(code);
                writelnToFile("return " + frags[i].getResultVariable()
                        + ".isTrue();");
            }
        } catch (OclTypeException e2) {
            System.out.println("Handling the constraint of rule "
                    + currentRuleName
                    + " failed.\n"
                    + e2.getMessage());
            errorOccured = true;
        } catch (MalformedURLException e1) {
            System.out.println("A passed name of a file could not be "
                    + "transformed into a URL.");
            errorOccured = true;
        } catch (SAXException e) {
            System.out.println("An error occured while reading the meta "
                    + "modell file.");
            errorOccured = true;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            System.out.println("A general IOException occured.");
            errorOccured = true;
        }
    }

    /**
     * The handler that is called by the visitor design pattern when a
     * constraint node is entered.
     * This method writes the full <code>apply</code> method to the check and
     * starts the <code>eval</code> method.
     * @param node The constraint.
     */
    public final void inAConstraintPart(final AConstraintPart node) {
        // this part has to be generated by maybe Dresden OCL toolkit
        writelnToFile("");
        writelnToFile("public final void apply("
                + "final StateChartCheckerBase checker,"
                + " final Object o) {");
            tabcounter++;
        writelnToFile("privateChart = checker.getStateChart();");
        writelnToFile("// Message producing code");
        writelnToFile("if (eval(o)) {");
        tabcounter++;
        // make generateConformsMessage a default method in BaseCheck
        // that prevents an error, if no conforms part was specified within the
        // rule
        writelnToFile("generateConformsMessage(checker, o);");
        tabcounter--;
        writelnToFile("} else {");
        tabcounter++;
        writelnToFile("generateFailsMessage(checker, o);");
        tabcounter--;
        writelnToFile("}");
        tabcounter--;
        writelnToFile("}");

        writelnToFile("");
        writelnToFile("private final boolean eval(final Object o) {");
        tabcounter++;
        writelnToFile("// ocl to java translated");

        handleConstraint(node.getString().getText());
        // writelnToFile(getTabs() + node.getString().getText());
        tabcounter--;
        writelnToFile("}");
    }

    /**
     * Handler for the visitor, that is called when a conforms part is entered.
     * It starts to write the method <code>generateConformsMessage</code>
     * to the check.
     * @param node The entered node.
     */
    public final void inAConformsReturnPart(final AConformsReturnPart node) {
        writelnToFile("");
        writelnToFile("public final void "
                + "generateConformsMessage(final StateChartCheckerBase checker,"
                + " final Object o) {");
        tabcounter++;
    }

    /**
     * This method is called when the visitor leaves the conforms returns part.
     * It finishes the <code>generateConformsMessage</code> method of the check.
     * @param node The left node.
     */
    public final void outAConformsReturnPart(final AConformsReturnPart node) {
        tabcounter--;
        writelnToFile("}");
    }

    /**
     * The method for generating the fails message.
     * The method <code>generateFailsMessage</code> is added to the
     * generated java class.
     * The method is called when the declaration node is visited.
     * @param node The node from the abstract syntax tree.
     */
    public final void inAFailsReturnPart(final AFailsReturnPart node) {
        writelnToFile("");
        writelnToFile("public final void "
                + "generateFailsMessage(final StateChartCheckerBase checker, "
                + "final Object o) {");
        tabcounter++;
    }

    /**
     * This method ends the <code>generateFailsMessage</code>
```

```
     * when the handling of the declared fails part is finished.
     * @param node The node of the syntax tree.
     */
    public final void outAFailsReturnPart(final AFailsReturnPart node) {
        tabcounter--;
        writelnToFile("}");
    }

    /**
     * Handles an if-then-statement.
     * The method is called when the node is entered.
     * It writes <code>if (</code> to the generated class.
     * @param node The node representing the if then statement.
     */
    public final void inAIfThenStatement(final AIfThenStatement node) {
        writeToFile(getTabs() + "if (");
    }

    /**
     * Handles an if-then-statement.
     * The method is called when the node is left.
     * It writes code that generates the problem and that adds it to the
     * correct list of the checker depending on the value of isError.
     * It calls the method <code>generateMessageHandling</code> to write
     * the code, that takes care of the property handling.
     * @param node The node representing the if then statement.
     */
    public final void outAIfThenStatement(final AIfThenStatement node) {
        writelnToFile("");

        String code = "checker.";

        if (isError) {
            code += "getErrors().";
        } else {
            code += "getWarnings().";
        }

        /*
         * Parameter Prefix the message with NodeType, NodeName and ID
         * if the corresponding properties are set
         */

        generateMessageHandling();
        writelnToFile("String mes = "
            + node.getIdentifier().getText() + ";");
        writelnToFile("mes = head + \": \" + mes;");

        code += "add(new CheckingProblem(o, mes));";
        tabcounter++;
        writelnToFile(code);
        tabcounter--;

        writelnToFile("}");
    }

    /**
     * Internally used method generating the code that handles the message
     * generation of a problem.
     * Called by {@link outAIfThenStatement}.
     */
    private void generateMessageHandling() {
        /*
         * Parameter Prefix the message with NodeType, NodeName and ID
         * if the corresponding properties are set
         */
        writelnToFile("String head = \"\";");
        writelnToFile("String s = o.getClass().getName();");
        writelnToFile("GraphicalObject go = null;");
        writelnToFile("if (o instanceof GraphicalObject) {");
        tabcounter++;
        writelnToFile("go = (GraphicalObject) o;");
        tabcounter--;
        writelnToFile("}");

        writelnToFile("if (CheckingProperties.showRuleName()) {");
        tabcounter++;
        writelnToFile("head = \"[\" + this.getRuleName() + \"] \";");
        tabcounter--;
        writelnToFile("}");


        writelnToFile("if (o instanceof Node) {");
        tabcounter++;
        writelnToFile("Node n = (Node) o;");

        writelnToFile("if (CheckingProperties.showNodeName()) {");
        tabcounter++;
        writelnToFile("head += s.substring(s.lastIndexOf('.') + 1);");
          writelnToFile("if (!n.getName().equals(\"\")) {");
          tabcounter++;
          writelnToFile("head += \" \\\"\" + n.getName() + \"\\\"\";");
          tabcounter--;
          writelnToFile("}");
        tabcounter--;
        writelnToFile("}");
        tabcounter--;
        writelnToFile("}");

        writelnToFile("if (CheckingProperties.showNodeID()) {");
        tabcounter++;
        writelnToFile("if (go != null) {");
        tabcounter++;
        writelnToFile("head += \" (ID: \" + go.getID() + \")\";");
        tabcounter--;
        writelnToFile("}");
        tabcounter--;
        writelnToFile("}");

    }

    /**
     * The method is called when an if then else statement is left.
     * @see outAIfThenStatement
     * @param node The node representing the left if-then-else-statement.
     */
    public final void outAIfThenElseStatement(
            final AIfThenElseStatement node) {
        writelnToFile(" else {");
        String code = "checker.";
```

```
        if (isError) {
            code += "getErrors().";
        } else {
            code += "getWarnings().";
        }

        tabcounter++;
        generateMessageHandling();

        writelnToFile("String mes = "
                + node.getIdentifier().getText() + ";");
        writelnToFile("mes = head + \": \" + mes;");

        code += "add(new CheckingProblem(o, "
            + node.getIdentifier().getText() + "));";
        tabcounter++;
        writelnToFile(code);
        tabcounter--;
        tabcounter--;
        writelnToFile("}");
    }


    /**
     * Event handler, when a simple expression is entered.
     * If <code>isOnlyExpression</code> is true, the identifier
     * is written to the file.
     * In all other cases the expression is part of an condition expression
     * and an <code>o instanceof ...</code> statement is written to the class.
     * @param node The expression.
     */
    public final void inASimpleExpression(final ASimpleExpression node) {
        if (isOnlyExpression) {
            writeToFile(node.getIdentifier().getText());
        } else {
            writeToFile("o instanceof "
                    + node.getIdentifier().getText() + ") {");
        }
    }


    /**
     * Event handler, if the message to be returned for the problem
     * is declared without using an condition.
     * Sets <code>isOnlyExpression</code> true.
     *
     * @param node The statement.
     */
    public final void inADirectEvalStatementPart(
            final ADirectEvalStatementPart node) {
        isOnlyExpression = true;
        tabcounter++;

        generateMessageHandling();
        writeToFile(getTabs() + "String mes = ");
    }


    /**
     * Finishes the code of {@link inADirectEvalStatementPart} and
     * adds the code to add the problem to the list of the checker instance.
     * This method is called, when the statement is left.
     * @param node The left direct eval statement part.
     */
    public final void outADirectEvalStatementPart(
            final ADirectEvalStatementPart node) {
        String code = "checker.";

        if (isError) {
            code += "getErrors().";
        } else {
            code += "getWarnings().";
        }
        writeToFile(";");
        writelnToFile("");


        writelnToFile("mes = head + \": \" + mes;");

        code += "add(new CheckingProblem(o, mes));";
        writeToFile(getTabs() + code);

        writelnToFile("");
        tabcounter--;
        isOnlyExpression = false;
    }


    /**
     * This method is called, if <code>Error;</code> was parsed in the rule.
     * It sets the field isError to true.
     * @param node The node in the syntax tree currently handled.
     */
    public final void inAIsErrorPart(final AIsErrorPart node) {
        isError = true;
    }


    /**
     * Public getter for the field errorOccured.
     * @return True if an error has occured while parsing the rules file.
     */
    public final boolean hasErrorOccured() {
        return errorOccured;
    }

        public String getCurrentRuleName() {
                return currentRuleName;
        }

}
```

G.3. Checking-Plug-In

G.3.1. StateChartCheckerBase.java

```
//$Id: StateChartCheckerBase.java,v 1.22 2006/11/08 10:13:03 kbe Exp $
package kiel.checking;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Writer;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.jar.JarFile;
import java.util.zip.ZipEntry;

import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JMenu;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;

import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.Transition;
import kiel.util.IStateChartVisitor;
import kiel.util.KielFrameRefresh;
import kiel.util.LogFile;
import kiel.util.StateChartDepthFirstAdapter;

/**
 *<p> Description: This class manages the checking of a statechart. It uses a
 * {@link kiel.util.StateChartDepthFirstAdapter} to visit all nodes of the
 * data structure.
 * The checks to be performed are loaded dynamically.</p>
 *
 * <p> Copyright: (c) 2006 </p>
 * <p> Company: Uni Kiel </p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.22 $ last modified $Date: 2006/11/08 10:13:03 $
 */
public class StateChartCheckerBase extends AbstractTableModel implements
       IStateChartVisitor {

   /**
    * Internal used id for serializing.
    */
   private static final long serialVersionUID = -4071513810514182545L;

   /**
    * The depth first adapter to visit all nodes of the datastructure.
    */
   private StateChartDepthFirstAdapter depthfirst;

   /**
    * Property to decide wether the checker should dispatch regions or not.
    */
   private boolean dispatchRegions = true;

   /**
    * Holds instances of dynamically loaded checks.
    */
   private ArrayList rules;

   /**
    * Stores the found errors.
    */
   private ArrayList errors;

   /**
    * Stores generated warnings.
    */
   private ArrayList warnings;

   /**
    * Directory containing the jar file with the checks to be loaded.
    */
   private String rulePackage;

   /**
    * Internal field for future usage of different robustness levels.
    */
   private int robustnessLevel = 0;

   /**
    * True in GUI version, false in commandline version.
    */
   private boolean showResultsInGUI = false;

   /**
    * The instance of the gui where the results are presented.
    */
   private CheckerOutputGUI gui;

   /**
    * The index of the tab on which the gui is located on in the KielFrame.
    */
```

```
 private int tabbedIndex;

 /**
  * The cvclProcess.
  */
private Process cvclProcess;

/**
 * The instance of the Statechart which is checked.
 */
private StateChart stateChart;

/**
 * The field denotes wheter Transitions shall be checked or not.
 */
private boolean doDispatchTransitions;


//private boolean doCheckStateChartModelConformity;


/**
 * This field contains all rules for the current statechart dialect
 * from the properties.
 */
private String ruleSet;


/**
 * The menu entry in the KielFrame.
 */
private JMenu menu;


/**
 * This method tests if the projekt is running in cygwin.
 * This is important, due to problems with the file separator char
 * @return True if the projekt was started from cygwin
 */
public final boolean isRunningInCygwin() {
    Process p;
    try {
        p = Runtime.getRuntime().exec("uname -a");
        BufferedReader input =
            new BufferedReader(
                    new InputStreamReader(p.getInputStream()));

        String line = input.readLine();
        return line.toLowerCase().startsWith("cygwin");
    } catch (IOException e1) {
        return false;
    }

}

 /**
  * The constructor.
  *
  * @param dir The sourcedir of the jar file.
  * @param doShow True in GUI version, false in commandline version.
  */
public StateChartCheckerBase(final String dir, final boolean doShow) {
    KielFrameRefresh.register(this);
    cvclProcess = null;
    rules = new ArrayList();
    rulePackage = dir;
    errors = new ArrayList();
    warnings = new ArrayList();
    showResultsInGUI = doShow;
    doDispatchTransitions = false;
    //doCheckStateChartModelConformity = false;
    menu = null;
    if (showResultsInGUI) {
        gui = new CheckerOutputGUI(this);
        setTextInLabel();
    }
}

/**
 * The constructor with an extra process argument.
 *
 * @param dir The sourcedir of the jar file.
 * @param doShow True in GUI version, false in commandline version.
 * @param cvclProc The CVCl process.
 */
public StateChartCheckerBase(final String dir, final boolean doShow,
        final Process cvclProc) {
    KielFrameRefresh.register(this);
    cvclProcess = cvclProc;
    rules = new ArrayList();
    rulePackage = dir;
    errors = new ArrayList();
    warnings = new ArrayList();
    showResultsInGUI = doShow;
    doDispatchTransitions = false;
    menu = null;
    //doCheckStateChartModelConformity = false;
    if (showResultsInGUI) {
        gui = new CheckerOutputGUI(this);
        setTextInLabel();
    }
}

/**
 * Performs a depth first search to visit all subnodes of the passed node.
 *
 * @param chart The current StateChart.
 * @return Returns True if no errors and no warnings were found.
 */
public final boolean checkDataStructure(final StateChart chart) {
    errors.clear();
    warnings.clear();
    stateChart = chart;
    chart.setChanged();
    chart.getAllObjects();
    dispatchChart(chart);

    depthfirst = new StateChartDepthFirstAdapter();
    depthfirst.traverse(this, stateChart.getRootNode(), dispatchRegions);
    if (showResultsInGUI) {
```

```
            this.fireTableDataChanged();
            setTextInLabel();
        }
        return ((getErrors().size() == 0) && (getWarnings().size() == 0));
    }


    private final void dispatchChart(final StateChart chart) {
        BaseCheck rule;
        for (int i = 0; i < rules.size(); i++) {
            rule = (BaseCheck) rules.get(i);
            if ((rule.isEnabled())
                && (rule.isValidTarget(chart))) {
                rule.apply(this, chart);
            }
        }
    }

    /**
     * @param o The object to perform all checks on.
     */
    public final void dispatch(final GraphicalObject o) {
        BaseCheck rule;
        for (int i = 0; i < rules.size(); i++) {
            rule = (BaseCheck) rules.get(i);
            if ((rule.isEnabled())
                && (rule.isValidTarget(o))) {
                rule.apply(this, o);
            }

            if ((doDispatchTransitions)
                && (o instanceof Node)) {
                Node n = (Node) o;

                Iterator iter = n.getOutgoingTransitions().iterator();

                while (iter.hasNext()) {
                    Transition t = (Transition) iter.next();
                    if (rule.isValidTarget(t)
                        && (rule.isEnabled())) {
                        rule.apply(this, t);
                    }
                }
            }
        }
    }

    /**
     * Returns the type of the StateChart.
     * @return The type of the StateChart.
     */
    public final String getStateChartType() {
        return stateChart.getModelSource();
    }

    /**
     * Returns the model version of the currently checked statechart.
     * @return The version number string.
     */
    public final String getStateChartVersion() {
        return stateChart.getModelVersion();
    }

    /**
     * @return The collection of all warnings that occured during the last run
     * of checkDataStructure.
     */
    public final Collection getWarnings() {
        return warnings;
    }

    /**
     *
     * @return The collection of all errors that occured during the last run of
     * checkDataStructure.
     */
    public final Collection getErrors() {
        return errors;
    }

    /**
     *
     * @return Does the actual instance dispatch objects of type Region?
     */
    public final boolean isDispatchRegions() {
        return dispatchRegions;
    }

    /**
     *
     * @param doDispatch Denote if the instance should dispatch Region.
     */
    public final void setDispatchRegions(final boolean doDispatch) {
        this.dispatchRegions = doDispatch;
    }

    /**
     * This method loads the rules from the directory passed to the constructor.
     * The rules are dynamically load from a jar file and instances of the found
     * checks are stored in the
     */
    public final void loadRules() {
        rules.clear();

        File f = new File(this.rulePackage);

        if (!f.exists()) {
            CheckingProperties.getRobustnessLog().log(LogFile.ALL,
                   "Passed package " + this.rulePackage + " does not exist."
                  + "No rules loaded.");
            return;
        } else {
            CheckingProperties.getRobustnessLog().log(LogFile.DETAIL,
                   "Starting to load rules from: " + this.rulePackage);
        }

        String classname = "";
        Class c;
        Object o;
        JarFile jar = null;
```

```
        // Manifest m = null;
        try {
            jar = new JarFile(f);
            if (jar != null) {
                Enumeration entries = jar.entries();
                URL[] url = {(new File(jar.getName())).toURL()};
                URLClassLoader ucl = new URLClassLoader(url, this
                        .getClass().getClassLoader());
                while (entries.hasMoreElements()) {

                    classname = ((ZipEntry) entries.nextElement()).getName();
                    if ((classname.indexOf(".class") > -1)
                            && (classname.indexOf("$") == -1)) {

                        classname = classname.replaceAll("/", ".");

                    classname = classname.substring(0,
                            classname.lastIndexOf("."));
                    c = ucl.loadClass(classname);
                    if (c.getSuperclass() == BaseCheck.class) {
                        o = c.newInstance();
                        if (o instanceof BaseCheck && cvclProcess != null) {
                            ((BaseCheck) o).setCvclProcess(cvclProcess);
                        }
                        if (o != null) {
                            rules.add(o);
                            CheckingProperties.getRobustnessLog().log(
                                    LogFile.DETAIL,
                                    "Loading of rule:" + classname
                                    + " successful.");
                        }
                    }
                }

            } else {
                CheckingProperties.getRobustnessLog().log(LogFile.ERROR,
                        "No valid jarfile specified.");
            }
            //loadRuleStatuses();
        } catch (Exception e) {
            e.printStackTrace();
            CheckingProperties.getRobustnessLog().log(LogFile.ERROR,
                    "Rule loading failed.");
        }
    }

    /**
     * Loads the statuses of all rules in the current rulePackage.
     */
    /*private void loadRuleStatuses() {
        BaseCheck b;
        for (int i = 0; i < rules.size(); i++) {
            b = (BaseCheck) rules.get(i);
            b.setEnabled(CheckingProperties.getRuleStatus(
                    b.getClass().getName()));
/* b.setEnabled(CheckingProperties.getRuleStatus(
                this.rulePackage.substring(
                    this.rulePackage.lastIndexOf(File.separatorChar) + 1,
                    this.rulePackage.lastIndexOf('.'))
                        + "." + b.getRuleName()));
    */
    /* }
        }*/

        /**
         *
         * @return The level of semantic robustness checks
         */
        public final int getRobustnessLevel() {
            return robustnessLevel;
        }

        /**
         * Set the level of the checks manually.
         * @param level The level of the checks.
         */
        public final void setRobustnessLevel(final int level) {
            this.robustnessLevel = level;
        }

        /**
         * This method prints the errors and warnings, that were generated during
         * checkDataStructure.
         * @param w The destination writer, where the messages will be written to.
         **/
        public final void printMessages(final Object w) {

            CheckingProblem r;
            try {

                for (int i = 0; i < errors.size(); i++) {
                    r = (CheckingProblem) errors.get(i);
                    ((Writer) w).write("Error " + (i + 1) + ": "
                            + r.getProblem() + "\n");
                    //CheckingProperties.getSemLog().log(LogFile.ERROR,
                            //);
                }
                for (int i = 0; i < warnings.size(); i++) {
                    r = (CheckingProblem) warnings.get(i);
                    ((Writer) w).write("Warning " + (i + 1) + ": "
                            + r.getProblem() + "\n");
                }
            } catch (IOException e) {
                CheckingProperties.getRobustnessLog().log(LogFile.ERROR,
                        "Printing the problem failed.");
            }
        }

        /**
         * @return One column is needed for displaying the problems in a table.
         */
        public final int getColumnCount() {
            return 1;
        }

        /**
         * The count of rows is calculated by adding the size of the errors and
         * warnings vector.
         * @return The count of rows the table has to display.
```

```
     */
    public final int getRowCount() {
        return errors.size() + warnings.size();
    }

    /**
     * This method transforms the given cell coordinates to the appropriate
     * robustness problem and returns the message of the problem.
     * @param rowIndex The index of the row to be displayed
     * @param colIndex .
     * @return The text to be displayed in the cell with the given coordinates.
     */
    public final Object getValueAt(final int rowIndex, final int colIndex) {
        String result;
        CheckingProblem p;
        if (rowIndex >= errors.size()) {
            p = (CheckingProblem) (warnings.get(rowIndex - errors.size()));
            if (CheckingProperties.showClassification()) {
                if (CheckingProperties.showCounters()) {
                    result = "[Warning " + (rowIndex - errors.size() + 1)
                            + "] ";
                } else {
                    result = "[Warning] ";
                }
            } else {
                if (CheckingProperties.showCounters()) {
                    result = "[" + (rowIndex - errors.size() + 1) + "] ";
                } else {
                    result = "";
                }
            }
        } else {
            p = (CheckingProblem) (errors.get(rowIndex));
            if (CheckingProperties.showClassification()) {
                if (CheckingProperties.showCounters()) {
                    result = "[Error " + (rowIndex + 1) + "] ";
                } else {
                    result = "[Error] ";
                }
            } else {
                if (CheckingProperties.showCounters()) {
                    result = "[" + (rowIndex + 1) + "] ";
                } else {
                    result = "";
                }
            }
        }

        return result + p.getProblem();
    }

    /**
     * @param col .
     * @return The string, which is shown in the header of the outputtable.
     */
    public final String getColumnName(final int col) {
        return "Messages";
    }

    /**
     *
     * @param index The index of the Problem.
     * @return The Graphicalobject of the Problem.
     *
     */
    public final ArrayList getProblemsByIndex(final int index) {
        CheckingProblem p;

        if (index > (errors.size() + warnings.size())) {
            return null;
        } else {

            if (index < errors.size()) {
                p = (CheckingProblem) errors.get(index);
            } else {
                p = (CheckingProblem) warnings.get(index - errors.size());
            }

            return p.getObjects();
        }
    }

    /**
     *
     * @return The table if the gui is shown.
     */
    public final JTable getOutputTable() {
        if (showResultsInGUI) {
            return gui.getTable();
        } else {
            return null;
        }
    }

    /**
     *
     * @return The component from the gui.
     */
    public final JComponent getComponent() {
        return gui.getComponent();
    }

    /**
     * This method sets the warning and error messages.
     */
    private void setTextInLabel() {
        gui.setLabelText(getSummary());
    }

    /**
     * This method builds the warning and error messages.
     * @return The warning and error messages.
     */
    public final String getSummary() {
        String result = "";

        if (errors.size() == 1) {
            result = errors.size() + " Error, ";
        } else {
            result = errors.size() + " Errors, ";
```

```
            }
            if (warnings.size() == 1) {
                result += warnings.size() + " Warning";
            } else {
                result += warnings.size() + " Warnings";
            }

            return result;
        }

        /**
         *
         * @return Index of the appropriate tab in KIELFrame.
         */
        public final int getTabbedIndex() {
            return tabbedIndex;
        }

        /**
         * Setter for the rules.
         * @param ti The index of the tab.
         */
        public final void setTabbedIndex(final int ti) {
            this.tabbedIndex = ti;
        }

        /**
         * Getter for the rules.
         * @return The rules.
         */
        public final ArrayList getRules() {
            return (ArrayList) rules.clone();
        }

        /**
         * Enables or disables a rule.
         * @param ruleName The name of the rule.
         * @param isEnabled .
         */
        public final void setEnabled(final String ruleName,
                final boolean isEnabled) {
            BaseCheck item;
            JCheckBoxMenuItem mItem;
            for (int i = 0; i < rules.size(); i++) {
                item = (BaseCheck) rules.get(i);
                if (item.getRuleName().equals(ruleName)) {
    /* CheckingProperties.setRuleStatus(this.rulePackage.substring(
                            this.rulePackage.lastIndexOf(File.separatorChar) + 1,
                            this.rulePackage.lastIndexOf('.'))
                            + "." + ruleName, isEnabled);*/
                    item.setEnabled(isEnabled);

                    if (menu != null) {
                        mItem = (JCheckBoxMenuItem) menu.getItem(i + 2);
                        mItem.setSelected(isEnabled);
                    }

                }
            }
        }

        /**
         * Clears the errors and warnings in the table.
         */
        public final void refresh() {
            errors.clear();
            warnings.clear();
            if (showResultsInGUI) {
                    this.setTextInLabel();
                    this.fireTableDataChanged();
            }
        }

        /**
         * Clears all user input.
         */
        public final void clearUserInput() {
            JTable table = getOutputTable();
            table.clearSelection();
        }


        /**
         * Setter for the property DoDispatchTransitions.
         * If the property is set to true, the loaded checks will also be
         * applied to transitions.
         * @param doDispatch Boolean value, denoting if
         * transitions will be checked also.
         */
        public final void setDoDispatchTransitions(final boolean doDispatch) {
            this.doDispatchTransitions = doDispatch;
        }


        /**
         * Messagehandler for the message, that a new statechart was loaded.
         * @param sc The new statechart
         */
        public final void newStateChartLoaded(final StateChart sc) {
            //System.out.println("New Statechart loaded");
            String model = sc.getModelSource() + sc.getModelVersion();
            // replace all whitespaces
            model = model.replaceAll(" ", "");
            // all other charakters that needed to be replaced
            model = model.replaceAll("[-|+|*|.|#|'|/|(|)]", "");

            String rule = CheckingProperties.modelRuleSet(model);
            // System.out.println("Loading ruleset for " + model + " ...");
            // System.out.println(rule);
            if (rule == null) {
                rule = "";
            }
            if (rule.equals("")) {
                // No Ruleset for the actual Statechart available
                // enable all rules by default
                for (int i = 0; i < rules.size(); i++) {
                    BaseCheck c = (BaseCheck) rules.get(i);
```

```
                this.setEnabled(c.getRuleName(), true);
            }
        } else {
            // if the rule is in the ruleset, enable it.
            for (int i = 0; i < rules.size(); i++) {
                BaseCheck c = (BaseCheck) rules.get(i);
                this.setEnabled(c.getRuleName(),
                        rule.indexOf(c.getRuleName()) > -1);
            }
        }

    }


    /**
     * Setter for the main menu item that reflects the statuses of
     * the loaded rules.
     * @param m A menu, that has the rules as items
     */
    public final void setMenu(final JMenu m) {
        this.menu = m;
    }

    /**
     * Get the chart currently being checked.
     * @return A Statechart.
     */
    public final StateChart getStateChart() {
        return this.stateChart;
    }


    /**
     * Use this method to enable loaded rules according to a profile.
     * @param profile A string containg the rules to enable separated by
     * comma.
     */
    public final void applyRuleProfile(final String profile) {

        // disable all rules at first
        for (int i = 0; i < rules.size(); i++) {
            BaseCheck c = (BaseCheck) rules.get(i);
                setEnabled(c.getRuleName(), false);
        }

        // now enable the rules found in the profile
        for (int i = 0; i < rules.size(); i++) {
            BaseCheck c = (BaseCheck) rules.get(i);
            this.setEnabled(c.getRuleName(),
                    profile.indexOf(c.getRuleName()) > -1);
        }
    }
}
```

G.3.2. BaseCheck.java

```
//$Id: BaseCheck.java,v 1.10 2006/08/04 09:54:13 kbe Exp $
package kiel.checking;

import java.util.ArrayList;
import java.util.Iterator;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.Transition;




/**
 * <p>Description: This interface defines a method to be implemented by all
 * custom robustness checks.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.10 $ last modified $Date: 2006/08/04 09:54:13 $
 *
 */
public abstract class BaseCheck {

    /**
     * True, if the check is enabled.
     */
    private boolean enabled = true;

    /**
     * This method is called from an instance of
     * {@link kiel.checking.StateChartCheckerBase}.
     * @param checker The checker where the
     * {@link kiel.checking.CheckingProblem} will be
     * added.
     * @param o The current object in the depth first search of the statechart.
     */
    public abstract void apply(StateChartCheckerBase checker,
            Object o);

    /**
     * Determines if this rule should be invoked on the
     * current {@link kiel.dataStructure.GraphicalObject}.
     * @param o The current object in the depth first traversal.
     * @return Returns true if the rule should be applied to object o.
     */
    public abstract boolean isValidTarget(final Object o);

    /**
     * .
     * @param isEnabled .
     */
    public final void setEnabled(final boolean isEnabled) {
        this.enabled = isEnabled;
    }

    /**
     * .
     * @return True, if the check is enabled.
     */
    public final boolean isEnabled() {
        return this.enabled;
    }

    /**
     * Returns the name of the rule.
     * @return The name of the rule.
     */
    public final String getRuleName() {
        String name;
        name = this.getClass().getName();
        return name.substring(name.lastIndexOf('.') + 1);
    }

    /**
     * Placeholder for the autogenerated checks.
     * Called if a statechart conforms to the check.
     * @param o The source of a problem
     * @param checker The instance of the checker that holds the problems.
     */
    public void generateConformsMessage(final StateChartCheckerBase checker,
            final Object o) {

    }

    /**
     * Placeholder for the autogenerated checks. Called if a check failed.
     * @param o The source of a problem.
     * @param checker The instance of the checker that holds the error.
     */
    public void generateFailsMessage(final StateChartCheckerBase checker,
            final Object o) {

    }

    /**
     * Sets the robustness properties.
     * @param props The robustness properties to be set.
     */
    public void setProperties(final CheckingProperties props) {

    }

    /**
     * Sets the CVCL process.
     * @param cvclProc The process object.
     */
    public void setCvclProcess(final Process cvclProc) {
```

```
    }


    /**
     * This method will get all parents of the passed node.
     * @param n The node to collect all parents for.
     * @return A list of all nodes, that are parent nodes of the passed node.
     */
    public final ArrayList getParentStates(final Object n) {
        ArrayList result = new ArrayList();

        if (n instanceof Node) {

            CompositeState c = ((Node) n).getParent();
            while (c != null) {
                result.add(c);
                c = c.getParent();
            }
        }

        return result;
    }

    /**
     * This is only a method stub.
     * Can be used to fill with an algorithm to get all children of a node.
     * @param n The node to collect all child nodes for.
     * @return A list of all children nodes.
     */
    public final ArrayList getChildStates(final Node n) {
        // TODO Evaluate if the method is needed
        return null;
    }


    /**
     * This method checks, whetern a path from the passed Node to the
     * Initial State of the same Region respektivly ORState exists.
     * @param n The node to check if it can be reached from the InitialState.
     * @return True, if a connection exists, false otherwise.
     */
    public final boolean isConnectedToInitial(final Node n) {

        ArrayList white = new ArrayList();
        ArrayList black = new ArrayList();

        white.add(n);
        Node wNode = null;
        Transition t = null;
        Iterator iter = null;

        while (white.size() > 0) {
               wNode = (Node) white.get(0);

               if (wNode instanceof InitialState) {
                      return true;
               }

               white.remove(0);
               if (black.indexOf(wNode) == -1) {
                      black.add(wNode);
               }

               iter = wNode.getIncomingTransitions().iterator();
               while (iter.hasNext()) {
                      t = (Transition) iter.next();
                      if ((white.indexOf(t.getSource()) == -1)
                                    && (black.indexOf(t.getSource()) == -1)) {
                             white.add(t.getSource());
                      }
               }
        }
        return false;
    }

}
```

G.3.3. CheckerOutputGUI.java

```
//$Id: CheckerOutputGUI.java,v 1.1 2006/04/10 10:22:22 gsc Exp $
package kiel.checking;

import java.awt.BorderLayout;

import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;

/**
 * <p>Description: This class displays the problems found by a robustness
 * checker.</p>
 *
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 *
 * @version $Revision: 1.1 $ last modified $Date: 2006/04/10 10:22:22 $
 */
public class CheckerOutputGUI {


    /**
     * The table that displays the data.
     */
    private JTable table;


    /**
     * The panel that holds the table and the label.
     */
    private JPanel panel;

    /**
     * The label displys the number of errors and warnings.
     */
    private JLabel label;

    /**
     * This method is used to create and setup the table
     * which is displayed in the frame.
     * @param c The StateChartCheckerBase.
     */
    private void createTable(final StateChartCheckerBase c) {
        table = new JTable(c);
        /* Set the selection mode to full row selection*/
        table.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        /* Set the cell renderer. */
        table.getColumnModel().getColumn(0).setCellRenderer(
                new MyCellRenderer(c));
    }

    /**
     * The constructor of the frame.
     * It sets the title of the frame, creates and adds the table to itself.
     * @param c The StateChartCheckerBase.
     */
    public CheckerOutputGUI(final StateChartCheckerBase c) {

        createTable(c);
        label = new JLabel("");
        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.add(label, BorderLayout.NORTH);
        panel.add(new JScrollPane(table), BorderLayout.CENTER);
    }

    /**
     * Get the table.
     * @return The instance of the table.
     */
    public final JTable getTable() {
        return table;
    }

    /**
     * @return Return the Scrollpane
     */
    public final JComponent getComponent() {
        return panel;
    }

    /**
     *
     * @param t Text to be displayed in the label above the table.
     */
    public final void setLabelText(final String t) {
        label.setText(t);
    }
}
```

G.3.4. CheckingProblem.java

```
//$Id: CheckingProblem.java,v 1.4 2006/07/21 17:51:56 kbe Exp $
package kiel.checking;

import java.util.ArrayList;

import kiel.dataStructure.GraphicalObject;


/**
 *
 * <p>Description: This class holds the informations about problems that
 * occured when parsing the datastructure. Instances will be stored in an
 * instance of {@link kiel.checking.StateChartCheckerBase}.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.4 $ last modified $Date: 2006/07/21 17:51:56 $
 *
 */
public class CheckingProblem {

    /**
     * Holds a short description of the problem.
     */
    private String problem;


    /*
     * Holds the object which is the source of the problem.
     */
    //private GraphicalObject object;

    /**
     * Holds the object that are the source of the problem.
     */
    private ArrayList objects;

    /**
     * The constructor of the robustness problem.
     * @param o The source of the problem.
     * @param p A problem description.
     */
    public CheckingProblem(final Object o, final String p) {
        this.objects = new ArrayList();
        this.problem = p;
        this.objects.add(o);
    }


    /**
     *
     * @param o An ArrayList containing the source of the problem.
     * @param p A description of the problem.
     */
    public CheckingProblem(final ArrayList o, final String p) {
        this.problem = p;
        this.objects = new ArrayList(o);
    }


    /**
     *
     * @return Returns the source of the problem.
     */
    public final GraphicalObject getFirstObject() {
        return (GraphicalObject) objects.get(0);
    }

    /**
     * Set the source of a problem manually.
     * @param o The source of the problem.
     */
    public final void setFirstObject(final Object o) {
        this.objects.set(0, o);
    }


    /**
     *
     * @return A short description of the problem.
     */
    public final String getProblem() {
        return problem;
    }


    /**
     *
     * @param p Set the description of the problem manually.
     */
    public final void setProblem(final String p) {
        this.problem = p;
    }


    /**
     *
     * @return The sources of the problem in an ArrayList.
     */
    public final ArrayList getObjects() {
        return objects;
    }


    /**
     *
     * @param o An ArrayList containing graphical objects.
     */
    public final void setObjects(final ArrayList o) {
        this.objects = o;
    }

}
```

G.3.5. CheckingProperties.java

```
//$Id: CheckingProperties.java,v 1.12 2006/11/08 10:13:03 kbe Exp $
package kiel.checking;

import java.awt.Color;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Set;

import kiel.util.preferences.Preferences;
import kiel.util.LogFile;

/**
 * <p>Description: Used to load and store user defineable properties.</p>
 *
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.12 $ last modified $Date: 2006/11/08 10:13:03 $
 */
public final class CheckingProperties {

    /**
     * These are the internal properties.
     */
    private static Preferences prefs;

    /**
     * This the key for the resouce file containing the default values.
     */
    private static final String DEFAULTRESOURCE = "checking.properties";

    /**
     * The robustness log.
     */
    private static LogFile checkingLog = new LogFile("Checking", 0);

    static {
        prefs = Preferences.createInstance("checking",
                CheckingProperties.class.getResource(DEFAULTRESOURCE),
                checkingLog);
        checkingLog.setLogLevel(getRobustnessLogLevel());
    }

    /**
     * @return The robustness log.
     */
    public static LogFile getRobustnessLog() {
        return checkingLog;
    }

    /**
     * The semantic checker log.
     */
    private static LogFile corrLog = new LogFile("Correctness",
            getCorrLogLevel());

    /**
     * @return The semantic checker log.
     */
    public static LogFile getCorrLog() {
        return corrLog;
    }

    /**
     * The semantic checker log.
     */
    private static LogFile semLog = new LogFile("Semantic Robustness",
            getSemLogLevel());

    /**
     * @return The semantic checker log.
     */
    public static LogFile getSemLog() {
        return semLog;
    }

    /**
     * The syntactic checker log.
     */
    private static LogFile synLog = new LogFile("Syntactic Robustness",
            getSynLogLevel());

    /**
     * @return The syntactic checker log.
     */
    public static LogFile getSynLog() {
        return synLog;
    }

    /**
     * Static class, do not instantiate.
     */
    private CheckingProperties() {

    }

    /**
     * This is the key for property key delimiter.
     */
    private static final String DELIMITER = ".";

    /**
     * This is the key for the checking properties.
     */
    private static final String CHECKING = "checking";

    /**
     * This is the key for the property holding the color for errors.
     */
```

```
    private static final String ERRORCOLOR = "errorColor";

    /**
     * This is the key for the property holding the color for warnings.
     */
    private static final String WARNINGSCOLOR = "warningColor";

    /**
     * This is the key for the property denoting if a single rule is enabled.
     */
    private static final String RULESENABLED = "ruleEnabled";

    /**
     * The name of the key for the cvclMode.
     */
    private static final String CVCLMODE = "cvclMode";

    /**
     * The name of the key for the log level.
     */
    private static final String CHECKINGLOGLEVEL = "CheckingLogLevel";

    /**
     * The name of the key for the log level of correctness checks.
     */
    private static final String CORRECTNESSLOGLEVEL = "CorrLogLevel";

    /**
     * The name of the key for the log level of semantic checks.
     */
    private static final String SEMANTICLOGLEVEL = "SemLogLevel";

    /**
     * The name of the key for the log level of syntactic checks.
     */
    private static final String SYNTACTICLOGLEVEL = "SynLogLevel";

    /**
     * The key denoting if to show the ID of the Object in the message.
     */
    private static final String SHOWNODEID = "showNodeID";

    /**
     * The name of the key for the property if the name of the object is
     * returned in the message.
     */
    private static final String SHOWNODENAME = "showNodeName";

    /**
     * The name of the key.
     */
    private static final String SHOWRULENAME = "showRuleName";

    /**
     * Property name.
     */
    private static final String SHOWCLASSIFICATION = "showClassification";

    /**
     * The property if the counters are shown.
     */
    private static final String SHOWCOUNTERS = "showCounters";

    /**
     *
     */
    private static final String SHOWTRANSITIONS = "showTransitions";

    /**
     *
     */
    private static final String SHOWPRIORITY = "showPriority";

    /**
     * The property of the benchmarking part.
     */
    private static final String TIMING = "timing";

    /**
     * The prefix for the benchmark file.
     */
    private static final String TIMINGPRESTRING = "timingPrestring";

    /**
     * The postfix for the benchmark file.
     */
    private static final String TIMINGPOSTSTRING = "timingPoststring";

    /**
     * Property for the transition overlap check.
     */
    private static final String OVERLAPHIERARCHY = "transitionOverlapHierarchy";

    /**
     * The key for the profiles.
     */
    private static final String MODELRULESET = "modelRuleSet";

    /**
     * Reloads the properties file.
     */
    protected static void reload() {
        prefs.reload();
    }

    /**
     *
     * @return The color of errors.
     */
    public static Color getErrorColor() {
        return prefs.getRGBColor(CHECKING + DELIMITER + ERRORCOLOR);
    }

    /**
     *
     * @return The color of warnings.
     */
    public static Color getWarningColor() {
        return prefs.getRGBColor(CHECKING + DELIMITER + WARNINGSCOLOR);
    }
```

```
    /**
     * Reads the rule status from the properties.
     * @param ruleName The name of the rule.
     * @return The status of the rule.
     */
/* public static boolean getRuleStatus(final String ruleName) {
       return prefs.getBoolean(CHECKING + DELIMITER + RULESENABLED
           + DELIMITER + ruleName, true);
}*/

    /**
     * Sets the rule status in the properties.
     * @param ruleName The name of the rule.
     * @param value The value.
     */
    public static void setRuleStatus(final String ruleName,
            final boolean value) {
        prefs.setBoolean(ruleName, value);
    }

    /**
     * @return .
     */
    public static int getRobustnessLogLevel() {
        return prefs.getInt(CHECKING + DELIMITER + CHECKINGLOGLEVEL);
    }

    /**
     * @return .
     */
    public static int getCorrLogLevel() {
        return prefs.getInt(CHECKING + DELIMITER + CORRECTNESSLOGLEVEL);
    }

    /**
     * @return .
     */
    public static int getSemLogLevel() {
        return prefs.getInt(CHECKING + DELIMITER + SEMANTICLOGLEVEL);
    }

    /**
     * @return .
     */
    public static int getSynLogLevel() {
        return prefs.getInt(CHECKING + DELIMITER + SYNTACTICLOGLEVEL);
    }

    /**
     * @return Whether to use CVCL lib or bin.
     */
    public static boolean isCvclModeBin() {
        return prefs.getString(CHECKING + DELIMITER + CVCLMODE)
                .equalsIgnoreCase("bin");
    }

    /**
     * @return .
     */
    public static boolean showNodeID() {
        return prefs.getString(CHECKING + DELIMITER + SHOWNODEID)
                .equals("true");
    }


    /**
     *
     */
    public static boolean showRuleName() {
        return prefs.getString(CHECKING + DELIMITER + SHOWRULENAME)
            .equals("true");
    }

    /**
     * @return .
     */
    public static boolean showNodeName() {
        return prefs.getString(CHECKING + DELIMITER + SHOWNODENAME)
                .equals("true");
    }

    /**
     * @return .
     */
    public static boolean showClassification() {
        return prefs.getString(CHECKING + DELIMITER + SHOWCLASSIFICATION)
                .equals("true");
    }

    /**
     * @return .
     */
    public static boolean showCounters() {
        return prefs.getString(CHECKING + DELIMITER + SHOWCOUNTERS)
                .equals("true");
    }

    /**
     * @return .
     */
    public static boolean showTransitions() {
        return prefs.getString(CHECKING + DELIMITER + SHOWTRANSITIONS)
                .equals("true");
    }

    /**
     * @return .
     */
    public static boolean showPriority() {
        return prefs.getString(CHECKING + DELIMITER + SHOWPRIORITY)
                .equals("true");
    }

    /**
     * @return .
     */
    public static boolean timing() {
        return prefs.getString(CHECKING + DELIMITER + TIMING).equals("true");
    }
```

```
    /**
     * @return .
     */
    public static String timingPrestring() {
        return prefs.getString(CHECKING + DELIMITER + TIMINGPRESTRING);
    }

    /**
     * @return .
     */
    public static String timingPoststring() {
        return prefs.getString(CHECKING + DELIMITER + TIMINGPOSTSTRING);
    }

    /**
     * @return .
     */
    public static boolean overlapHierarchy() {
        return prefs.getString(CHECKING + DELIMITER + OVERLAPHIERARCHY)
            .equals("true");
    }


    /**
     * Get the profile of rules for a model according to a rule set.
     * @param modelString A string containing the model source and version.
     * @return A string containing all rules, that are automatically enabled
     * when loading a statechart.
     */
    public static String modelRuleSet(final String modelString) {
// return prefs.getString(CHECKING + DELIMITER + MODELRULESET
// + DELIMITER + ruleSet + DELIMITER + modelString);
        return prefs.getString(CHECKING + DELIMITER + MODELRULESET
                + DELIMITER + modelString);
    }

    /**
     * Use this method to get the list of available profiles.
     * @return List containing the names of available profiles.
     */
    public static ArrayList getAvailableProfiles() {
        ArrayList result = new ArrayList();
        Set keys = prefs.getUserKeySet();
        String key = "";
        String id = CHECKING + DELIMITER + MODELRULESET + DELIMITER;
        Iterator iter = keys.iterator();
        while (iter.hasNext()) {
            key = (String) iter.next();
            if (key.indexOf(id) > -1) {
                result.add(key.substring(id.length()));
            }
        }


        return result;
    }

    /**
     * This method returns the value for the profile passed in the param.
     * @param profileName The name of the profile to return
     * @return A comma separated string containing names of rules.
     */
    public static String getProfile(final String profileName) {
        return prefs.getString(CHECKING + DELIMITER + MODELRULESET
                + DELIMITER + profileName);
    }

}
```

G.3.6. MyCellRenderer.java

```
//$Id: MyCellRenderer.java,v 1.4 2006/05/31 15:00:20 kbe Exp $
package kiel.checking;

import java.awt.Color;
import java.awt.Component;

import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.table.TableCellRenderer;

/**
 * Description: The cell renderer for the table of errors and warnings.
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.4 $ last modified $Date: 2006/05/31 15:00:20 $
 */
public class MyCellRenderer extends JTextArea implements TableCellRenderer {

    /**
     * The automatically generated serial Version UID.
     */
    private static final long serialVersionUID = 8099471781811987891L;

    /**
     * The StateChartCheckerBase.
     */
    private StateChartCheckerBase checker;

    /**
     * Constructor of a CellRenderer.
     * @param c The StateChartCheckerBase.
     */
    public MyCellRenderer(final StateChartCheckerBase c) {
        this.checker = c;
        setLineWrap(true);
        setWrapStyleWord(true);
    }

    /**
     * @param aTable The table of which the given cell has to be rendered.
     * @param value The value that has to be displayed.
     * @param isSelected Denoting if the cell is selected.
     * @param hasFocus Denoting if the cell has the focus.
     * @param row The rowindex of the cell.
     * @param column The columnindex of the cell.
     * @return A instance of the renderer.
     */
    public final Component getTableCellRendererComponent(final JTable aTable,
            final Object value, final boolean isSelected,
            final boolean hasFocus, final int row, final int column) {
        this.setText(value.toString());
        setSize(aTable.getColumnModel().getColumn(column).getWidth(),
                getPreferredSize().height);
        if (row < checker.getErrors().size()) {
            this.setForeground(CheckingProperties.getErrorColor());
        } else {
            this.setForeground(CheckingProperties.getWarningColor());
        }
        if (aTable.getRowHeight(row) != getPreferredSize().height) {
            aTable.setRowHeight(row, getPreferredSize().height);
        }
        if (isSelected) {
            this.setBackground(aTable.getSelectionBackground());
            this.setForeground(Color.BLACK);
        } else {
            this.setBackground(Color.WHITE);
        }
        return this;
    }
}
```

G.3.7. IStateChartVisitor.java

```
package kiel.util;

import java.util.Collection;

import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.StateChart;

/**
 *
 * <p>Title: Kiel Checker Interface.</p>
 *
 * <p>Description: An interface for checkers working on
 * kiel.datastructure.StateChart.</p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.4 $ last modified $Date: 2006/04/03 09:29:59 $
 */
public interface IStateChartVisitor {


    /**
     * This method does the datastructure checking.
     * @param chart A Statechart
     * @return boolean true if there where no problems in the parse tree.
     */
    boolean checkDataStructure(StateChart chart);


    /**
     * This method is called when an object has to be dispatched.
     * <br>See StateChartDepthFirstAdapter for a sample.
     * @param o The currently handled object of the Statechart.
     * @see kiel.util.StateChartDepthFirstAdapter
     */
    void dispatch(final GraphicalObject o);

    /**
     * Ask this method for warnings found in the StateChart datastructure.
     * @return Collection of StateChartProblems describing the warnings.
     */
    Collection getWarnings();

    /**
     * Ask this method for errors found in the StateChart datastructure.
     * @return Collection of StateChartProblems describing the errors.
      */
    Collection getErrors();
}
```

G.3.8. StateChartDepthFirstAdapter.java

```
//$Id: StateChartDepthFirstAdapter.java,v 1.6 2006/07/21 17:52:47 kbe Exp $
package kiel.util;

import java.util.ArrayList;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Node;
import kiel.dataStructure.Region;

/**
 * <p>Description: State chart visitor using the depth first search to
 * traverse the datastructure</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>.
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @author <a href="mailto:gsc@informatik.uni-kiel.de">Gunnar Schaefer</a>
 * @version $Revision: 1.6 $ last modified $Date: 2006/07/21 17:52:47 $
 */
public class StateChartDepthFirstAdapter {

    /**
     *
     * @param intf The visitor whose dispatch method is called when a node is
     * found
     * @param root The start node of the search
     * @param doVisitRegions Denoting wheter to dispatch regions to the visitor
     * or not
     * @return Can be modified for later usage.
     */
    public final boolean traverse(final IStateChartVisitor intf,
            final Node root, final boolean doVisitRegions) {

        ArrayList stack = new ArrayList();
        Node n;

        stack.add(root);

        while (stack.size() > 0) {

            n = (Node) stack.remove(0);

            /**
             * The node is dispatched if either the parameter doVisitRegions is
             * true or the node is not a region.
             */
            if (doVisitRegions || (!(n instanceof Region))) {
                intf.dispatch(n);
            }

            if (n instanceof CompositeState) {
                stack.addAll(((CompositeState) n).getSubnodes());
            }
        }
        return true;
    }
}
```

G.4. XMI-Fileinterface

G.4.1. XMI.java

```
package kiel.fileInterface.xmi;

import java.io.File;

import javax.swing.filechooser.FileFilter;

import kiel.dataStructure.StateChart;
import kiel.fileInterface.FileInterface;
import kiel.fileInterface.FileInterfaceException;
import kiel.graphicalInformations.View;

/**
 * <p>Description: The main module of the XMI fileinterface plugin.. </p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/10/25 15:27:10 $
 */
public class XMI extends FileInterface {

    /**
     * An instance of the class reader module.
     * Contains information of available XMI-Readers.
     */
    private ClassReader cr = null;


    /**
     * Method from the interface. Used to read a statechart from a XMI file.
     * @param arg0 The file to be read.
     * @return A statechart instance.
     * @throws FileInterfaceException .
     */
    public final StateChart readStateChartDocument(final File arg0)
        throws FileInterfaceException {

        if (cr == null) {
            cr = new ClassReader();
        }

        XMIInfoReader xmiinfo = new XMIInfoReader();

        xmiinfo.assignFile(arg0.toString());

        if (xmiinfo.isAssigned()) {
            AbstractXMIReader reader = cr.createReaderModule(
                    xmiinfo.getExporterName(),
                    xmiinfo.getExporterVersion(),
                    xmiinfo.getXMIVersion(),
                    xmiinfo.getUMLVersion(),
                    xmiinfo.getDoc(),
                    xmiinfo.getFilename());

            if (reader != null) {
                try {
                    return reader.createStateChartFromFile();
                } catch (Exception e) {
                    kiel.fileInterface.FileInterfaceModel.
                    getFileInterfaceLog().log(10, e.getMessage());
                    return null;
                }
            } else {
                FileInterfaceException ex = new FileInterfaceException(
                        "No reader for the passed XMI-file found.");
                throw ex;
            }

        } else {
            FileInterfaceException ex = new FileInterfaceException(
                    "The selected xmi-file is invalid.");
            throw ex;
        }

    }

    /**
     * Unused interface method.
     * @return <code>null</code>.
     * @throws FileInterfaceException .
     */
    public final View getReadView() throws FileInterfaceException {
        return null;
    }

    /**
     * Interface method to export a statechart.
     * @param arg0 The statechart to export.
     * @param arg1 unused.
     * @param arg2 The file to write the export to.
     * @return A file containing the exported statechart.
     * .
     */
    public final File writeStateChartDocument(final StateChart arg0,
            final View arg1, final File arg2) {
        XMIGenerator gen = new XMIGenerator();
        return gen.generateChart(arg0, arg2);
    }

    /**
     * Returns the global filefilter instance.
```

```
     * @return a FileFilter instance.
     */
    public final FileFilter getStateChartFileFilter() {
        return XMIFileFilter.getInstance();
    }

    /**
     * As of 2006-07-28 this Plug-In can also write XMI files.
     * @return true.
     */
    public final boolean canWrite() {
        return true;
    }

    /**
     * As of 2006-07-10 this Plug-In is able to read XMI files.
     * @return True
     */
    public final boolean canRead() {
        return true;
    }

    /**
     * The name of this plugin.
     * @return The name of this Fileplugin.
     */
    public final String getName() {
        return "XMI FilePlugin";
    }
}
```

G.4.2. XMIFileFilter.java

```
package kiel.fileInterface.xmi;

import java.io.File;
import javax.swing.filechooser.FileFilter;

/**
 * <p>Description: The file filter for XMI files. Used by Fileinterface.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.2 $ last modified $Date: 2006/10/25 15:27:11 $
 */
public class XMIFileFilter extends FileFilter {

    /** an intance counter.*/
    private static XMIFileFilter instance = null;


    /**
     * Returns the global xmi file filter.
     * @return A global file filter instance.
     */
    public static XMIFileFilter getInstance() {
        if (instance == null) {
            instance = new XMIFileFilter();
        }
        return instance;
    }


    /**
     * Tests if a passed file is accepted by this file filter.
     * @param pathname A name of a file.
     * @return <code>true</code> if the file is accepted.
     */
    public final boolean accept(final File pathname) {
        if (pathname.isDirectory()) {
            return true;
        }
        return (
            pathname.canRead()
                && pathname.isFile()
                && pathname.getPath().toLowerCase().endsWith(".xmi"));

    }

    /**
     * @return "XMI Files .xmi"
     */
    public final String getDescription() {
        return "XMI Files .xmi";
    }


}
```

G.4.3. ArgoUMLReader.java

```
package kiel.fileInterface.xmi.ArgoUML;

import java.util.Collection;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;

import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//import kiel.dataStructure.*;
import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.ConditionalTransition;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Region;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.StringLabel;
import kiel.dataStructure.Transition;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.Signal;
import kiel.dataStructure.intexp.IntegerVariable;
import kiel.fileInterface.FileInterfaceException;
import kiel.fileInterface.xmi.AbstractXMIReader;
import kiel.fileInterface.xmi.XMIReaderDescriptor;
import kiel.fileInterface.xmi.common.KielUMLAttribute;
import kiel.fileInterface.xmi.common.KielUMLDataType;

import kiel.fileInterface.JavaTransitionLabels.lexer.Lexer;
import kiel.fileInterface.JavaTransitionLabels.node.Start;
import kiel.fileInterface.JavaTransitionLabels.parser.Parser;
import kiel.fileInterface.JavaTransitionLabels.parser.ParserException;

import java.io.PushbackReader;
import java.io.StringReader;

/**
 * <p>DescriptionThe ArgoUMLReader reads XMI-files generated by
 * ArgoUML Version 0.20.x.
 * Use the method <code>createStateChartFromFile</code> to
 * parse the given XMI-File.
 * The ArgoUMLReader utilizes the LabelGenerator to parse the Transition
 * labels. It is adopted from a Rhapsody XMI reader.
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.8 $ last modified $Date: 2006/11/08 10:26:20 $
 *
 */
public class ArgoUMLReader extends AbstractXMIReader {

    /**
     * The max size of the XMI reader buffer.
     */
    private final int maxbuffersize = 1024;

    /**
     * The supported exporter.
     */
    private final String supportedExporter =
        "ArgoUML (using Netbeans XMI Writer version 1.0)";

    /**
     * The version of the exporter supported by this module.
     */
    private final String supportedExporterVersion = "0.20.x";

    /**
     * The version of xmi supported by this module.
     */
    private final String supportedXMIVersion = "1.2";

    /**
     * The version of UML supported by this module.
     */
    private final String supportedUMLVersion = "1.4";

    /**
     * The date of the revision.
     */
    private final String sRevisionDate = "0.2 - August 2006";

    /**
     * The author of this module.
     */
    private final String sModulAuthor = "Ken Bell";

    /**
     * A short description of this module.
     */
    private final String sModulDescription =
        "Module for importing xmi files generated by ArgoUML";

    /**
     * The name of this module.
     */
    private final String sModuleName = "ArgoUML . Importer";

    /**
     * Denotes, if there can be more than one statechart.
     */
    private final boolean bMultipleSCSupport = true;

    /**
```

```
     * The name of the XMI Tag for pseudo states.
     */
    private final String tagPseudoState = "UML:Pseudostate";

    /**
     * The name of the XMI tag for states.
     */
    private final String tagState = "UML:State";

    /**
     * The name of the XMI tag for simple states.
     */
    private final String tagSimpleState = "UML:SimpleState";

    /**
     * The name of the XMI tag for composite states.
     */
    private final String tagCompositeState = "UML:CompositeState";

    /**
     * The name of the XMI tag for final states.
     */
    private final String tagFinalState = "UML:FinalState";

    /**
     * The name of the XMI tag for synch states.
     */
    private final String tagSynchState = "UML:SynchState";

    /**
     * The name of the XMI tag for the exporter.
     */
    private final String tagExporterName = "XMI.exporter";

    /**
     * The name of the XMI tag for the exporter version.
     */
    private final String tagExporterVersion = "XMI.exporterVersion";

    /**
     * The name of the XMI tag for the id reference.
     */
    private final String attrIDRef = "xmi.idref";

    /**
     * The name of the attribute of the item id.
     */
    private final String attrID = "xmi.id";

    /**
     * The name of the XMI tag containing the value of some fields.
     */
    private final String attrXMIValue = "xmi.value";

    /**
     * The name of the XMI tag containing the version of XMI.
     */
    private final String attrXMIVersion = "xmi.version";

    /**
     * All transitions are added to this vector.
     */
    private Vector vGlobalTransitions = null;

    /**
     * All attributes are added to this vector while parsing the document.
     */
    private Vector vGlobalAttributes = null;

    /**
     * All signals are added to this vector while parsing the document.
     */
    private Vector vGlobalPackageSignals = null;

    /**
     * All signal events are added to this vector while parsing the document.
     */
    private Vector vGlobalSignalEvents = null;

    /**
     * All signal event receptions are added to this vector
     * while parsing the document.
     */
    private Vector vGlobalEventReception = null; // RPSClassEventReception

    /**
     * All datatypes are added to this vector while parsing the document.
     */
    private Vector vGlobalDatatypes = null;

    /**
     * All Statecharts are added to this vector while parsing the document.
     */
    private Vector vStateCharts = null;

    /**
     * The hashtable for the internally created ids.
     * This dictionary is used as a lookup reference.
     */
    private Hashtable idDictionary = null;

    /**
     * A counter for creating ids.
     */
    private int idCounter;


    /**
     * Creator for the XMI Reader module.
     *
     */
    public ArgoUMLReader() {
        super();
        vGlobalTransitions = new Vector();
        vGlobalAttributes = new Vector();
        vGlobalPackageSignals = new Vector();
        vGlobalSignalEvents = new Vector();
        vGlobalEventReception = new Vector();
        vGlobalDatatypes = new Vector();
        vStateCharts = new Vector();
```

```
    idDictionary = new Hashtable();
    idCounter = 0;
}


/**
 * Internally used method to clear all temporary data.
 *
 */
private void clearAllXMIData() {
    vGlobalTransitions.clear();
    vGlobalAttributes.clear();
    vGlobalPackageSignals.clear();
    vGlobalSignalEvents.clear();
    vGlobalEventReception.clear();
    vGlobalDatatypes.clear();
    vStateCharts.clear();
    idDictionary.clear();
    idCounter = 0;
}

/**
 * Returns the list of all found statecharts.
 * @return The list of all statecharts found in the file.
 */
public final Vector getStatechartList() {

    if (vStateCharts.size() < 1) {
        FileInterfaceException ex = new FileInterfaceException(
                "The selected XMI-File does not contain any statecharts.");
        ex.showMessageBox();
        return null;
        //throw ex;
    } else {
        return vStateCharts;
    }
}


/**
 * This method returns the first statechart.
 *
 * @return The first statechart found in the file to be read.
 */
public final StateChart createStateChartFromFile() {
    StateChart sc = null;

    if (vStateCharts.size() > 0) {
        sc = (StateChart) vStateCharts.get(0);
    } else {
        FileInterfaceException ex = new FileInterfaceException(
                "The selected XMI-File does not contain any statecharts.");
        ex.showMessageBox();
        //throw ex;
    }

    /*Vector v = getSCClassList();
    if (v.size() == 1) {
      XMIClassInfo classInfo = (XMIClassInfo)v.elementAt(0);
      sc = createStateChartFromFile(classInfo.classID);
    }*/
    return sc;
}

/**
 * Creates a and returns the statechart from the class with the passed id.
 * @param classID The id of the class to get the statechart.
 * @return The statechart associated with the passed class.
 */
public final StateChart createStateChartFromFile(final String classID) {
    StateChart sc = null;
    /* UMLClass c = getClass(classID);
    if (c != null) {
      // Statechart erstellen
      sc = new StateChart("Statechart of "+c.getName(), c.getName());
      sc.setGeneratorName("Importmodul fÃ¤r I-Logix"
            + "Rhapsody v4.0, Steffen Pietsch");
      RPSPackage p = getPackage(c.getPackageID());
      if (p != null)
        sc.setPackageName(p.getName());
      // Zustï¿½nde hinzufï¿½gen
      Vector vRPSStates = getStatesByClassID(classID);
      sc.addStates(convertRPSToNormalStates(vRPSStates));
      sc.findRoot();
      // Trigger hinzufï¿½gen
      Vector vRPSPackageSignals = getPackageSignalsByClassID(classID);
      sc.addEvents(convertRPSPackageSignalsToNormalTrigger(
          vRPSPackageSignals));
      // Transitionen hinzufï¿½gen
      Vector vRPSTransitions = getTransitionsByClassID(classID);
      sc.addTransitions(convertRPSTransitionsToNormalTransitions(
          vRPSTransitions));
      // Attribute
      Vector vRPSAttributes = getAttributesByClassID(classID);
      sc.addClassAttributes(convertRPSAttributesToNormalAttributes(
          vRPSAttributes));

    } */
    return sc;
}

/**
 * This file filter can read multiple statecharts from a file.
 * @return True.
 */
public final boolean supportsMultipleSCInFile() {
    return true;
}


/**
 * Classes are not handled in KIEL. Therfore the number of classes
 * equals zero.
 * @return The number of classes found.
 */
private int getClassCount() {
    return 0;
}

/**
```

```
     * Get the ID of the class associated with the passed index.
     * This function will be useful, as a dialogue to choose a certain
     * statechart from the set of defined ones will be read.
     * @param index The number of the class to be retrieved.
     */
    private void getClass(final int index) {
        //return (RPSClass)vGlobalClasses.elementAt(index);
    }

    /**
     * Get a class by id.
     * @param iD A id string.
     */
    private void getClass(final String iD) {
        /*RPSClass uc = null;
        for (int i = 0; i < getClassCount(); i++) {
            if (getClass(i).getID().equals(ID))
                return getClass(i);
        }
        return uc;*/
    }

    /**
     * Returns the number of found statecharts.
     * @return The number of read Statecharts read from the file.
     */
    public final int getStateChartCount() {
        int iSCCounter = 0;
        for (int i = 0; i < getClassCount(); i++) {
            return 0;
            /* if (getClass(i).hasStateChart())
            iSCCounter++;*/
        }
        return iSCCounter;
    }

    /**
     * Classes are not handled in KIEL therefore <code>null</code> is returned.
     * @return The list of all classes.
     */
    public final Vector getSCClassList() {
        return null;
    }

    /**
     * Implementation of an abstract method.
     * @return The name of this module.
     */
    public final String getModuleName() {
        return sModuleName;
    }

    /**
     * Implementation of an abstract method.
     * @return The date of the current revision.
     */
    public final String getRevisionDate() {
        return sRevisionDate;
    }

    /**
     * Implementation of an abstract method.
     * @return A string describing this module.
     */
    public final String getModuleDescription() {
        return sModulDescription;
    }

    /**
     * Implementation of an abstract method.
     * @return The name of the author of this import module.
     */
    public final String getModuleAuthor() {
        return sModulAuthor;
    }

    /**
     * Implementation of an abstract method.
     * @return The supported UML version.
     */
    public final String getUMLVersion() {
         String sVersion = "?";
         NodeList nl = getDocument().getElementsByTagName("XMI.metamodel");
         if (nl.getLength() == 1) {
             sVersion = nl.item(0).getAttributes().getNamedItem(
                 "xmi.version").getNodeValue();
         }
         return sVersion;
    }


    /**
     * Implementation of an abstract method.
     * @return The supported version of XMI.
     */
    public final String getXMIVersion() {
         String sVersion = "?";
         NodeList nl = getDocument().getElementsByTagName("XMI");
         if (nl.getLength() == 1) {
             sVersion = nl.item(0).getAttributes().getNamedItem(
                     attrXMIVersion).getNodeValue();
         }
         return sVersion;
    }

    /**
     * Currently not implemented.
     * @return The name of the exporter found.
     */
    public final String getExporterName() {
        // TODO Auto-generated method stub
        return null;
    }

    /**
     * Currently not implemented.
     * @return The version of the exporter.
     */
    public final String getExporterVersion() {
        // TODO Auto-generated method stub
```

```
        return null;
    }

    /**
     * @return A string that has to match the exporter specified in the
     * XMI file.
     */
    public final String getSupportedExporterName() {
        return supportedExporter;
    }

    /**
     * @return A version string that has to match the exporter version
     * as it is specified in the XMI file.
     */
    public final String getSupportedExporterVersion() {
        return supportedExporterVersion;
    }

    /**
     * @return The version of the supported XMI version.
     */
    public final String getSupportedXMIVersion() {
        return supportedXMIVersion;
    }

    /**
     * @return The supported version of UML.
     */
    public final String getSupportedUMLVersion() {
        return supportedUMLVersion;
    }

    /**
     * @return An instance of the XMIReaderDescriptor containing all needed
     * informations.
     */
    public final XMIReaderDescriptor getDescriptor() {
        XMIReaderDescriptor xrd = new XMIReaderDescriptor(null);
        xrd.moduleName = sModuleName;
        xrd.revisionDate = getRevisionDate();
        xrd.moduleAuthor = getModuleAuthor();
        xrd.moduleDescription = getModuleDescription();
        xrd.exporterName = getSupportedExporterName();
        xrd.exporterVersion = getSupportedExporterVersion();
        xrd.xmiVersion = getSupportedXMIVersion();
        xrd.umlVersion = getSupportedUMLVersion();
        return xrd;
    }


    /**
     * Returns the first node of type ELEMENT_NODE from the node list
     * that matches the passed name.
     * @param nl Source list.
     * @param elementName The name of the node to be returned.
     * @return A node that matches the given name.
     */
    private Node getFirstNodeFromList(final NodeList nl,
            final String elementName) {
        if (nl.getLength() > 0) {
            for (int i = 0; i < nl.getLength(); i++) {
                if (nl.item(i).getNodeType() == Node.ELEMENT_NODE) {
                    if (nl.item(i).getNodeName().equals(elementName)) {
                        return nl.item(i);
                    }
                }
            }
        }
        return null;
    }

    /**
     * Returns the first node of type ELEMENT_NODE from the list.
     * @param nl A list of nodes.
     * @return The first node from the list with node type ELEMENT_NODE.
     */
    private Node getFirstNodeFromListNoText(final NodeList nl) {
        if (nl.getLength() > 0) {
            for (int i = 0; i < nl.getLength(); i++) {
                if (nl.item(i).getNodeType() == Node.ELEMENT_NODE) {
                    return nl.item(i);
                }
            }
        }
        return null;
    }

    /**
     * Implementation of the abstract method stub.
     * Reads all statecharts from the currently handled XMI file.
     */
    public final void readAll() {
        // alle Vektoren leeren
        clearAllXMIData();
        // Einlesen beginnen
        NodeList nl = getDocument().getElementsByTagName("UML:Model");
        if (nl.getLength() == 1) {
            Node ndMainModel = nl.item(0);
            if (ndMainModel.hasChildNodes()) {
                NodeList nlFoundationNodes = ndMainModel.getChildNodes();
                Node ndOwnedElement = getFirstNodeFromList(
                        nlFoundationNodes, "UML:Namespace.ownedElement");
                if (ndOwnedElement != null) {
                    if (ndOwnedElement.hasChildNodes()) {
                        Vector vPackages = getNodesFromList(
                                ndOwnedElement.getChildNodes(), "UML:Package");
                        /* for (int i = 0; i < vPackages.size(); i++) {
                            try {
                                readPackage((Node)vPackages.elementAt(i),"");
                            } catch (FileInterfaceException e) {
                                e.printStackTrace();
                            }
                        }*/
                      // now read all model informations from the default package
                            readPackage(ndMainModel, "");


                        /*Vector vClasses = getNodesFromList(
                                ndOwnedElement.getChildNodes(), "UML:Class");
```

```
                    for (int i = 0; i < vClasses.size(); i++) {
                        try {
                            readClassFromPackage(
                            (Node)vClasses.elementAt(i),"","");
                        } catch (FileInterfaceException e) {
                            e.printStackTrace();
                        }
                    }*/
                }
            }
        }
    } else {
        getPrintStream().println("Es ist ein fataler Fehler aufgetreten: "
                + "Das Element \"Model_Management.Model\" "
                + "konnte nicht gefunden werden.");
        getPrintStream().println("Der Vorgang wird abgebrochen.");
    }

}


/**
 * This method returns all nodes of the passed element from a list
 * of nodes.
 * @param nl The source list.
 * @param elementName The name of the nodes to be returned.
 * @return A list containing all nodes with the given name.
 */
private Vector getNodesFromList(final NodeList nl,
        final String elementName) {
    Vector vResult = new Vector();
    if (nl.getLength() > 0) {
        for (int i = 0; i < nl.getLength(); i++) {
            if (nl.item(i).getNodeType() == Node.ELEMENT_NODE) {
                if (nl.item(i).getNodeName().equals(elementName)) {
                    vResult.add(nl.item(i));
                }
            }
        }
    }
    return vResult;
}


/**
 * Processes the package.
 * @param ndModelPackage The node from the xmi tree.
 * @param parentPackageID The id of the owning package.
 */
public final void readPackage(final Node ndModelPackage,
        final String parentPackageID) {
    String sPackageName = "";
    String sPackageID = "";
    Vector vDataTypes = null;
    Vector vClasses = null;
    Vector vSignals = null;
    Vector vInnerPackages = null;
    if (ndModelPackage.hasChildNodes()) {
        // Packagedaten auslesen
        NodeList nlModelChildNodes = ndModelPackage.getChildNodes();
        Node ndOwnedElement = getFirstNodeFromList(nlModelChildNodes,
        "UML:Namespace.ownedElement");
        if (ndOwnedElement != null) {
            NodeList nlOwned = ndOwnedElement.getChildNodes();
            vDataTypes = getNodesFromList(nlOwned, "UML:DataType");
            vClasses = getNodesFromList(nlOwned, "UML:Class");
            vSignals = getNodesFromList(nlOwned, "UML:SignalEvent");
            vInnerPackages = getNodesFromList(nlOwned, "UML:Package");
        } else {
            vDataTypes = new Vector();
            vClasses = new Vector();
            vSignals = new Vector();
            vInnerPackages = new Vector();
        }

        // Package als Objekt erzeugen u. globalem Vektor hinzufï¿½gen
        /* RPSPackage up = new RPSPackage(sPackageID, sPackageName);
         up.setParentPackage(parentPackageID);
         vGlobalPackages.add(up);

         if (!parentPackageID.equals("")) {
         RPSPackage parent = getPackage(parentPackageID);
         parent.addInnerPackage(sPackageID);
         }*/


        // zugehï¿½rige Klassen, Events, weitere Unterpackages etc. auslesen
        for (int i = 0; i < vSignals.size(); i++) {
            readClassSignalEventFromNode((Node) vSignals.elementAt(i));
        }

        for (int i = 0; i < vClasses.size(); i++) {
            readClassFromPackage((Node) vClasses.elementAt(i),
                    sPackageID, "");
        }

        //for (int i = 0; i < vInnerPackages.size(); i++)
        // readPackage((Node)vInnerPackages.elementAt(i), sPackageID);

        for (int i = 0; i < vDataTypes.size(); i++) {
            readDataTypeFromNode((Node) vDataTypes.elementAt(i));
        }

    }
}


/**
 * Reads the datatype from the passed node.
 * @param ndDatatype The node from the xmi tree.
 */
private void readDataTypeFromNode(final Node ndDatatype) {
    String sDatatypeID = "";
    String sDatatypeName = "";

    if (ndDatatype.hasChildNodes()) {
        // Read data
        sDatatypeID = ndDatatype.getAttributes().getNamedItem(
                attrID).getNodeValue();
        NodeList nlDTChildNodes = ndDatatype.getChildNodes();
```

```
            Node ndName = getFirstNodeFromList(nlDTChildNodes,
                    "Foundation.Core.ModelElement.name");
            sDatatypeName = ndName.getFirstChild().getNodeValue();
        }

        // BaseElement dt = new BaseElement(sDatatypeID, sDatatypeName);
        //vGlobalDatatypes.add(dt);

    }

    /**
     * Reads a statemachine from the node.
     * @param ndStateMachine The node from the xmi tree.
     * @param sClassID The id of the owning class.
     */
    private void readStateMachineFromNode(final Node ndStateMachine,
            final String sClassID) {
        String sMachineName = "";

        Vector vTransitions = new Vector();
        StateChart scChart = new StateChart();

        scChart.setModelSource(supportedExporter);
        scChart.setModelVersion(supportedExporterVersion);

        for (int i = 0; i < vGlobalSignalEvents.size(); i++) {
            scChart.addInputEvent((Event) vGlobalSignalEvents.get(i));
        }

        /* TODO: 2006-08-02
         * Distinguish the type of the attribute and add the variable of
         * the right type to the chart
         */
        for (int i = 0; i < vGlobalAttributes.size(); i++) {
            KielUMLAttribute ua = (KielUMLAttribute) vGlobalAttributes.get(i);
            IntegerVariable v = new IntegerVariable(ua.getAttrName());
            v.setIDManual(ua.getAttrID());
            scChart.addVariable(v);
        }


        NodeList nlMachineChildNodes = ndStateMachine.getChildNodes();

        if (ndStateMachine.getAttributes().getNamedItem("name") != null) {
            if (ndStateMachine.getAttributes().getNamedItem("name") != null) {
                sMachineName = ndStateMachine.getAttributes().getNamedItem(
                        "name").getNodeValue();
            }
        } else {
            sMachineName = "";
        }

        Node ndTop = getFirstNodeFromList(nlMachineChildNodes,
        "UML:StateMachine.top");
        Node ndTransitions = getFirstNodeFromList(nlMachineChildNodes,
        "UML:StateMachine.transitions");

        // Read states of the current statemachine and add them to the global
        // vector
        Node ndRootState = getFirstNodeFromListNoText(ndTop.getChildNodes());
        readStateFromStateNode(scChart,
                ndRootState, true, "");

        // Read transitions and add all available to the global vector
        if (ndTransitions != null) {
            Vector vTransitionNodes = getNodesFromList(
                    ndTransitions.getChildNodes(), "UML:Transition");

            if (vTransitions != null) {
                for (int i = 0; i < vTransitionNodes.size(); i++) {
                    Node ndTrans = (Node) vTransitionNodes.elementAt(i);
                    readTransitionFromNode(scChart, ndTrans, false, null);
                }
            }
        }

        vStateCharts.add(scChart);
        scChart.getRootNode().setName(sMachineName);
    }


    /**
     * This internally used method normalizes the ids found in xmi files
     * generated by ArgoUML.
     * @param id The id to be normalized.
     * @return The normalized id.
     */
    private String normalizeID(final String id) {

        String s = "";

        Object o = idDictionary.get(id);

        if (o == null) {
            s = "#" + (idCounter++);
            idDictionary.put(id.toString(), s.toString());
        } else {
            s = (String) o;
        }

        return s;
    }


    /**
     *
     * @param signalID The id of the signal to be returned.
     * @return The signal if found, null otherwise.
     */
    private Signal getSignalByID(final String signalID) {

        Signal result = null;
        Iterator iter = vGlobalSignalEvents.iterator();

        while (iter.hasNext()) {
            result = (Signal) iter.next();

            if (!result.getID().equals(signalID)) {
                result = null;
            } else {
```

```
                break;
            }
        }
        return result;

    }


    /**
     * Reads a transition from a node.
     * @param scChart The actually created statechart.
     * @param ndTransition The node from the xmi tree.
     * @param isInternal Denoting wether the current transition is internal.
     * @param trans The transition.
     */
    private void readTransitionFromNode(final StateChart scChart,
            final Node ndTransition, final boolean isInternal,
            final Transition trans) {
        String sTransitionID = "";
        String sTargetStateID = "";
        String sSignalEventID = "";
        String sSignalEventExp = "";
        String sGuardExpression = "";
        String sActionStatements = "";

        sTransitionID = normalizeID(ndTransition.getAttributes().getNamedItem(
                attrID).getNodeValue());
        NodeList nlTransitionChildNodes = ndTransition.getChildNodes();
        Node ndTarget = getFirstNodeFromList(nlTransitionChildNodes,
                "UML:Transition.target");
        if (ndTarget != null) {
            sTargetStateID = normalizeID(getFirstNodeFromListNoText(
                    ndTarget.getChildNodes()).getAttributes().getNamedItem(
                            attrIDRef).getNodeValue());
        }

        Node ndSignalEvent = getFirstNodeFromList(nlTransitionChildNodes,
                "UML:Transition.trigger");
        if (ndSignalEvent != null) {
            sSignalEventID = getFirstNodeFromListNoText(
                    ndSignalEvent.getChildNodes()).getAttributes().
                    getNamedItem(attrIDRef).getNodeValue();
        }

        // Trigger bzw. Guard auslesen
        Node ndTransGuard = getFirstNodeFromList(nlTransitionChildNodes,
        "UML:Transition.guard");
        if (ndTransGuard != null) {
            Node ndStatemachineGuard = getFirstNodeFromList(
                    ndTransGuard.getChildNodes(), "UML:Guard");
            if (ndStatemachineGuard != null) {
                Node ndGuardExpression = getFirstNodeFromList(
                        ndStatemachineGuard.getChildNodes(),
                        "UML:Guard.expression");
                if (ndGuardExpression != null) {
                    Node ndTemp = getFirstNodeFromList(
                            ndGuardExpression.getChildNodes(),
                            "UML:BooleanExpression");
                    if (ndTemp != null) {
                        sGuardExpression = ndTemp.getAttributes().getNamedItem(
                                "body").getNodeValue();
                    }
                }
            }
        }
        // Action auslesen
        Node ndTransEffect = getFirstNodeFromList(
                nlTransitionChildNodes,
                "UML:Transition.effect");
        if (ndTransEffect != null) {
            Node ndUninterpretedAction = getFirstNodeFromList(
                    ndTransEffect.getChildNodes(),
                    "UML:CallAction");
            if (ndUninterpretedAction != null) {
                Node ndScriptExpression = getFirstNodeFromList(
                        ndUninterpretedAction.getChildNodes(),
                        "UML:Action.script");
                if (ndScriptExpression != null) {
                    Node ndTemp = getFirstNodeFromList(
                            ndScriptExpression.getChildNodes(),
                            "UML:ActionExpression");
                    if (ndTemp != null) {
                        sActionStatements = ndTemp.getAttributes().getNamedItem(
                                "body").getNodeValue();
                    }
                }
            }
        }

        Transition t = null;
        if (isInternal) {
            t = trans;
        } else {
            kiel.dataStructure.Node target = null;

            Collection objects = scChart.getAllObjects();
            Iterator iter = objects.iterator();
            GraphicalObject go = null;
            while (iter.hasNext()) {
                go = null;
                try {
                    Object o = iter.next();
                    if (o instanceof GraphicalObject) {
                        go = (GraphicalObject) o;
                    } else {
                        System.out.println("Instance of "
                                + o.getClass().getName() + " found.");
                    }
                    if (go != null) {
                        if (go.getID().equals(sTransitionID)) {
                            t = (Transition) go;
                        }
                        if (go.getID().equals(sTargetStateID)) {
                            target = (kiel.dataStructure.Node) go;
                        }
                    }
                } catch (Exception e) {
                    System.out.println(
                            "Error while assigning transition with id "
                            + sTransitionID
```

```
                    + ". Found object with id " + go.getID());
            }
        }

        if ((t != null) && (target != null)) {
            t.setTarget(target);
        }
    }

    if ((t != null)
            && ((!sSignalEventID.equals(""))
                    || (!sGuardExpression.equals(""))
                    || (!sActionStatements.equals("")))) {
        // set the trigger, guard and effect
        // get the according signal event from the global vector
        if (!sSignalEventID.equals("")) {
            Signal s = getSignalByID(normalizeID(sSignalEventID));
            if (s != null) {
                sSignalEventExp = s.getName();
            } else {
                sSignalEventExp = "";
            }
        }
        String label = sSignalEventExp;
        if (!sGuardExpression.equals("")) {
            label += " [ " + sGuardExpression + " ]";
        }
        if (!sActionStatements.equals("")) {
            label += " /" + sActionStatements;
        }
        // new handling methodology use xmi.transitionLabel.parser
        if (!label.equals("")) {
            Parser p;
            try {
                // final statement
                p = new Parser(new Lexer(new PushbackReader(
                        new StringReader(label), maxbuffersize)));
                // Statement for testing purposes
                /* p = new Parser(new Lexer(new PushbackReader(
                        new StringReader("[a==5]/a=6;e;"),
                        1024)));*/
                        Start tree = p.parse();
                        LabelGenerator lg = new LabelGenerator(scChart);
                        tree.apply(lg);
                        t.setLabel(lg.getTransitionLabel());
            } catch (ParserException ex) {
                StringLabel l = new StringLabel(label);
                t.setLabel(l);
                System.out.println("The label string ("
                        + label
                        + ")could not be parsed: "
                        + ex.getMessage());
            } catch (Exception e) {
                FileInterfaceException ex = new FileInterfaceException(
                        "Handling of transition label failed.\n"
                        + "Found label: " + label + "\n"
                        + e.getMessage());
                StringLabel l = new StringLabel(label);
                t.setLabel(l);
                ex.showMessageBox();
            }
        } else {
            StringLabel l = new StringLabel(label);
            t.setLabel(l);
        }
    }
}

/**
 * Handle a statechart state node.
 * @param scChart The actual statechart.
 * @param ndState The node from the xmi tree.
 * @param isRoot Denoting if the actual read state will become the root
 * state.
 * @param parentStateID The id of the parent state.
 */
private void readStateFromStateNode(final StateChart scChart,
        final Node ndState, final boolean isRoot,
        final String parentStateID) {

    String sStateID = "";
    String sStateName = "";
    boolean bIsConcurrent = false;

    kiel.dataStructure.Node kielNode = null;


    NodeList nlStateChildNodes = ndState.getChildNodes();

    sStateID = normalizeID(
            ndState.getAttributes().getNamedItem(attrID).getNodeValue());


    if (ndState.getAttributes().getNamedItem("name") != null) {
        sStateName = ndState.getAttributes().getNamedItem(
                "name").getNodeValue();
    } else {
        sStateName = sStateID;
    }


    if (ndState.getNodeName().equals(tagPseudoState)) {
        String sKind = ndState.getAttributes().getNamedItem(
                "kind").getNodeValue();
        if (sKind.equals("choice")) {
            // TODO: Distinguish between Choice or DynamicChoice
            kielNode = new kiel.dataStructure.Choice();
        } else if (sKind.equals("deepHistory")) {
            kielNode = new kiel.dataStructure.DeepHistory();
        } else if (sKind.equals("fork")) {
            kielNode = new kiel.dataStructure.ForkConnector();
        } else if (sKind.equals("initial")) {
            kielNode = new kiel.dataStructure.InitialState();
        } else if (sKind.equals("join")) {
            kielNode = new kiel.dataStructure.Join();
        } else if (sKind.equals("junction")) {
            kielNode = new kiel.dataStructure.Junction();
        } else if (sKind.equals("shallowHistory")) {
            kielNode = new kiel.dataStructure.History();
```

```
            } else {
                FileInterfaceException ex = new FileInterfaceException(
                        "A parsed pseudostate could not be mapped to "
                        + " the KIEL datastructure.");
                ex.showMessageBox();
                //throw ex;
            }

        } else if (ndState.getNodeName().equals(tagCompositeState)) {
            String sConcurrent = ndState.getAttributes().getNamedItem(
                    "isConcurrent").getNodeValue();
            if (sConcurrent.equals("true")) {
                bIsConcurrent = true;
            }

            if (bIsConcurrent) {
                kielNode = new kiel.dataStructure.ANDState();
            } else {
                kielNode = new kiel.dataStructure.ORState();
            }


        } else if (ndState.getNodeName().equals(tagSimpleState)
                || ndState.getNodeName().equals(tagState)) {
            kielNode = new kiel.dataStructure.SimpleState();


        } else if (ndState.getNodeName().equals(tagFinalState)) {
            kielNode = new kiel.dataStructure.FinalSimpleState();


        } else if (ndState.getNodeName().equals(tagSynchState)) {
            kielNode = new kiel.dataStructure.SynchState();


        } else {
            FileInterfaceException ex = new FileInterfaceException(
                    "A parsed tag name could not be mapped to "
                    + " the KIEL datastructure.");
            ex.showMessageBox();
            //throw ex;
            return;
        }

        if (kielNode == null) {
            kielNode = new kiel.dataStructure.SimpleState();
        }

        kielNode.setIDManual(sStateID);
        kielNode.setName(sStateName);

        // ausgehende Transitionen
        Node ndOutgoing = getFirstNodeFromList(nlStateChildNodes,
                "UML:StateVertex.outgoing");
        if (ndOutgoing != null) {
            NodeList nlOut = ndOutgoing.getChildNodes();
            for (int i = 0; i < nlOut.getLength(); i++) {
                Node ndOutTrans = nlOut.item(i);
                if (ndOutTrans.getNodeType() == Node.ELEMENT_NODE) {
                    Transition t;
                    if (kielNode instanceof InitialState) {
                        t = new InitialArc();
                    } else if (kielNode instanceof Choice) {
                        t = new ConditionalTransition();
                    } else {
                        t = new Transition();
                    }
                    t.setIDManual(normalizeID(
                            ndOutTrans.getAttributes().getNamedItem(
                                    attrIDRef).getNodeValue()));
                    t.setSource(kielNode);
                }
            }
        }

        // interne Transitionen
        Node ndInner = getFirstNodeFromList(nlStateChildNodes,
                "UML:State.internalTransition");
        if (ndInner != null) {
            NodeList nlInner = ndInner.getChildNodes();
            Iterator iter = null;
            Collection objects = kielNode.getOutgoingTransitions();
            Transition t = null;
            for (int i = 0; i < nlInner.getLength(); i++) {
                if (nlInner.item(i).getNodeType() == Node.ELEMENT_NODE) {
                    // Internal transitions have been created by
                    // reading the outgoing transitions
                    // so get it from the collection of all objects
                    t = null;
                    iter = objects.iterator();
                    String sTransID = normalizeID(
                            nlInner.item(i).getAttributes().getNamedItem(
                                    attrID).getNodeValue());

                    while (iter.hasNext()) {
                        t = (Transition) iter.next();

                        if (t.getID().equals(sTransID)) {
                            break;
                        } else {
                            t = null;
                        }
                    }
                    if (t != null) {
                        //t.setTarget(kielNode);
                        kielNode.removeOutgoingTransition(t);
                        ((State) kielNode).setInternalTransition(t);

                        readTransitionFromNode(scChart, nlInner.item(i),
                                true, t);
                    }
                }
            }
        }


        // Entry
        /*Node ndEntry = getFirstNodeFromList(nlStateChildNodes,
          "UML:State.entry");
        if (ndEntry != null) {
```

```
  NodeList nlEntry = ndEntry.getChildNodes();
  Node ndActionSequence = getFirstNodeFromList(ndEntry.getChildNodes(),
   "Behavioral_Elements.Common_Behavior.ActionSequence");
  Node ndActionSequenceAction = getFirstNodeFromList(
      ndActionSequence.getChildNodes(),
      "Behavioral_Elements.Common_Behavior.ActionSequence.action");
  Node ndUninterpretedAction = getFirstNodeFromList(
    ndActionSequenceAction.getChildNodes(),
    "Behavioral_Elements.Common_Behavior.UninterpretedAction");
  if (ndUninterpretedAction != null) {
    String sID = ndUninterpretedAction.getAttributes().getNamedItem(
      AttrID).getNodeValue();
    String sActions = getNameFromNodeList(
      ndUninterpretedAction.getChildNodes());
    RPSTransition innerTrans = new RPSTransition(sID);
    innerTrans.setActionStatements(sActions);
    innerTrans.setSignalEventID(""); // kein Trigger und keine Guards
    innerTrans.setGuardExpression("");
    innerTrans.setClassID(classID);
    innerTrans.setSourceID(sStateID);
    innerTrans.setTargetID(sStateID);
    innerTrans.setType(Transition.ENTRY);
    vGlobalTransitions.add(innerTrans);
    vTransInternal.add(innerTrans.getID());
  }
}
// EXIT
Node ndExit = getFirstNodeFromList(nlStateChildNodes, "UML:State.exit");
if (ndExit != null) {
  NodeList nlExit = ndExit.getChildNodes();
  Node ndActionSequence = getFirstNodeFromList(
      ndExit.getChildNodes(),
      "Behavioral_Elements.Common_Behavior.ActionSequence");
  Node ndActionSequenceAction = getFirstNodeFromList(
    ndActionSequence.getChildNodes(),
    "Behavioral_Elements.Common_Behavior.ActionSequence.action");
  Node ndUninterpretedAction = getFirstNodeFromList(
      ndActionSequenceAction.getChildNodes(),
    "Behavioral_Elements.Common_Behavior.UninterpretedAction");
  if (ndUninterpretedAction != null) {
    String sID = ndUninterpretedAction.getAttributes().getNamedItem(
        AttrID).getNodeValue();
    String sActions = getNameFromNodeList(
        ndUninterpretedAction.getChildNodes());
    RPSTransition innerTrans = new RPSTransition(sID);
    innerTrans.setActionStatements(sActions);
    innerTrans.setSignalEventID(""); // kein Trigger und keine Guards
    innerTrans.setGuardExpression("");
    innerTrans.setClassID(classID);
    innerTrans.setSourceID(sStateID);
    innerTrans.setTargetID(sStateID);
    innerTrans.setType(Transition.EXIT);
    vGlobalTransitions.add(innerTrans);
    vTransInternal.add(innerTrans.getID());
  }
}*/

// deferrableEvents
Node ndDef = getFirstNodeFromList(nlStateChildNodes,
       "UML:State.deferrableEvent");
if (ndDef != null) {
   NodeList nlDefEvents = ndDef.getChildNodes();

   for (int i = 0; i < nlDefEvents.getLength(); i++) {
       Node ndDefEvent = nlDefEvents.item(i);
       if (ndDefEvent.getNodeType() == Node.ELEMENT_NODE) {

           String nodeId = normalizeID(
                  ndDefEvent.getAttributes().getNamedItem(
                         attrIDRef).getNodeValue());
           Signal s = getSignalByID(nodeId);
           Signal newSignal = new Signal(s.getName());

           ((State) kielNode).addDeferrableEvent(newSignal);

       }
   }
}

if (isRoot) {
   try {
       scChart.setRootNode((CompositeState) kielNode);
   } catch (Exception e) {
       FileInterfaceException ex = new FileInterfaceException(
              "The XMI file contains a root node of a wrong type.");
       ex.showMessageBox();
       //throw ex;
   }
} else {
   try {
       Collection allObjects = scChart.getAllObjects();
       Iterator iter = allObjects.iterator();
       kiel.dataStructure.GraphicalObject go = null;

       while (iter.hasNext()) {
          go = (kiel.dataStructure.GraphicalObject) iter.next();
          if (go.getID().equals(parentStateID)) {
              break;
          } else {
              go = null;
          }
       }

       if (go != null) {

          if ((go instanceof ANDState)
                 && (kielNode instanceof CompositeState)) {
              Region r = new Region();
              r.setIDManual(kielNode.getID());
              r.setName(kielNode.getName());
              ((ANDState) go).addSubnode(r);

              if (((ANDState) go).getSubnodes().size() > 1) {
                 DelimiterLine delLine = new DelimiterLine();

                 ((ANDState) go).addDelimiterLine(delLine);
              }
          } else {
              ((CompositeState) go).addSubnode(kielNode);
          }
```

```
                scChart.setChanged();
            }
        } catch (Exception e) {
            FileInterfaceException ex = new FileInterfaceException(
                    "Adding the node with the ID " + kielNode.getID()
                    + " to the datastructure was not possible.");
            ex.showMessageBox();
            //throw ex;

        }
    }

    // Rekursion
    Node ndSubVertex = getFirstNodeFromList(nlStateChildNodes,
            "UML:CompositeState.subvertex");
    if (ndSubVertex != null) {
        NodeList nlSubVertex = ndSubVertex.getChildNodes();
        for (int i = 0; i < nlSubVertex.getLength(); i++) {
            Node ndSubState = nlSubVertex.item(i);
            if (ndSubState.getNodeType() == Node.ELEMENT_NODE) {
                readStateFromStateNode(scChart, ndSubState, false,
                        sStateID);
            }
        }
    }

}

/**
 * Read a signal from the xmi document.
 * @param ndSignal The node from the xmi tree.
 */
private void readClassSignalEventFromNode(final Node ndSignal) {
    // nur Referenz auf eigentliches Signal
    String sSignalEvtID = "";
    String sSignalName = "";
    sSignalEvtID = ndSignal.getAttributes().getNamedItem(
            attrID).getNodeValue();
    sSignalName = ndSignal.getAttributes().getNamedItem(
            "name").getNodeValue();

    Signal s = new Signal(sSignalName);
    s.setIDManual(normalizeID(sSignalEvtID));

    vGlobalSignalEvents.add(s);
}


/**
 * An attribute from the xmi.
 * @param ndAttribute The node from the xmi tree.
 */
private void readClassAttributeFromNode(final Node ndAttribute) {
    String sAttributeID = "";
    String sAttributeName = "";
    String sDataTypeID = "";

    sAttributeID = ndAttribute.getAttributes().getNamedItem(
            attrID).getNodeValue();
    sAttributeName = ndAttribute.getAttributes().getNamedItem(
            "name").getNodeValue();

    NodeList nlAttrChildNodes = ndAttribute.getChildNodes();


    Node ndDT = getFirstNodeFromList(nlAttrChildNodes,
            "UML:StructuralFeature.type");

    sDataTypeID = getFirstNodeFromListNoText(
            ndDT.getChildNodes()).getAttributes().getNamedItem(
                    attrIDRef).getNodeValue();

    KielUMLAttribute ua = new KielUMLAttribute(sAttributeID,
            sAttributeName, sDataTypeID);

    vGlobalAttributes.add(ua);
}



/**
 * Handle a class.
 * @param ndClass The node from the xmi tree.
 * @param sPackageID The id of the owning package.
 * @param sParentClass The name of the owning class.
 */
private void readClassFromPackage(final Node ndClass,
        final String sPackageID, final String sParentClass) {

    String className = "";
    String classID = "";

    Vector vStateMachines = null;
    Vector vSignalEvents = null;
    Vector vFeatureAttr = null;
    Vector vDataTypes = null;
    Vector vClasses = null;


    if (ndClass.hasChildNodes()) {
        // Daten der Klasse auslesen
        classID = ndClass.getAttributes().getNamedItem(
                attrID).getNodeValue();
        NodeList nlClassChildNodes = ndClass.getChildNodes();
        className = ndClass.getAttributes().getNamedItem(
                "name").getNodeValue();

        KielUMLDataType dt = new KielUMLDataType(className, classID);
        if (dt.isPrimitive()) {
            vGlobalDatatypes.add(dt);
        }


        Node ndOwnedElement = getFirstNodeFromList(nlClassChildNodes,
                "UML:Namespace.ownedElement");
        if (ndOwnedElement != null) {
            NodeList nlOwned = ndOwnedElement.getChildNodes();
            vStateMachines = getNodesFromList(nlOwned,
                    "UML:StateMachine");
            vSignalEvents = getNodesFromList(nlOwned,
```

```
            "UML:SignalEvent");
        //vCallEvents = getNodesFromList(nlOwned, "UML:CallEvent");
        vDataTypes = getNodesFromList(nlOwned, "UML:Datatype");
        vClasses = getNodesFromList(nlOwned, "UML:Class");
    }
    Node ndFeature = getFirstNodeFromList(nlClassChildNodes,
            "UML:Classifier.feature");
    if (ndFeature != null) {
        NodeList nlFeatures = ndFeature.getChildNodes();
        vFeatureAttr = getNodesFromList(nlFeatures,
                "UML:Attribute");
    }
}

if (vStateMachines != null) {

    if (vDataTypes != null) {
        for (int i = 0; i < vDataTypes.size(); i++) {
            readDataTypeFromNode((Node) vDataTypes.elementAt(i));
        }
    }

    if (vClasses != null) {
        for (int i = 0; i < vClasses.size(); i++) {
            readClassFromPackage((Node) vClasses.elementAt(i),
                    sPackageID, classID);
        }
    }

}

if (vFeatureAttr != null) {
    for (int i = 0; i < vFeatureAttr.size(); i++) {
        readClassAttributeFromNode(
                (Node) vFeatureAttr.elementAt(i));

        }
    }

    if (vSignalEvents != null) {
        for (int i = 0; i < vSignalEvents.size(); i++) {
            readClassSignalEventFromNode(
                    (Node) vSignalEvents.elementAt(i));
            }
        }

        /*if (vCallEvents != null) {
            for (int i = 0; i < vCallEvents.size(); i++) {

            }
        }*/

        for (int i = 0; i < vStateMachines.size(); i++) {
            readStateMachineFromNode((Node) vStateMachines.elementAt(i),
                    "");
            }
        }
    }
}
```

G.4.4. KielUMLAttribute.java

```
package kiel.fileInterface.xmi.common;

/**
 * <p>Description: Helper class for UML Attribute information. </p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.2 $ last modified $Date: 2006/11/08 10:26:21 $
 */
public class KielUMLAttribute {


    /**
     * The id of the attribute.
     */
    private String attrID;

    /**
     * The name of the attribute.
     */
    private String attrName;

    /**
     * The id of the datatype.
     */
    private String dataTypeID;


    /**
     * Creator of an Attribute.
     * @param aID The id of the attribute.
     * @param aName The name of the attribute
     * @param aDataTypeID The id of the associated datatype.
     */
    public KielUMLAttribute(final String aID,
           final String aName, final String aDataTypeID) {
        this.attrID = aID;
        this.attrName = aName;
        this.dataTypeID = aDataTypeID;
    }


    /**
     * Returns the ID of the attribute.
     * @return A String containing the ID.
     */
    public final String getAttrID() {
        return attrID;
    }


    /**
     * Set the ID of the attribute.
     * @param sAttrID The new ID.
     */
    public final void setAttrID(final String sAttrID) {
        this.attrID = sAttrID;
    }


    /**
     * Returns the name of the attribute.
     * @return The name of the attribute.
     */
    public final String getAttrName() {
        return attrName;
    }


    /**
     * Set the name of the attribute.
     * @param sAttrName A name.
     */
    public final void setAttrName(final String sAttrName) {
        this.attrName = sAttrName;
    }


    /**
     * Returns the ID of the attriburte's datatype.
     * @return An ID.
     */
    public final String getDataTypeID() {
        return dataTypeID;
    }


    /**
     * Set the ID of the attribute's datatype.
     * @param sDataTypeID An ID.
     */
    public final void setDataTypeID(final String sDataTypeID) {
        this.dataTypeID = sDataTypeID;
    }
}
```

G.4.5. KielUMLDataType.java

```
package kiel.fileInterface.xmi.common;


/**
 * <p>Description: Helper class for the Datatype translation. </p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.2 $ last modified $Date: 2006/11/08 10:26:21 $
 */
public class KielUMLDataType {


    /**
     * The array for mapping primitive types.
     */
    private static String[] primitiveTypes =
    {"int", "integer", "int8", "int16", "int32",
    "bool", "boolean",
    "float", "numeric", "long"};

    /**
     * The array of possible integer types.
     */
    private static String[] integerNames =
    {"int", "integer", "int8", "int16", "int32"};

    /**
     * The array of possible floating types.
     */
    private static String[] floatNames =
    {"float", "numeric", "long"};

    /**
     * The array of possible names for boolean types.
     */
    private static String[] boolNames =
    {"bool", "boolean"};


    /**
     * The name of the data type.
     */
    private String name;

    /**
     * The id of the datatype.
     */
    private String id;

    /**
     * Creator for a datatype wrapper.
     * @param sName Name of the datatype.
     * @param sID ID from the XMI file.
     */
    public KielUMLDataType(final String sName, final String sID) {
        super();
        this.name = sName;
        this.id = sID;
    }

    /**
     * .
     * @return .
     */
    public final String getId() {
        return id;
    }

    /**
     * .
     * @param sID .
     */
    public final void setId(final String sID) {
        this.id = sID;
    }

    /**
     * .
     * @return .
     */
    public final String getName() {
        return name;
    }

    /**
     * .
     * @param sName .
     */
    public final void setName(final String sName) {
        this.name = sName;
    }



    /**
     * Decides if the datatype is a primitive.
     * @return Boolean value.
     */
    public final boolean isPrimitive() {
        boolean result = false;

        for (int i = 0; i < primitiveTypes.length; i++) {
            result = primitiveTypes[i].equalsIgnoreCase(name);
            if (result) {
                break;
            }
        }
        return result;
    }
```

```
    /**
     * Return the Class of primitives.
     * @return A Class.
     */
    public final Class getPrimitive() {
        Class result = null;

        for (int i = 0; i < integerNames.length; i++) {
            if (integerNames[i].equalsIgnoreCase(name)) {
                result = int.class;
            }
        }

        if (result != null) {
            return result;
        }

        for (int i = 0; i < floatNames.length; i++) {
            if (floatNames[i].equalsIgnoreCase(name)) {
                result = float.class;
            }
        }

        if (result != null) {
            return result;
        }

        for (int i = 0; i < boolNames.length; i++) {
            if (boolNames[i].equalsIgnoreCase(name)) {
                result = boolean.class;
            }
        }

        return result;
    }

}
```

G.4.6. LabelGenerator.java

```
package kiel.fileInterface.xmi.ArgoUML;

import kiel.dataStructure.action.Action;
import kiel.dataStructure.action.GenerateEvent;
import kiel.dataStructure.action.IntegerAssignment;
import kiel.dataStructure.boolexp.*;
import kiel.dataStructure.eventexp.*;
import kiel.dataStructure.intexp.*;

import java.util.Iterator;
import java.util.Stack;

import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.TransitionLabel;
import kiel.fileInterface.FileInterfaceException;
import kiel.fileInterface.JavaTransitionLabels.analysis.DepthFirstAdapter;
import kiel.fileInterface.JavaTransitionLabels.node.*;

/**
 * <p>Description: Label generator called by the ArgoUMLReader.
 * This class extends a DepthFirstAdapter generated with SableCC. </p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/11/08 10:26:20 $
 */
public class LabelGenerator extends DepthFirstAdapter {

    /**
     * The statechart of which the generated label is part of.
     */
    private StateChart sc;

    /**
     * A stack for parsing.
     */
    private Stack stack;


    /**
     * The generated label.
     */
    private CompoundLabel label;

    /**
     * Flag for parsing.
     */
    private boolean inTriggerMode = false;

    /**
     * Flag for parsing.
     */
    private boolean inGuardMode = false;

    /**
     * Flag for parsing.
     */
    private boolean inActionMode = false;

    /**
     * Flag for parsing.
     */
    private boolean inAssignment = false;

    /**
     * Internal method to show the passed error in a message box.
     * @param aError String description of the error.
     */
    private void printError(final String aError) {
        FileInterfaceException ex = new FileInterfaceException(aError);
        ex.showMessageBox();
        //System.out.println(aError);
    }


    /**
     * Creator for the LabelGenerator.
     * The labels created are referencing the variables and signals found
     * within the passed statechart.
     * @param chart The chart to look for variable names.
     */
    public LabelGenerator(final StateChart chart) {
        super();
        sc = chart;
        label = new CompoundLabel();

        stack = new Stack();
    }

    /**
     * Returns the created transition label.
     * @return The transition label object.
     */
    public final TransitionLabel getTransitionLabel() {
        return label;
    }


    /**
     * @param node The trigger node.
     */
    public final void inATrigger(final ATrigger node) {
        inTriggerMode = true;
    }

    /**
     * Message handler for the event when a trigger node is left.
     * @param node The trigger node.
     */
```

```
    public final void outATrigger(final ATrigger node) {

        label.setTrigger((DelayExpression) stack.pop());

        inTriggerMode = false;
    }


    /**
     * Event handler called when an Expression statement is left.
     * @param node .
     */
    public final void outAExpressionStatement(final AExpressionStatement node) {
        label.getEffect().addAction((Action) stack.pop());
    }

    /**
     * Event handler called when a single effect statement is left.
     * @param node .
     */
    public final void outASingleEffect(final ASingleEffect node) {
        label.getEffect().addAction((Action) stack.pop());
    }


    /**
     * Event handler called when a single effect statement is entered.
     * @param node .
     */
    public final void inASingleEffect(final ASingleEffect node) {
        inActionMode = true;
    }

    /**
     * Event handler called when a multiple effect statement is entered.
     * @param node .
     */
    public final void inAMultipleEffect(final AMultipleEffect node) {
        inActionMode = true;
    }

    /**
     * Called when a assignement statement is entered.
     * @param node .
     */
    public final void inAAssignment(final AAssignment node) {
        inAssignment = true;
    }

    /**
     * Called when an assignement statement is left.
     * @param node .
     */
    public void outAAssignment(final AAssignment node) {
        IntegerExpression exp = (IntegerExpression) stack.pop();
        IntegerVariable var = (IntegerVariable) stack.pop();

        IntegerAssignment assignment = new IntegerAssignment(var, exp);
        stack.push(assignment);
        inAssignment = false;
    }


    /**
     * Called when a guard is entered.
     * @param node .
     */
    public final void inAGuard(final AGuard node) {
        stack.clear();
        inGuardMode = true;
    }

    /**
     * Called when a guard is left.
     * @param node .
     */
    public final void outAGuard(final AGuard node) {
        label.setCondition((BooleanExpression) stack.pop());
        inGuardMode = false;
        stack.clear();
    }


    /**
     * Called when a statement surrounded by brackets is left.
     * @param node .
     */
    public final void outALParenthesePrimaryNoNewArray(
            final ALParenthesePrimaryNoNewArray node) {
        Object item;
        item = stack.pop();
        BooleanBrackets bracket =
            new BooleanBrackets((BooleanExpression) item);
        stack.push(bracket);
    }

    /**
     * Called when a decimal integer literal is parsed.
     * @param node .
     */
    public final void inADecimalIntegerLiteral(final ADecimalIntegerLiteral node) {
        IntegerConstant constant = new IntegerConstant();
        constant.setValue(Integer.parseInt(
                node.getDecimalIntegerLiteral().getText()));
        stack.push(constant);
    }


    /**
     * Internally used method to get the integer expression from the statechart
     * by the passed name.
     * @param itemName Name of the integer variable.
     * @return The IntegerExpression object from the statechart if found.
     */
    private IntegerExpression getVariableByName(final String itemName) {

        IntegerVariable v = null;

        Iterator iter = sc.getAllVariables().iterator();
        while (iter.hasNext()) {
```

```
            v = (IntegerVariable) iter.next();

            if (v.getName().equals(itemName)) {
                return v;
            }
        }
        return null;
    }

    /**
     * Returns the signal object for the passed signal name.
     * @param signalName Name of the signal to return
     * @return Event object if a matching signal was found.
     */
    private Event getSignalByName(final String signalName) {

        Event ev = null;

        Iterator iter = sc.getInputEvents().iterator();
        while (iter.hasNext()) {
            ev = (Event) iter.next();
            if (ev.getName().equals(signalName)) {
                return ev;
            }
        }
        return null;
    }

    /**
     * Called when a simple name was parsed.
     * @param node .
     */
    public final void inASimpleName(final ASimpleName node) {
        // TODO: As of 2006-08-02:
        // All variables and Signals in the stacks
        // are of type integer for the sake of simplicity
        // IntegerVariable v = new IntegerVariable(
        // node.getIdentifier().getText(),
        // null);

        Object o = null;
        if (inGuardMode) {
            o = getVariableByName(node.getIdentifier().getText());
        }
        if (inTriggerMode) {
            o = new DelayExpression();
            ((DelayExpression) o).setEventExpression(
                    getSignalByName(node.getIdentifier().getText()));
        }
        if (inActionMode) {
            if (inAssignment) {
                o = getVariableByName(node.getIdentifier().getText());
            } else {
                Event e = getSignalByName(node.getIdentifier().getText());
                o = new GenerateEvent(e);
            }
        }
        if (o != null) {
            stack.push(o);
        } else {
            if ((inTriggerMode || (inActionMode && !inAssignment))) {
                o = new Signal("error_"
                        + node.getIdentifier().getText());
                if (!inTriggerMode) {
                    o = new GenerateEvent((Event) o);
                }
            } else {
                o = new IntegerVariable("error_"
                        + node.getIdentifier().getText());
            }
            // push the event on the stack. otherwise, an accessviolation
            // will occur.
            stack.push(o);

            printError("The simple name \""
                    + node.getIdentifier().getText()
                    + "\" at position ["
                    + node.getIdentifier().getLine()
                    + ","
                    + node.getIdentifier().getPos()
                    + "] was not resolved.");
        }
    }

    /**
     * Called when an Equality espression was parsed.
     * @param node .
     */
    public final void outAEqEqualityExpression(
            final AEqEqualityExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();
        Equal e = new Equal(left, right);
        stack.push(e);
    }

    /**
     * Called when a parsed non equality expression is left.
     * @param node .
     */
    public final void outANeqEqualityExpression(
            final ANeqEqualityExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();
        NotEqual neq = new NotEqual(left, right);
        stack.push(neq);
    }


    /**
     * Called when a two part conditional expression combined with and is left.
     * @param node .
     */
    public final void outAConditionalAndExpressionConditionalAndExpression(
            final AConditionalAndExpressionConditionalAndExpression node) {
```

```
        BooleanExpression right = (BooleanExpression) stack.pop();
        BooleanExpression left = (BooleanExpression) stack.pop();

        BooleanAnd and = new BooleanAnd(left, right);
        stack.push(and);
    }


    /**
     * Called when a two part conditional expression combined with or is left.
     * @param node .
     */
    public final void outAConditionalOrExpressionConditionalOrExpression(
            final AConditionalOrExpressionConditionalOrExpression node) {
        BooleanExpression right = (BooleanExpression) stack.pop();
        BooleanExpression left = (BooleanExpression) stack.pop();

        BooleanOr or = new BooleanOr(left, right);
        stack.push(or);
    }


    /**
     * Called when a less equal than expression is left.
     * @param node .
     */
    public final void outALteqRelationalExpression(
            final ALteqRelationalExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();

        LessOrEqual lteq = new LessOrEqual(left, right);
        stack.push(lteq);
    }

    /**
     * Called when a less than expression is left.
     * @param node .
     */
    public final void outALtRelationalExpression(
            final ALtRelationalExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();

        LessThan lt = new LessThan(left, right);
        stack.push(lt);
    }

    /**
     * Called when a greater than equal expression is left.
     * @param node .
     */
    public final void outAGteqRelationalExpression(
            final AGteqRelationalExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();

        GreaterOrEqual gteq = new GreaterOrEqual(left, right);
        stack.push(gteq);
    }

    /**
     * Called when a greater than expression is left.
     * @param node .
     */
    public final void outAGtRelationalExpression(
            final AGtRelationalExpression node) {
        IntegerExpression right = (IntegerExpression) stack.pop();
        IntegerExpression left = (IntegerExpression) stack.pop();

        GreaterThan gt = new GreaterThan(left, right);
        stack.push(gt);
    }

    /**
     * Called when a complementary expression is left.
     * @param node .
     */
    public final void outAComplementUnaryExpressionNotPlusMinus(
            final AComplementUnaryExpressionNotPlusMinus node) {
        BooleanExpression ex = (BooleanExpression) stack.pop();
        BooleanNot not = new BooleanNot(ex);
        stack.push(not);
    }


}
```

G.4.7. XMIGenerator.java

```
package kiel.fileInterface.xmi;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Collection;
import java.util.Iterator;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Namespace;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

import kiel.dataStructure.StateChart;
import kiel.dataStructure.Transition;


/**
 * <p>Description: This class is used to write an instance of a statechart
 * to a XMI file.</p>
 * <p>Copyright: (c) 2006</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:kbe@informatik.uni-kiel.de">Ken Bell</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/11/08 10:26:20 $
 */
public class XMIGenerator {


    /**
     * The name of the exporter.
     * It is written to the Meta-Tag section of the generated XMI file.
     */
    private static String sXMIEXPORTERNAME =
        "ArgoUML (using Netbeans XMI Writer version 1.0)";
    // "KIEL (Kiel integrated environment for Layout) - XMI Exporter";

    /**
     * The version of the exporter.
     * It is written to the Meta-Tag section of the generated XMI file.
     */
    private static String sXMIEXPORTERVERSION =
        "0.20.x";


    /**
     * The generated xmi document.
     */
    private Document output;

    /**
     * The namespace to use for the generated document.
     */
    private Namespace nsUML;


    /**
     * Writes the Meta-Tag information to the document.
     */
    private void writeXMIHeader() {
        Element root = new Element("XMI");
        nsUML = Namespace.getNamespace("UML", "org.omg.xmi.namespace.UML");
        root.setAttribute("xmi.version", "1.2");

        root.addNamespaceDeclaration(nsUML);
        output.setRootElement(root);

        Element header = new Element("XMI.header");
        Element header2 = new Element("XMI.header");
        header.addContent(header2);
        Element documentation = new Element("XMI.documentation");
        header2.addContent(documentation);
        Element exporter = new Element("XMI.exporter");
        exporter.setText(sXMIEXPORTERNAME);
        Element version = new Element("XMI.exporterVersion");
        version.setText(sXMIEXPORTERVERSION);
        Element metamodel = new Element("XMI.metamodel");
        metamodel.setAttribute("xmi.name", "UML");
        metamodel.setAttribute("xmi.version", "1.4");
        documentation.addContent(exporter);
        documentation.addContent(version);
        documentation.addContent(metamodel);
        root.addContent(header);
    }

    /**
     * Creates a package stub and an class stub in the XMI file.
     * The statechart is assigned to the class afterwards.
     * The model is given the name of the exported statechart.
     * @param chartName The name of the statechart.
     * @return An XMI element.
     */
    private Element writeStateChartPrequel(final String chartName) {

        Element content = new Element("XMI.content");
        output.getRootElement().addContent(content);

        Element model = new Element("Model", nsUML);
        content.addContent(model);
        model.setAttribute("xmi.id", "a");
        model.setAttribute("name", chartName);
        model.setAttribute("isSpecification", "false");
        model.setAttribute("isRoot", "false");
        model.setAttribute("isLeaf", "false");
        model.setAttribute("isAbstract", "false");

        Element nsOwned = new Element("Namespace.ownedElement", nsUML);
        model.addContent(nsOwned);

        Element eleClass = new Element("Class", nsUML);
```

```
        nsOwned.addContent(eleClass);

        eleClass.setAttribute("xmi.id", "#class01");
        eleClass.setAttribute("name", "");
        eleClass.setAttribute("visibility", "public");
        eleClass.setAttribute("name", "");
        eleClass.setAttribute("isSpecification", "false");
        eleClass.setAttribute("isRoot", "false");
        eleClass.setAttribute("isLeaf", "false");
        eleClass.setAttribute("isAbstract", "false");
        eleClass.setAttribute("isActive", "false");

        nsOwned = new Element("Namespace.ownedElement", nsUML);
        eleClass.addContent(nsOwned);

        return nsOwned;
    }

    /**
     * Writes informations about transitions to the XMI file in the state
     * tag.
     * @param label The name of the statevertex.
     * @param transitions The collection of all transitions.
     * @param ele The XMI element.
     */
    private void handleTransitions(final String label,
            final Collection transitions, final Element ele) {

        if (transitions.size() < 1) {
            return;
        }

        Element transVertex = new Element("StateVertex." + label, nsUML);
        ele.addContent(transVertex);

        Iterator iter = transitions.iterator();
        while (iter.hasNext()) {
            Transition t = (Transition) iter.next();
            Element eleTrans = new Element("Transition", nsUML);
            eleTrans.setAttribute("xmi.idref", t.getID());
            transVertex.addContent(eleTrans);
        }

    }

    /**
     * Returns the Tag for the passed KIEL node.
     * @param n The node from the statechart.
     * @return A string containing the type string of the node.
     */
    private String getUMLNodeTagFromKielNode(final kiel.dataStructure.Node n) {
        String result = "";
        if (n instanceof kiel.dataStructure.CompositeState) {
            result = "CompositeState";
        } else if (n instanceof kiel.dataStructure.FinalSimpleState) {
            result = "FinalState";
        } else if (n instanceof kiel.dataStructure.SimpleState) {
            result = "SimpleState";
        } else if (n instanceof kiel.dataStructure.PseudoState) {
            result = "Pseudostate";
        }
        return result;
    }

    /**
     * This method recursively traverses the tree structure.
     * @param state The current state.
     * @param ele The current XMI element.
     */
    private void writeState(final kiel.dataStructure.Node state,
            final Element ele) {
        Element eleState = null;

        eleState = new Element(getUMLNodeTagFromKielNode(state), nsUML);

        ele.addContent(eleState);
        eleState.setAttribute("xmi.id", state.getID());
        eleState.setAttribute("name", state.getName());
        eleState.setAttribute("isSpecification", "false");
        if (state instanceof kiel.dataStructure.CompositeState) {
            eleState.setAttribute("isConcurrent",
                    Boolean.toString(
                            state instanceof kiel.dataStructure.ANDState));
        } else if (state instanceof kiel.dataStructure.PseudoState) {
            String kind = "";
            if (state instanceof kiel.dataStructure.ForkConnector) {
                kind = "fork";
            } else if (state instanceof kiel.dataStructure.InitialState) {
                kind = "initial";
            } else if (state instanceof kiel.dataStructure.Join) {
                kind = "join";
            } else if (state instanceof kiel.dataStructure.History) {
                kind = "shallowHistory";
            } else if (state instanceof kiel.dataStructure.DeepHistory) {
                kind = "deepHistory";
            } else if (state instanceof kiel.dataStructure.Junction) {
                kind = "junction";
            } else if (state instanceof kiel.dataStructure.Choice) {
                kind = "choice";
            }
            eleState.setAttribute("kind", kind);
        }

        if (eleState != null) {
            handleTransitions("incoming", state.getIncomingTransitions(),
                    eleState);
            handleTransitions("outgoing", state.getOutgoingTransitions(),
                    eleState);
        }

        if (state instanceof kiel.dataStructure.CompositeState) {
            Element subVertex = new Element("CompositeState.subvertex", nsUML);
            eleState.addContent(subVertex);


            kiel.dataStructure.CompositeState comp =
                (kiel.dataStructure.CompositeState) state;
            for (int i = 0; i < comp.getSubnodes().size(); i++) {
                writeState((kiel.dataStructure.Node) comp.getSubnodes().get(i),
                        subVertex);
```

```
            }
        }

    }


    /**
     * Writes information about all transitions after all state information
     * was processed to the XMI file.
     * @param transitions The XMI element to write the transition information
     * to.
     * @param chart The handled statechart.
     */
    private void writeTransitions(final Element transitions,
            final StateChart chart) {

        Iterator iter = chart.getAllObjects().iterator();
        while (iter.hasNext()) {
            Object o = iter.next();
            if (o instanceof Transition) {
                Transition t = (Transition) o;

                Element eleTrans = new Element("Transition", nsUML);
                transitions.addContent(eleTrans);

                eleTrans.setAttribute("xmi.id", t.getID());
                eleTrans.setAttribute("isSpecification", "false");

                Element source = new Element("Transition.source", nsUML);
                Element target = new Element("Transition.target", nsUML);
                Element srcInstance = new Element(
                        getUMLNodeTagFromKielNode(t.getSource()), nsUML);
                srcInstance.setAttribute("xmi.idref", t.getSource().getID());
                source.addContent(srcInstance);
                Element trgInstance = new Element(
                        getUMLNodeTagFromKielNode(t.getTarget()), nsUML);
                trgInstance.setAttribute("xmi.idref", t.getTarget().getID());
                target.addContent(trgInstance);

                eleTrans.addContent(source);
                eleTrans.addContent(target);
            }
        }

    }


    /**
     * Writes the statechart information to the XMI Document.
     * @param chart The statechart to export.
     */
    private void writeStateChart(final StateChart chart) {

        Element ownedElement = writeStateChartPrequel(
                chart.getRootNode().getName());

        Element stateMachine = new Element("StateMachine", nsUML);
        ownedElement.addContent(stateMachine);
        stateMachine.setAttribute("xmi.id", "machineID001");
        stateMachine.setAttribute("isSpecification", "false");

        Element smCtx = new Element("StateMachine.context", nsUML);
        stateMachine.addContent(smCtx);

        Element classRef = new Element("Class", nsUML);
        classRef.setAttribute("xmi.idref", "#class01");
        smCtx.addContent(classRef);

        Element top = new Element("StateMachine.top", nsUML);
        stateMachine.addContent(top);

        // handle nodes
        writeState(chart.getRootNode(), top);


        // handle Transitions
        Element transitions = new Element("StateMachine.transitions", nsUML);
        stateMachine.addContent(transitions);
        writeTransitions(transitions, chart);
    }


    /**
     * Creator for the XMI exporter. The XMI document is created here.
     */
    public XMIGenerator() {
        output = new Document();
    }


    /**
     * Writes the created XMI Document to the passed file.
     * @param f The file to write the document to.
     * @return <code>true</code> if successful.
     */
    private boolean writeXMIToFile(final File f) {
        XMLOutputter writer = new XMLOutputter();

        FileWriter fw = null;
        Format outputFormat = writer.getFormat();
        outputFormat.setIndent(" ");
        writer.setFormat(outputFormat);
        boolean result = false;
        try {
            fw = new FileWriter(f);
            if (fw != null) {
                writer.output(output, fw);
                result = true;
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    /**
     * This method calls all necessary methods to write the passed statechart
     * to the passed file.
```

```
     * @param chart The statechart to export.
     * @param f The destination file.
     * @return The file.
     */
    public final File generateChart(final StateChart chart,
            final File f) {
        writeXMIHeader();
        writeStateChart(chart);
        writeXMITofile(f);
        return f;
    }
}
```