Christian-Albrechts-Universität zu Kiel

Diplomarbeit

Textuelle Darstellung und strukturbasiertes Editieren von Statecharts

cand. inform. Mirko Wischer

17. Oktober 2007

Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

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

Im Rahmen dieser Arbeit wurde ein Modul für das Projekt *KIEL* entwickelt, das es ermöglicht, die Struktur eines Statecharts zu editieren. Dieses ist nur möglich, da das Projekt *KIEL* über ein Modul verfügt, dass allein aus der Struktur des Statecharts graphische Informationen erzeugen kann. Der Vorgang wird automatisches Layout genannt. Ein Editieren der Struktur ist mittels Maus-, Tastatureingaben und einer Statechart-Beschreibungssprache (*KIT*) möglich. An dieser Arbeit wird deutlich, dass es durch Editieren der Struktur auf schnelle und komfortable Art und Weise möglich ist, Statecharts zu erzeugen und zu verändern.

Schlüsselwörter Statechart, *KIEL*, *KIT*, Strukturediteor, automatisches Layout

In this work a module for the project *KIEL* is developed, that enables the user to edit the structure of a statechart. After editing occurs an automatic layout step to receive a graphical representation of the edited statechart. The statechart structure is editable by mouse, keyboard and a new developed language *KIT*. This work shows that structure editing is a fast and comfortable way creating statecharts.

Keywords Statechart, *KIEL*, *KIT*, structure editing, automatic layout

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Verzeichnis der Abkürzungen

UML	*Unified Modeling Language*
KIEL	*Kiel Integrated Environment for Layout*
SSM	*Safe State Machine*
HUTN	*Human-Usable Textual Notation Specification*
KIT	*KIel statechart extension of doT*
EBNF	Erweiterte Backus-Naur-Form
SCXML	*StateChart XML*
XML	*eXtensible Markup Language*
SVM	*Statechart Virual Machine*
MVC	*Model-View-Controller*

1. Einleitung

Computer halten immer mehr Einzug in unser tägliches Leben. Dabei stellen die klassischen Computer nur einen kleinen Anteil am Gesamtaufkommen dar, der weitaus größere Teil sind so genannte eingebettete Systeme. Solche Systeme werden in der Regel von Mikrocontrollern gesteuert. Mikrocontroller sind Prozessoren, die häufig zusätzliche Speicher, digitale Ein- und Ausgänge sowie Digital-Analog Wandler enthalten. Diese befinden sich in vielen Geräten, die man auf den ersten Blick nicht als Computer erkennen kann, wie zum Beispiel in Waschmaschinen, Chipkarten, Unterhaltungselektronik, Mobiltelefonen oder auch Armbanduhren.

In Analogie zum klassischen Computer sind auch hier Programme nötig, um die Geräte zu steuern oder auf Eingaben des Benutzers zu reagieren. Durch die Vielzahl der eingebetteten Systeme wird auch die Programmierung solcher Systeme immer wichtiger. Programme werden meistens in Form von Programmiersprachen verfasst, d.h. sie entstehen in einer textuellen Form. Eine textuelle Form ist im Gegensatz zum Gehirn eindimensional und lässt visuelle Fähigkeiten des Gehirns ungenutzt. Die Analyse des gelesenen Textes ist Voraussetzung für das Verständnis der Funktionalität. Diagramme können, um komplexe Zusammenhänge und Abläufe vereinfacht darzustellen, eine Hilfestellung sein. Eine der dominierenden Sprachen für die Modellierung von Softwaresystemen ist die Unified-Modeling-Lanuguage(UML) [14] . Das bekannteste Diagramm der UML zur Beschreibung eines statischen Modells ist das Klassendiagramm, das die Klassen des Systems mit Attributen, Operationen, den Beziehungen zwischen den Klassen sowie den Vererbungsstrukturen beschreibt. Die ersten Fassungen der UML, auch UML 1.x genannt, wurden 2005 durch eine überarbeitete Version der UML 2.0 abgelöst. Die neue Version unterscheidet 13 verschiedene Diagrammarten, eine davon sind die Statecharts (deutsch: Zustandsdiagramme). Alle für diese Arbeit erstellten Statechart-Abbildungen sind mit Hilfe von *KIEL* [23] generiert worden. *KIEL* (**K**iel **I**ntegrated **E**nvironment for **L**ayout) ist ein experimentelles Modellierungswerkzeug, das am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme entwickelt wird.

Die Modellierungswerkzeuge werden zum Erstellen von Statecharts genutzt und arbeiten auf einer graphischen Ebene. Die Statecharts werden ähnlich wie in einem Zeichenprogramm gezeichnet, diese Vorgehensweise wird auch syntaxgerichtetes Editieren genannt. Genau hier liegt das Problem: Hat man ein Statechart erstellt und darauf geachtet, dass es kompakt und gut lesbar ist, muss es eventuell erweitert werden. In einer textuellen Darstellung ist das kein Problem, da es durch die Eindimensionalität der Darstellung möglich ist, an irgendeiner Stelle im Text etwas einzufügen, ohne dass der Text vom Anwender verschoben werden muss. In einer graphischen Darstellung muss zunächst Platz geschaffen werden, um die Änderun-

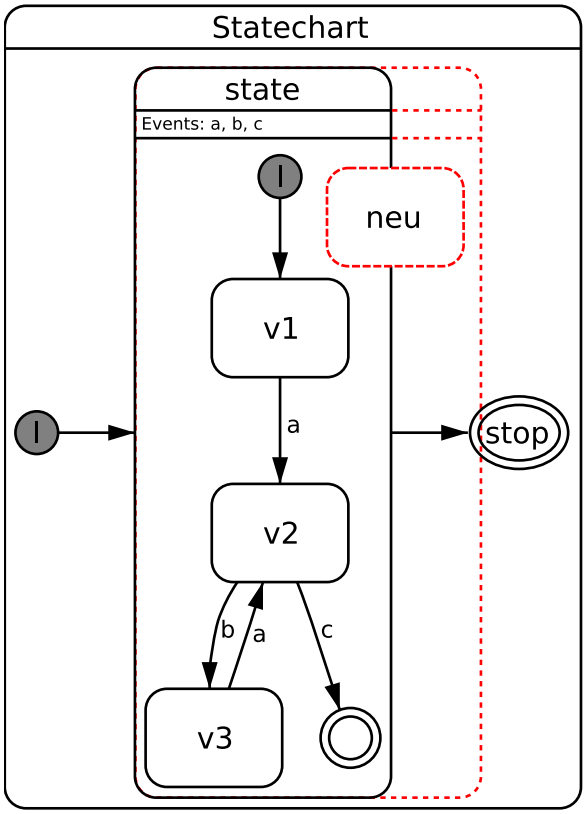


Abbildung 1.1.: Platzproblem beim Hinzufügen eines Zustandes

gen einfügen zu können, siehe Abbildung 1.1. Dieses Beispiel zeigt, dass es einen Optimierungsbedarf für das Editieren von Statecharts gibt. Daher widmet sich diese Arbeit der Optimierung des Editierprozesses. Als Grundlage wird dabei das Modellierungswerkzeug des Projektes *KIEL* genutzt. Der Aufbau der Arbeit stellt sich folgendermaßen dar:

- Im folgenden Kapitel werden zunächst die Grundlagen dieser Arbeit beschrieben. Dazu wird ein kurzer Überblick über das Gebiet der Statecharts gegeben.
- Im Kapitel 3 wird der Modellierungsprozess von Statecharts analysiert.
- Das Kapitel 4 vergleicht die einzelnen implementierten Verfahren miteinander, anhand eines realistischen Modells werden Vor- und Nachteile aufgezeigt.
- Aufbauend auf die vorherigen Kapitel erfolgt im Kapitel 5 die Umsetzung dieser Arbeit.
- Abschließend erfolgt eine kurze Zusammenfassung der Arbeit. Mögliche Erweiterungen runden die vorgestellte Arbeit ab.

2. Statecharts

In diesem Kapitel werden die Grundlagen der Arbeit erläutert. Es wird ein allgemeiner Überblick über das Gebiet der Statecharts gegeben. Dazu wird zum Einen auf die Bedeutung und zum Anderen auf den Aufbau der Statecharts eingegangen.

Bedeutung Statecharts gehören zu der Gruppe der visuellen Sprachen. Schiffer [21] definiert den Begriff wie folgt:

> „Eine visuelle Sprache ist eine formale Sprache mit visueller Syntax oder visueller Semantik und dynamischer oder statischer Zeichengebung."

Eine eng mit den Statecharts verwandte visuelle Sprache sind die Endlichen Automaten. Von ihnen wurde erstmals ein System auf Basis von Zuständen beschrieben. Ein Zustandsübergang, auch Transition genannt, beschreibt eine Änderung des aktuellen Zustandes und wird durch logische Bedingungen spezifiziert. Diese müssen erfüllt sein, um einen Zustandsübergang zu ermöglichen.

Ein einfaches System, das sich leicht mit Hilfe von Statecharts beschreiben lässt, ist eine Schreibtischlampe. Dieses System kennt zwei Zustände „Licht an" und „Licht aus", durch betätigen des Schalters wird der aktuelle Zustand gewechselt. Dieses System wird in Abbildung 2.1 dargestellt.

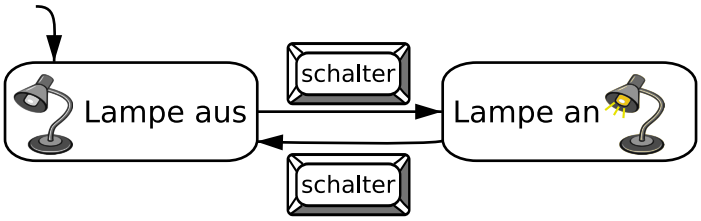


Abbildung 2.1.: Modell einer Schreibtischlampe

Statecharts erweitern Endliche Automaten um Hierarchie und Parallelität. Der Begriff des Statecharts wird erstmals von David Harel in einem 1987 veröffentlichten Papier [11] vorgestellt. Mittlerweile existieren neben dem ursprünglich von Harel vorgeschlagenen Statechart-Dialekt auch andere. Die verschiedenen Statechart-Dialekte unterscheiden sich in Semantik und Zeichengebung, dabei bringen die meisten Modellierungswerkzeuge ihren eigenen Dialekt mit. Auch die UML [14] definiert einen Statechart-Dialekt. Die in dieser Arbeit verwendeten Statecharts entsprechen in der Zeichengebung den *Safe State Machines* [2](SSM), die in *Esterel Studio* [5] verwendet werden.

Aufbau Ein Statechart ist im Allgemeinen aus zwei Arten von Komponenten aufgebaut: Aus Zuständen und aus Transitionen. Es gibt jedoch verschiedene Typen von Zuständen und unterschiedliche Typen von Transitionen. Hinter jedem Typ verbirgt sich eine andere Funktionalität. Transitionen besitzen zwei Attribute, eine Transitionsbeschriftung und eine Priorität.Sie werden in den Abbildungen durch beschriftete Pfeile dargestellt. Bei den Zuständen wird zwischen zwei Haupttypen unterschieden: den „normalen“ Zuständen und den Pseudo-Zuständen. Die Pseudo-Zustände sind dadurch charakterisiert, dass sie in einem Statechart nie der aktuelle Zustand sein können. Zusätzlich unterscheiden sie sich in der Zeichengebung von den anderen Zuständen, sie werden in der Regel als Kreise dargestellt. Folgende Pseudo-Zustände sind für diese Arbeit relevant:

Initialer Zustand: Der initiale Zustand ist der Zustand bei dem das Statechart beginnt. Er wird daher auch Startzustand genannt. Aus diesem Grunde hat der initiale Zustand keine eingehenden Transitionen. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.2 dargestellt.

Abbildung 2.2.: initialer Zustand

History-Zustand: Ein History-Zustand ermöglicht es, bei einem anderen Zustand als den Startzustand zu beginnen. Eine weitere Art des History-Zustandes ist der Deep-History-Zustand. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.3 zu sehen.

Abbildung 2.3.: Deep-History-Zustand

Suspend-Zustand: Ein Suspend-Zustand ist in der Lage, den momentanen Ablauf in einem anderen Zustand zu unterbrechen. Er bezieht sich daher immer auf einen anderen Zustand und besitzt daher nur ausgehende Transitionen. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.4 dargestellt.

Abbildung 2.4.: Suspend-Zustand

Choice-Zustand: Ein Choice-Zustand ermöglicht es, den Ablauf des Statechart anhand von Bedingungen zu verzweigen. Ein weiterer Typ von Choice-Zuständen

ist der Dynamic-Choice-Zustand. In anderen Statechart-Dialekten wird der Choice-Zustand *Junction* genannt. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.5 zu sehen.

Abbildung 2.5.: Choice-Zustand

Die „normalen“ Zustände unterscheiden ebenfalls eine Reihe von verschiedenen Typen. Allen gemeinsam ist, dass sie Namen haben können.

Einfacher Zustand: Ein einfacher Zustand wird auch Simple-State genannt. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.6 zu sehen.

Abbildung 2.6.: einfacher Zustand

Hierarchischer Zustand: Ein hierarchischer Zustand wird auch OR-Zustand oder *OR-State* genannt. Er versetzt den Modellierenden in die Lage Hierarchie in das Statechart einzubauen, da er alle Arten von Zuständen enthalten kann. Ein solcher Zustand enthält jeweils genau einen aktiven Zustand. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.7 dargestellt.

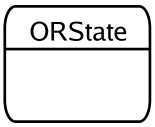


Abbildung 2.7.: hierarchischer Zustand

Paralleler Zustand: Ein paralleler Zustand wird auch AND-Zustand oder *AND-State* genannt. Er enthält beliebig viele parallele Regionen, die in der Zeichengebung dieser Arbeit durch gestrichelte Trennlinien dargestellt werden. Die Trennlinien werden auch *Delimiter-Lines* genannt. Jede dieser Regionen enthält einen aktiven Zustand. Die in dieser Arbeit verwendete Zeichengebung der SSM ist in Abbildung 2.8 zu sehen.

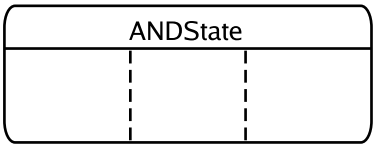


Abbildung 2.8.: paralleler Zustand

Abbildung 2.9.: einfacher finaler Zustand

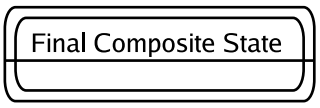


Abbildung 2.10.: hierarchischer finaler Zustand

Jeder der oben beschriebenen Zustände kann ein finaler Zustand sein, der das Gegenstück zum initialen Zustand ist. Ein solcher Zustand besitzt nur eingehende Transitionen. In diesem Fall ändert sich auch die Zeichengebung. Ein einfacher finaler Zustand ist in Abbildung 2.9 und ein hierarchischer finaler Zustand in Abbildung 2.10 zu sehen. Ein paralleler finaler Zustand wird analog zum hierarchischen finalen Zustand gezeichnet.

Ein besonderer hierarchischer bzw. paralleler Zustand ist der so genannte Root-Zustand, der alle anderen Zustände des Statecharts enthält.

Ein häufig verwendeter Begriff in den folgenden Kapiteln ist die Topologie, auch Struktur des Statecharts genannt. Darunter wird die reine Komponentensicht auf ein Statechart und die Beziehung der Komponenten untereinander verstanden.

Nachdem alle benötigten Grundlagen erläutert wurden, wird im nächsten Kapitel der Modellierungsprozess von Statecharts analysiert und Vorschläge zur Optimierung vorgestellt.

3. Analyse der Modellierungsprozesse von Statecharts

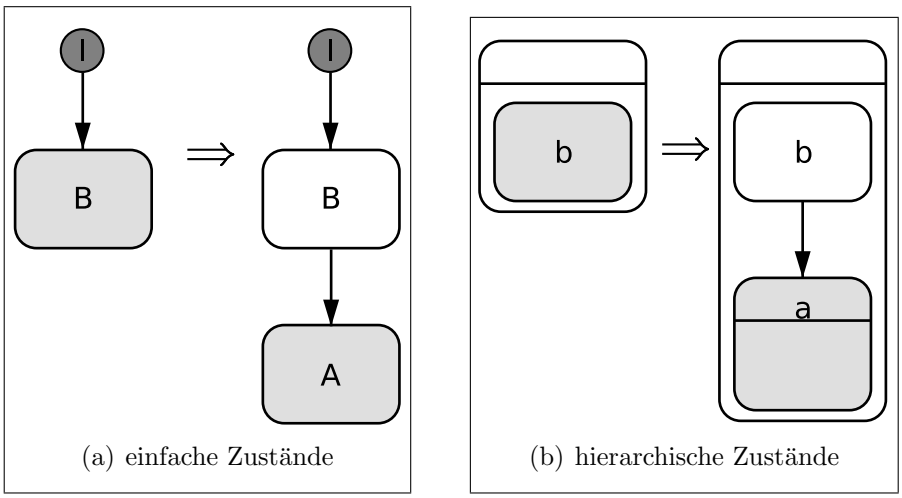
Im vorherigen Kapitel wurden die Grundlagen zum Verständnis dieser Arbeit vermittelt, in diesem Abschnitt folgt die genaue Analyse des Modellierungsprozesses von Statecharts. Es findet eine Bestandsaufnahme vorhandener Modellierungsmöglichkeiten statt und darauf aufbauend werden Optimierungsmöglichkeiten aufgezeigt.

Der Modellierungsprozess mit nahezu allen am Markt erhältlichen Modellierungswerkzeugen findet mit Statechart-Editoren statt. Beispiele solcher Modellierungswerkzeuge sind:

- *Matlab/Simulink/Stateflow* [24]
- *Esterel-Studio* [5]
- *Rational Rose* [18]
- *ArgoUML* [3]

Die vorhandenen Statechart-Editoren sind syntaxgerichtete Editoren, sie funktionieren nach dem Prinzip *WYSIWYG* (engl.: ***w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et*). Das bedeutet, dass die graphischen Objekte eines Statecharts direkt verändert werden können, d.h. jeder Zustand ist in seiner Größe und Position modifizierbar. Desweiteren wird der Verlauf einer Transition durch das Setzen von Stützpunkten manuell bestimmt. Einzig *ArgoUML* unterstützt den Entwickler beim Modellierungsprozess, mit Hilfe von intelligenten Bearbeitungswerkzeugen [19], wie einem Werkzeug zum automatischen Platz-Schaffen, um neue Elemente hinzufügen zu können.

Um den Prozess des Modellierens zu analysieren, werden zunächst verschiedene Schemata, die mögliche Änderungen an einem Statechart beschreiben, aufgezeigt. Aus diesen Schemata wird ein allgemeiner Ablauf in konventionellen Statechart-Editoren beim Durchführen von Änderungen an einem Statechart abgeleitet. Dieser Ablauf wird auf Optimierungsmöglichkeiten untersucht. Abschließend findet die Analyse der vorgestellten Schemata anhand der dargelegten Optimierungsmöglichkeiten statt. Im Folgenden werden mögliche Änderungen mit Hilfe verschiedener Schemata beschrieben.



(a) einfache Zustände (b) hierarchische Zustände

Abbildung 3.1.: Schemata zum Hinzufügen von Zuständen

3.1. Vorgehensweisen beim Editieren von Statecharts

In diesem Abschnitt sollen verschiedene Vorgehensweisen beim Editieren von Statecharts vorgestellt werden. Die Änderungsumsetzung findet für die konventionelle Herangehensweise im Abschnitt 3.2.1 und für die optimierte Herangehensweise im Abschnitt 3.3 statt.

Hinzufügen von Zuständen: Eine häufige Änderung ist das Hinzufügen von Zuständen. Das beinhaltet nicht nur das eigentliche Hinzufügen des Zustandes, sondern auch das Hinzufügen mindestens einer Transition, die auf den neuen Zustand zeigt. Ein Schema dieser Änderung befindet sich für einfache Zustände in Abbildung 3.1(a) und für hierarchische Zustände in Abbildung 3.1(b).

Hinzufügen von parallelen Regionen: Eine weitere Änderungsmöglichkeit ist das Hinzufügen von weiteren parallelen Regionen zu einem parallelen Zustand. Dabei ist es nötig, zu dem neu erzeugten parallelen Zweig einen initialen und einen einfachen Zustand hinzuzufügen, um eine Grundfunktionalität zu gewährleisten. Ein Schema der Änderung ist in Abbildung 3.2 zu finden.

Umwandeln von Zuständen: Erkennt man, dass der Typ eines Zustandes falsch gewählt wurde, muss eine Änderung des Zustandstyps stattfinden. Aus diesem Grund kommt es vor, dass ein einfacher Zustand zu einem hierarchischen Zustand umgewandelt werden muss. Dabei ist es nötig, zum hierarchischen Zustand einen initialen und einen einfachen Zustand hinzuzufügen, da diese für eine Grundfunktionalität des hierarchischen Zustandes erforderlich sind. Genauso kann ein hierarchischer Zustand zu einem parallelen Zustand umgewandelt werden. Dabei wird das Hinzufügen einer parallelen Region, der ein initialer und ein einfacher Zustand hinzugefügt wird, notwendig. Die entsprechenden Schemata sind in Abbildung 3.3 dargestellt.

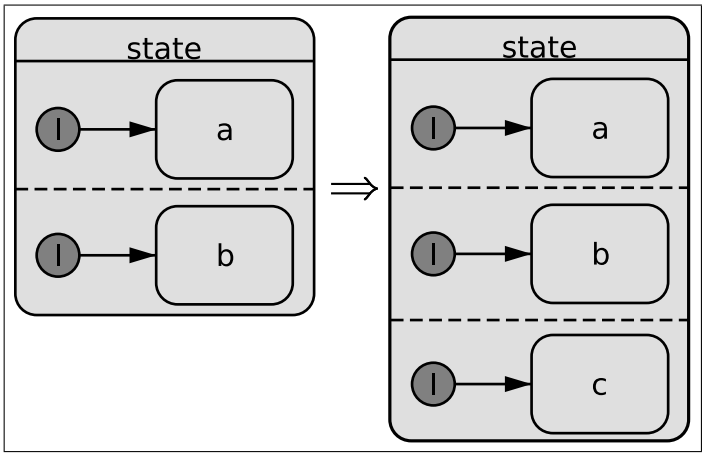
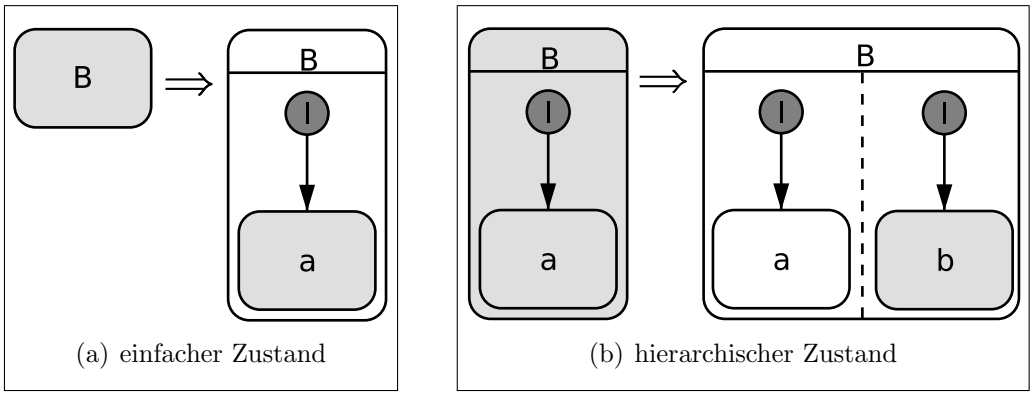


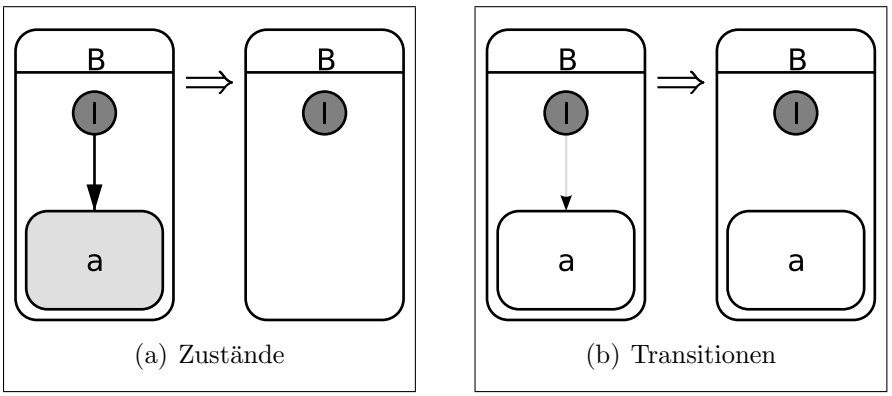
Abbildung 3.2.: Schema zum Hinzufügen einer Region



(a) einfacher Zustand

(b) hierarchischer Zustand

Abbildung 3.3.: Schema zum Umwandeln eines Zustandes



(a) Zustände (b) Transitionen

Abbildung 3.4.: Schema zum Entfernen von Komponenten

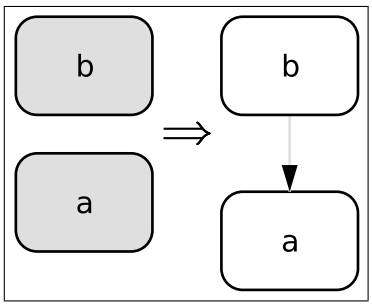


Abbildung 3.5.: Schema zum Erzeugen einer Transition

Entfernen von Zuständen: Desweiteren ist es häufig nötig, Komponenten zu entfernen. Dabei kann es sich um Transitionen oder um Zustände handeln. Bei Zuständen ist darauf zu achten, dass die von dem Zustand ausgehenden und eingehenden Transitionen ebenfalls entfernt werden müssen. Die zugehörigen Schemata sind in Abbildung 3.4 veranschaulicht.

Erzeugen von Transitionen: Das Verbinden zweier Zustände mit einer Transition wird in Abbildung 3.5 zur Darstellung gebracht.

Vertauschen von Quelle und Senke: Eine zusätzliche Änderung ist das Vertauschen von Quelle und Senke einer Transition. Verläuft eine Transition ursprünglich von „a“ nach „b“ soll die gleiche Transition nach der Änderung von „b“ nach „a“ verlaufen. Dieser Vorgang ist in Abbildung 3.6 zu sehen.

Hinzufügen von History-Zuständen: Das Hinzufügen von History-Zuständen stellt eine weitere Änderung dar. An welcher Stelle diese Pseudo-Zustände platziert werden, hängt von der Wahl des Modellierungswerkzeuges ab. In *Esterel-Studio* werden solche Zustände beispielsweise an der Außenseite des Zustandes, auf den sie sich beziehen, platziert. In Abbildung 3.7 ist ein Schema zum Hinzufügen von History-Zuständen abgebildet. Die Platzierung erfolgt wie im Modellierungswerkzeug des Projektes *KIEL*. Das Schema lässt sich analog auf

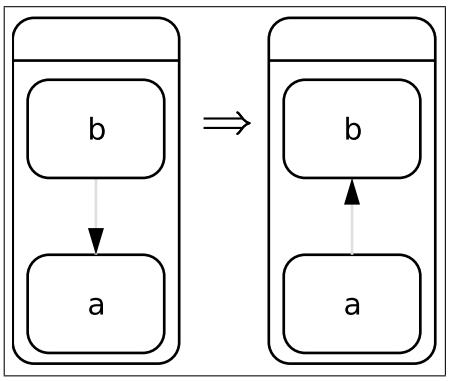


Abbildung 3.6.: Schema zum Vertauschen von Quelle und Senke

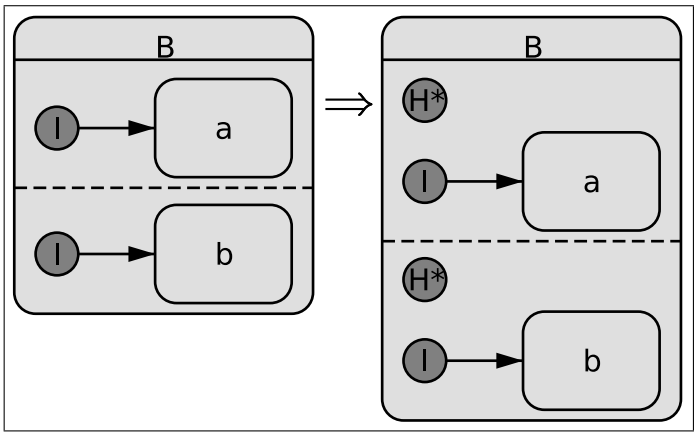


Abbildung 3.7.: Schema zum Hinzufügen von History-Zuständen

Deep-History-Zustände anwenden.

Hinzufügen von Suspend-Zuständen: Das Hinzufügen von Suspend-Zuständen, eine Besonderheit von *Esterel Studio*, besteht nicht nur aus dem eigentlichen Hinzufügen des Suspend-Zustandes, sondern auch aus dem Verbinden des Suspend-Zustandes mit dem zu suspendierenden Zustand. Die Vorgehensweise ist in Abbildung 3.8 veranschaulicht.

Hinzufügen von Choice-Konstrukten: Die letzte vorgestellte Änderung ist das Hinzufügen von Choice-Konstrukten. Dabei werden zwei Zustände mit einem Choice-Konstrukt, das aus einem Choice-Zustand und zwei Transitionen besteht, verbunden. Das Schema ist in Abbildung 3.9 zu sehen. Dieses Schema lässt sich analog auf Dynamic-Choice-Zustände und auf Junction-Zustände anwenden.

Im folgenden Abschnitt wird untersucht, wie sich die vorgestellten Schemata in konventionellen Statechart-Editoren umsetzten lassen.

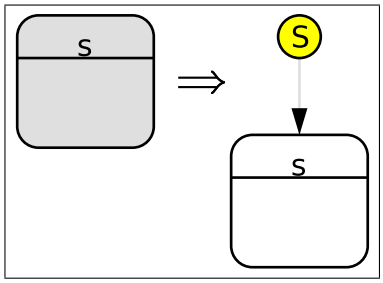


Abbildung 3.8.: Schema zum Hinzufügen von Suspend-Zuständen

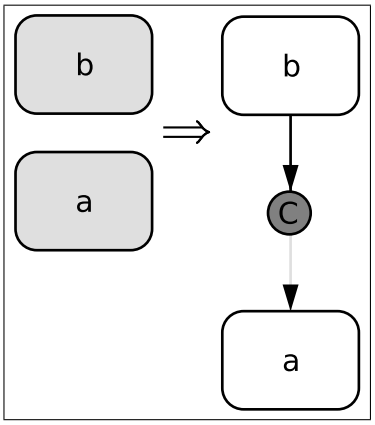


Abbildung 3.9.: Schema zum Hinzufügen von Choice-Konstrukten

3.2. Syntaxgerichtete Statechart-Editoren

3.2.1. Umsetzung der Editier-Schemata

In diesem Abschnitt wird die Änderung eines Statecharts mit klassischen syntaxgerichteten Statechart-Editoren erläutert. Dazu wird die Umsetzung der zuvor vorgestellten Schemata erklärt. Die Beschreibung bezieht sich auf *Esterel Studio* [5], lässt sich aber leicht auf andere Modellierungswerkzeuge übertragen.

Hinzufügen von Zuständen: Um die Schemata aus Abbildung 3.1 umsetzen zu können, sind folgende Schritte notwendig:

1. Platz-Schaffen für einen neuen Zustand
2. auswählen des Zustandes
3. platzieren des Zustandes
4. verbinden der beiden Zustände mit einer Transition

Hinzufügen von parallelen Regionen: Das in Abbildung 3.2 dargestellte Schema kann folgendermaßen umgesetzt werden:

1. vergrößern des Zustandes, dem eine parallele Region hinzugefügt werden soll
2. auswählen einer Trennlinie aus dem Menü
3. positionieren der Trennlinie
4. auswählen des initialen Zustandes
5. positionieren des initialen Zustandes
6. auswählen des einfachen Zustandes
7. positionieren des einfachen Zustandes
8. verbinden des initialen mit dem einfachen Zustand

Umwandeln von Zuständen: Für die Umsetzung der Schemata in Abbildung 3.3 ist eine Fallunterscheidung nötig. Soll ein einfacher Zustand umgewandelt werden, sind folgende Schritte notwendig:

1. umwandeln des einfachen Zustandes in ein hierarchischen Zustand
2. Platz-Schaffen für das Vergrößern des Zustandes
3. vergrößern des Zustandes, um weitere Zustände hinzufügen zu können
4. auswählen des initialen Zustandes
5. positionieren des initialen Zustandes
6. auswählen des einfachen Zustandes
7. positionieren des einfachen Zustandes

8. verbinden des initialen mit dem einfachen Zustand

Soll ein hierarchischer Zustand umgewandelt werden, sind die gleichen Schritte wie beim Hinzufügen von parallelen Regionen notwendig.

Entfernen von Zuständen: Die Umsetzung der Schemata in Abbildung 3.4 sieht folgendermaßen aus:

1. auswählen der zu löschenden Komponenten
2. löschen der Komponenten

Erzeugen von Transitionen: Um eine Transition wie in Abbildung 3.5 zu erzeugen, muss wie folgt vorgegangen werden:

1. Platz-Schaffen für die Transition
2. ziehen der Transition
3. setzen von zusätzlichen Stützpunkten, um den Transitionsverlauf zu verbessern

Vertauschen von Quelle und Senke: Um die Quelle und Senke einer Transition zu Vertauschen wie in Abbildung 3.6 muss zunächst die Transition entfernt und danach mit den gleichen Eigenschaften, jedoch mit vertauschter Quelle und Senke neu erzeugt werden.

Hinzufügen von History-Zuständen: Um History-Zustände wie in Abbildung 3.7 hinzufügen zu können, muss wie folgt vorgegangen werden:

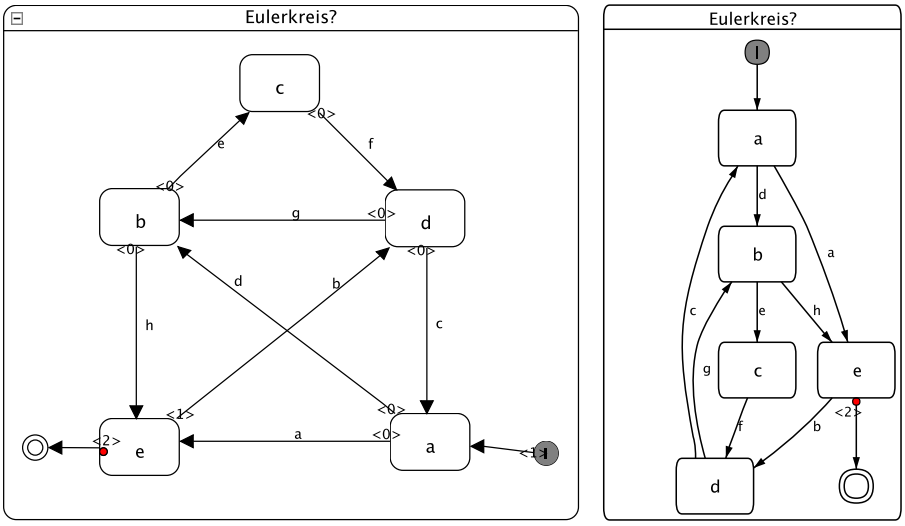
1. Platz-Schaffen für die neuen History-Zustände
2. auswählen des History-Zustandes
3. positionieren der History-Zustände

Hinzufügen von Suspend-Zuständen: Für Suspend-Zustände, wie sie das Schema in Abbildung 3.8 darstellt, muss folgendermaßen vorgegangen werden:

1. Platz-Schaffen für den neuen Suspend-Zustand
2. auswählen des Suspend-Zustandes
3. positionieren des Suspend-Zustandes
4. ziehen einer Transition von dem Suspend-Zustand

Hinzufügen von Choice-Konstrukten: Um Choice-Konstrukte wie in Abbildung 3.9 umsetzen zu können, sind folgende Schritte nötig:

1. Platz-Schaffen
2. auswählen des Choice-Zustandes
3. positionieren des Choice-Zustandes
4. ziehen einer Transition zum Choice-Zustand
5. ziehen einer Transition vom Choice-Zustand



(a) freies Layout

(b) Statechart Normalform [**?**]

Abbildung 3.10.: flexible Darstellung

3.2.2. Vor- und Nachteile

Nachdem die Umsetzung aller Schemata betrachtet worden ist, werden die Stärken und Schwächen des klassischen Ansatzes beleuchtet.

Eine Stärke des syntaxgerichteten Editierens stellt die Flexibilität dar. Diese bezieht sich vorrangig auf das Gestalten der Statecharts. Es ist ohne Probleme möglich, ein Statechart vorstellungsgemäß aussehen zu lassen, siehe dazu Abbildung 3.10. Ein weiterer Vorteil ist, dass ungeübte Benutzer einen schnellen Zugang zum Erstellen von Statecharts finden, da diese wie in einem Zeichenprogramm erstellt werden.

Beim syntaxgerichteten Editieren ist der gravierendste Nachteil, dass sich, bis auf das Entfernen von Komponenten, keine der vorgestellten Schemata direkt umsetzen läßt. Es sind immer mehrere Schritte notwendig, bis das komplette Editier-Schema erstellt werden kann. Ein weiterer Nachteil ist das häufig notwendige Platz- Schaffen, das zum Teil zu einer Umstrukturierung des ganzen Statecharts führt. Wird das gesamte Statechart auf einmal erstellt, ist der Arbeitsaufwand für die Verbesserung des Layouts gering. Möchte man jedoch ein vorhandenes Statechart erweitern, nehmen die Layoutarbeiten mit der Größe und Komplexität des Statecharts zu. Die soeben vorgestellten Nachteile wurden zum Anlass genommen, den Modellierungsprozess zu optimieren. Dazu werden im folgenden Abschnitt Vorschläge unterbreitet.

3.3. Vorschläge zur Optimierung

In diesem Abschnitt werden Vorschläge zur Optimierung des Modellierungsprozesses von Statecharts beschrieben. Die Analyse der Vorgänge in klassischen syntax-

gerichteten Statechart-Editoren hat gezeigt, dass hier Optimierungsbedarf besteht. Insbesondere das Erweitern von vorhandenen komplexen Statecharts erfordert viele Schritte, die es zu optimieren gilt. Zunächst wird dazu ein allgemeiner Ablauf zur Umsetzung der Schemata in konventionellen Statechart-Editoren vorgestellt. Anhand des Ablaufs werden Schritte aufgezeigt, die eine Optimierung zulassen.

Im Allgemeinen sind folgende Schritte nötig, um ein Schema aus Abschnitt 3.1 zu realisieren:

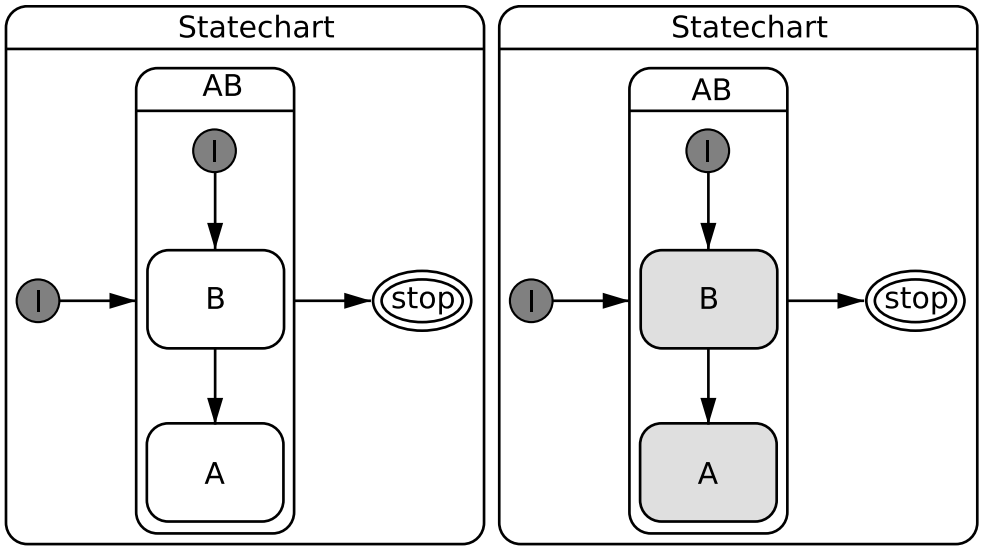
1. Komponente(n) suchen
2. Layoutkorrekturen:
   a) Größenänderung(en) durchführen
   b) Positionsänderung(en) durchführen
3. Änderung(en) durchführen
4. Layout verbessern

Als erstes erfolgt die Bestimmung der an dieser Änderung beteiligten Statechartkomponenten (1.). Danach wird abgewogen, ob Layoutkorrekturen nötig sind, um die Änderung durchzuführen (2.).Hat man die ersten drei Schritte abgeschlossen, können bestimmte Änderungen vorgenommen werden (3.). Abschließend wird das neu erstellte Layout nochmals überprüft und eventuell verbessert (4.).

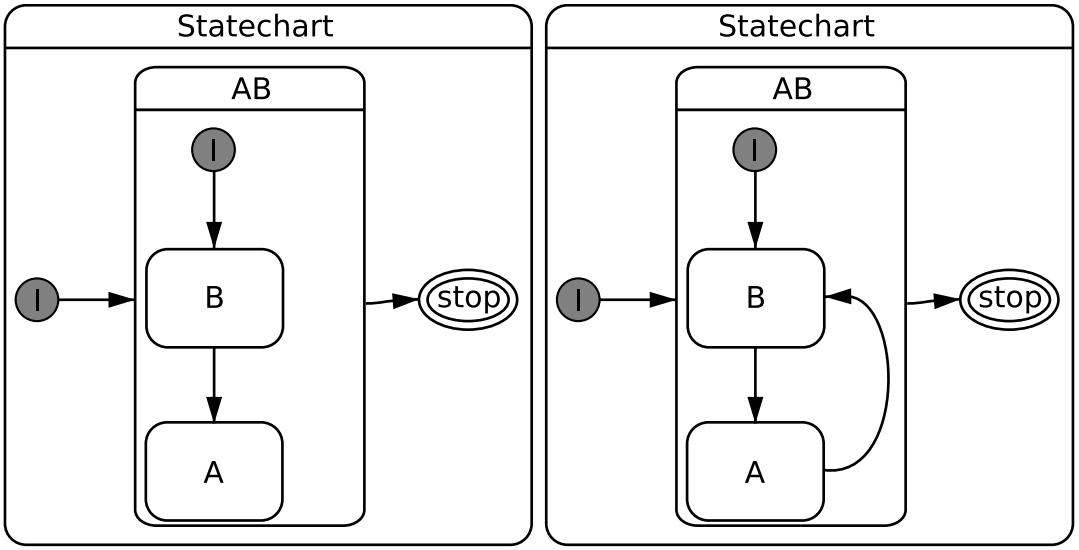
Zur Verdeutlichung des Ablaufs folgt Beispiel 3.11, in dem die Zustände „A“ und „B“ mit einer Transition verbunden werden. Das zu verändernde Statechart ist in Abbildung 3.11(a) aufgeführt.

1. suchen der Komponenten „A“ und „B“, siehe Abbildung 3.11(b)
2. Layoutkorrekturen:
   a) Vergrößerung des Zustandes „AB“, Abbildung 3.11(c)
3. zeichnen der neuen Transition, Abbildung 3.11(d)
4. verbessern des Layouts: Der Verlauf der Transitionen wird besser gestaltet und die Größe des Zustandes „AB“ wird etwas verkleinert, Abbildung 3.11(e)

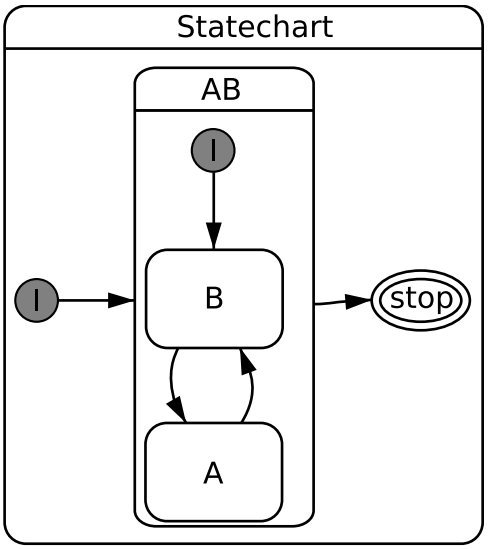
Die komplette Umsetzung eines Schemas erfolgt nach Durchlauf aller vier Schritte. Einige von diesen Schritten sind graphische Schritte: z. B. das Platz-Schaffen, den Vorlauf von Transitionen bestimmen und neu arrangieren. Um den Aufwand für die Modellierung zu verringern, gilt es, den vorgestellten Ablauf zu optimieren. Dazu ist es möglich, die gestaltenden Schritte durch entsprechende Automatismen zu ersetzen. Es ist möglich, die Schritte durch Verfahren zu ersetzen, die ein automatisches Layout auf dem Statechart durchführen [4]. Solche Verfahren sind bekannt und werden z. B. in der Arbeit von Tobias Kloss [**?**] beschrieben. Damit verkürzt sich der Arbeitsablauf auf folgende Schritte:



(a) ursprüngliches Statechart (b) Komponenten Suchen

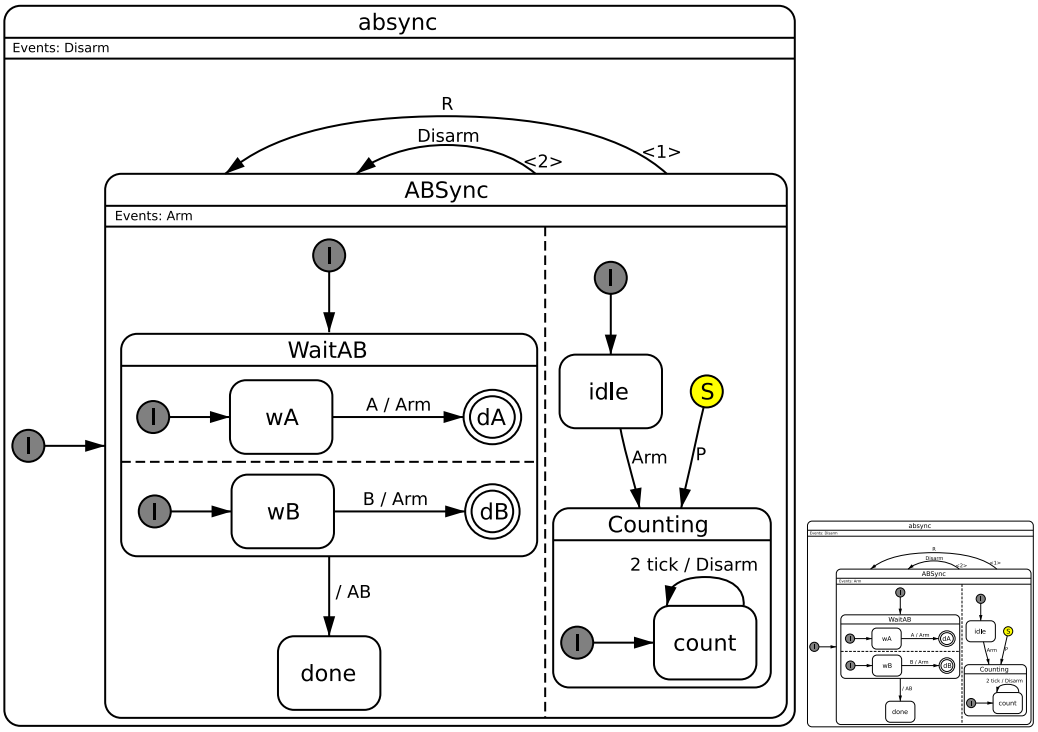


(c) Layout Korrekturen (d) hinzufügen von Komponenten



(e) Layout verbessern

Abbildung 3.11.: Beispiel eines konventionellen Editiervorganges



(a) normale Ansicht

(b) höchste Zoomstufe

Abbildung 3.12.: Lesbarkeit bei hohen Zoom-Stufen

1. Komponente(n) suchen
2. Schema anwenden

Diese beiden Schritten werden in den nächsten Abschnitten genauer betrachtet.

3.3.1. Komponenten suchen

Hat der Entwickler einen Änderungswunsch, muss er zunächst die beteiligten Komponenten, wie oben beschrieben, lokalisieren und eventuell selektieren. In konventionellen syntaxgerichteten Editoren geschieht das in der Regel durch das Suchen und das Klicken mit der Maus auf die Komponente in der Zeichenfläche, um diese auszuwählen. Sind die Statecharts größer als die sichtbare Zeichenfläche, müssen die Statecharts auf der Zeichenfläche so lange verschoben werden, bis die Komponenten gefunden werden. Hier kann eine Zoom-Funktion hilfreich sein, um das Verschieben möglichst gering zu halten. Sind jedoch die Statecharts besonders groß, ist nach dem Zoom-Vorgang eine Suche nach Komponenten unmöglich, da die einzelnen Elemente zu stark verkleinert werden. Hier muss jeweils abgewogen werden, ob alles sichtbar aber entsprechend klein oder alles gut lesbar jedoch nicht mehr auf die Zeichenfläche passend dargestellt werden soll, Abbildung 3.12.

Die klassische Vorgehensweise, das Klicken mit der Maus auf die gesuchte Komponente, ist gerade bei kleinen Statecharts die Methode der Wahl. Solange die Komponenten auf einen Blick gefunden werden können, besteht kein Optimierungsbedarf. Für komplexe Statecharts sind jedoch zusätzliche Suchoptionen, die die Komplexität reduzieren, sinnvoll. Dabei scheint es von Vorteil zu sein, die Sichtweise auf das Statechart zu verändern. Liegt die Betrachtung auf der Topologie des Statecharts und die Suche der Komponenten findet in der Topologie statt, sind folgende Ansätze denkbar:

1. Suchen von Komponenten findet über eine Baumansicht, deren Zweige jeweils ausblendbar sind, statt. Durch das Einklappen der verschiedenen Hierarchieebenen reduziert sich die Komplexität und ein schnelleres Finden gesuchter Komponenten ist möglich.

2. Suchen von Komponenten geschieht mit Hilfe des Zustandnamens oder der Transitionsbeschriftung. Nur Namen oder die Beschriftung der Komponenten müssen bekannt sein, um diese zu finden. Angaben über die Lage der Komponenten im Statechart werden dabei nicht nötig.

3. Statecharttopologie kann mittels Tastenkombinationen durchwandert werden.

Das Navigieren über die Tastatur im Statechart ist mit der Umsetzung folgender Konzepte möglich. Die Symbole entsprechen dabei den abgebildeten Tasten.

[↑] , [↓] Mit diesen Tasten könnten Auswahlmöglichkeiten selektiert werden, beispielsweise mehrere ausgehende Transitionen aus einem Zustand oder mehrere Regionen in einem AND-State, siehe Abbildung 3.13. Die Reihenfolge der Komponenten könnte mit Hilfe eines Sortierverfahrens bestimmt werden. Zum Beispiel könnten Transitionen nach ihrer Priorität geordnet werden.

[←] , [→] Mit diesen Tasten könnten Sequenzen durchlaufen werden, jeweils beginnend mit einem initialen Zustand,siehe Abbildung 3.14.

[Page ↑] , [Page ↓] Mit diesen Tasten ist es möglich, die Hierarchiestufen herauf bzw. herab zu gehen. Wobei das Hinabsteigen einer Hierarchiestufe nur auf AND-States oder OR-States funktioniert, siehe Abbildung 3.15.

[Space] Mit dieser Taste könnte ein selektierter Zustand gespeichert und ein zweiter Zustand ausgewählt werden. Zwei bereits ausgewählte Zustände könnten durch diese Taste wieder freigegeben werden, siehe Abbildung 3.16.

[Home] Mit dieser Taste könnte zum Beginn der Sequenz zurück gesprungen werden, d.h. zum initialen Zustand, siehe Abbildung 3.17.

[Ctrl] + [↑] , [Ctrl] + [↓] Mit diesen Tasten könnte eine Historie der bereits durchlaufenen Schritte nochmals durchlaufen werden. Es ist möglich, mit [Ctrl] + [↓] Schritte bis zum Anfang der Navigation zurück zu gehen. Mit [Ctrl] +

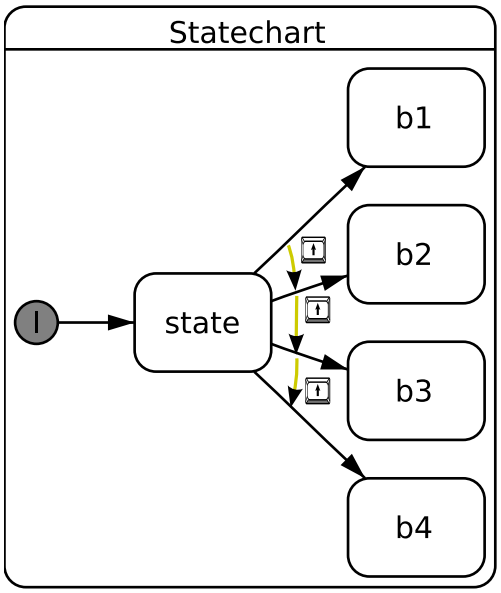


Abbildung 3.13.: Auswahlmöglichkeiten

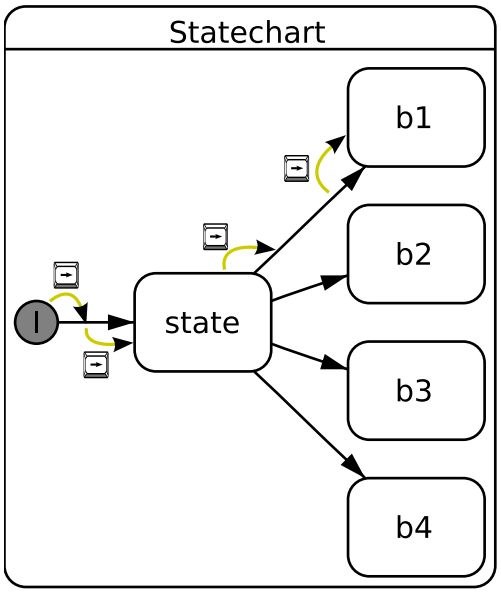


Abbildung 3.14.: durchlaufen der Sequenzen

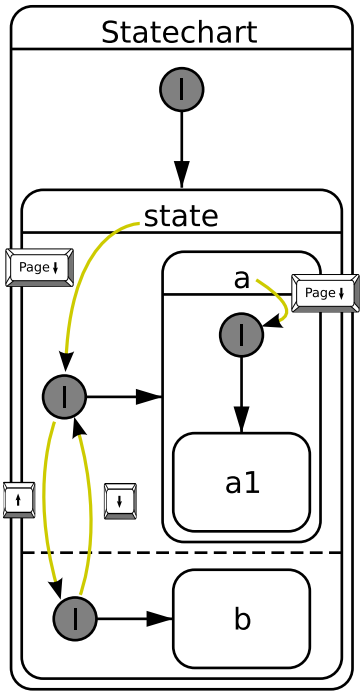


Abbildung 3.15.: Hierarchiestufen wechseln

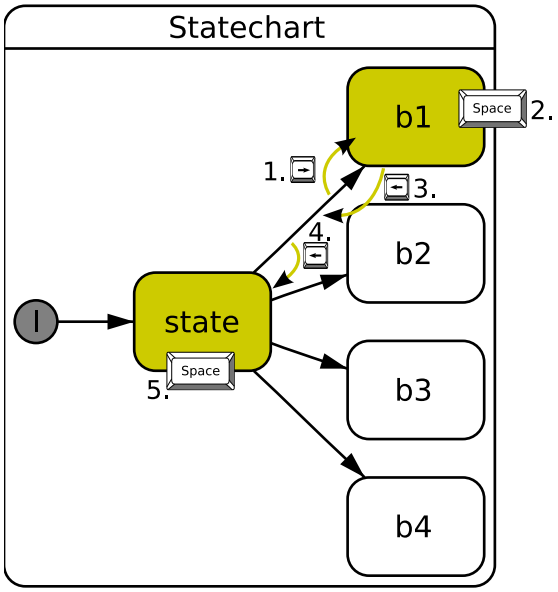


Abbildung 3.16.: bestätigen der Auswahl

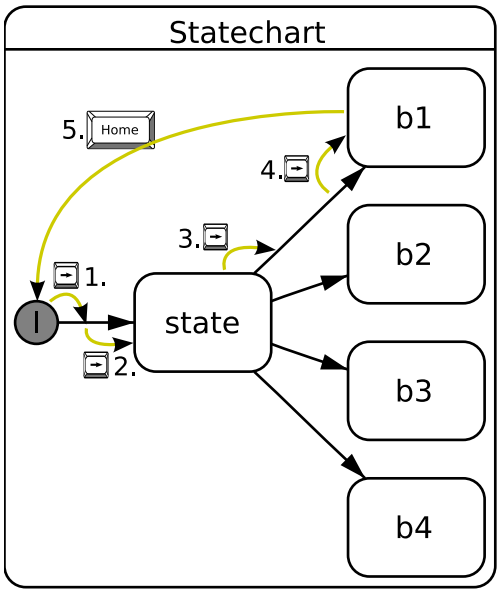


Abbildung 3.17.: zurückspringen zum Anfang der Sequenz

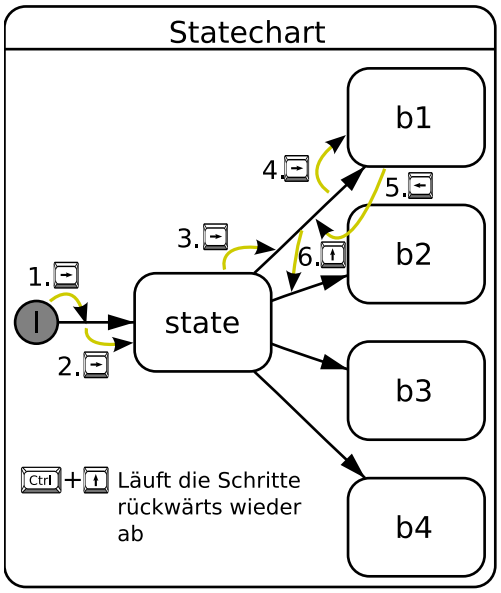


Abbildung 3.18.: durchlaufen der Historie

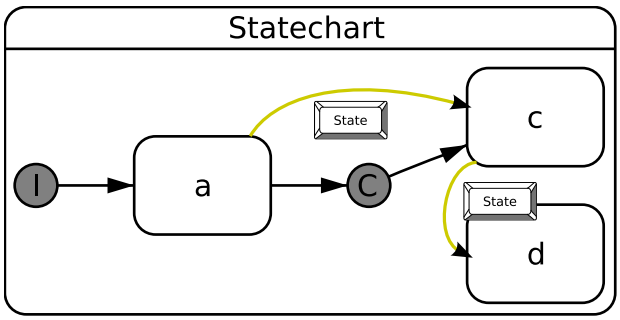


Abbildung 3.19.: auswählen des nächsten Zustandes

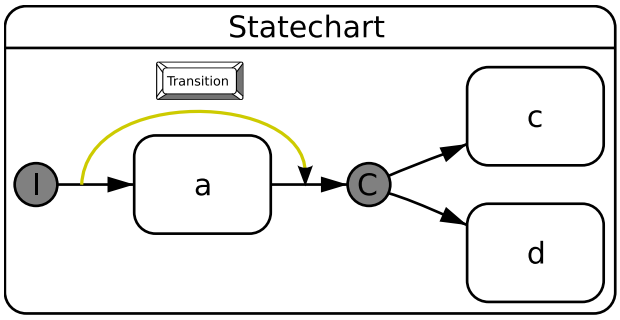


Abbildung 3.20.: auswählen der nächsten Transition

[↑] könnten die bereits getätigten Schritte erneut durchgeführt werden, siehe Abbildung 3.18.

Next State Mit dieser Taste könnte von der momentan selektierten Komponente ausgehend der nächste Zustand gesucht werden. Der Zustand könnte mit Hilfe einer Breitensuche über alle Komponenten des Statecharts bestimmt werden, siehe Abbildung 3.19.

Next Transition Mit dieser Taste könnte von der momentan selektierten Komponente ausgehend die nächste Transition gesucht werden. Dabei könnte eine Breitensuche über alle Komponenten des Statecharts durchgeführt werden. Das wird in Abbildung 3.20 veranschaulicht.

Next PseudoState Mit dieser Taste könnte, von der momentan selektierten Komponente aus gesehen, der nächste Pseudo-Zustand gesucht werden. Das könnte mit Hilfe einer Breitensuche über alle Komponenten des Statechart geschehen, siehe Abbildung 3.21.

Search Nach Eingabe einer Zeichenkette könnte diese mit Hilfe einer Breitensuche mit allen Namen verglichen werden. Die erste gefundene Übereinstimmung könnte selektiert werden, siehe Abbildung 3.22.

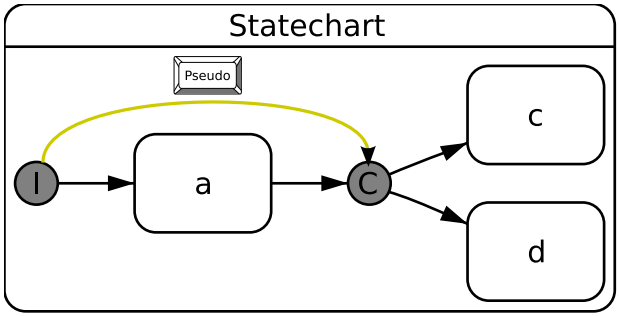


Abbildung 3.21.: auswählen des nächsten Pseudo-Zustandes

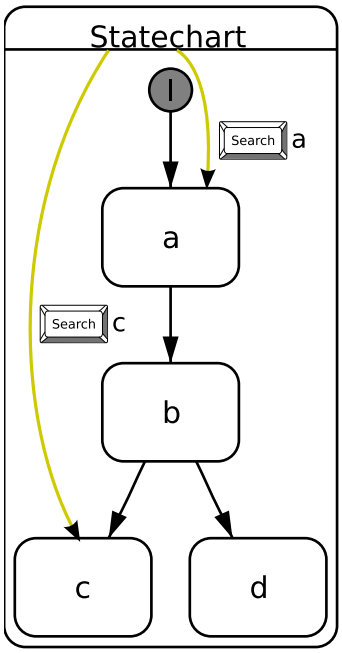


Abbildung 3.22.: suchen einer Komponente

Nachdem die Möglichkeiten zur Navigation in Statecharts beleuchtet wurden, wird im nächsten Abschnitt die Anwendung der Editier-Schemata erläutert.

3.3.2. Schema anwenden

Die Anwendung der Schemata funktioniert in den konventionellen Statechart-Editoren in der Regel über die wiederholte Auswahl entsprechender Aktionen und mittels Zeigens und Klickens mit der Maus auf die entsprechende Komponente (*point'n'click*). Transitionen werden häufig durch das Klicken mit der Maus auf den Quellzustand und das Ziehen zum Zielzustand (*drag'n'drop*) erzeugt. Der Zeitaufwand für die einzelnen Aktionen ist gering, um jedoch ein komplettes Schema umsetzen zu können, sind mehrere solcher Schritte hintereinander notwendig. Dieses Problem hat folgende Ursache: Ohne layoutunterstützende Maßnahmen ist es nur unter bestimmten Bedingungen möglich, mehr als einen Zustand zur Zeit hinzuzufügen, da zuerst Platz für jeden neuen Zustand geschaffen werden muss. Außerdem muss jede neu hinzugefügte Komponente mit einer Positionsangabe versorgt werden, d.h. mit Maus platziert werden. Daher ist ein automatisches Layout die Grundlage, die diese Arbeit erst möglich macht.

Im Gegensatz zur graphischen Sichtweise kann das Statechart, wie oben schon erwähnt, auf einer strukturellen Ebene bearbeitet werden. In einer solchen Darstellung wird, losgelöst von der visuellen Ausprägung der einzelnen Komponenten, nur die Struktur des Statecharts betrachtet. Durch eine solche Sichtweise ist es möglich, die in Abschnitt 3.1 vorgestellten Schemata direkt umzusetzen. Im Anschluss der strukturellen Umsetzung wird ein automatisches Layout nötig, um aus der veränderten Struktur wieder eine visuelle Darstellung des Statecharts zu gewinnen. Inwieweit die Strukturänderungen mit den Techniken der vorhandenen Statechart-Editoren umgesetzt werden können, soll in diesem Abschnitt analysiert werden.

Die Vorschläge zur Optimierung zeichnen sich dadurch aus, dass in der Regel mehr als eine Komponente auf einmal hinzugefügt wird. Für das Verbinden von zwei Zuständen ist es nötig, zwei Zustände zu wählen, siehe Abbildung 3.5. Die beiden Zustände müssen, bevor die entsprechende Aktion stattfinden kann, selektiert werden. In diesem Fall dreht sich die Reihenfolge um, erst werden die Komponenten selektiert und danach findet die Auswahl der Aktion statt. Die Bearbeitung aller Aktionen in dieser Reihenfolge ermöglicht es, jeder Aktion eine Tastenkombination zuzuweisen, da die beteiligten Komponenten schon vorher ausgewählt wurden. In syntaxgerichteten Editoren ist dieses Verhalten nicht möglich, da zusätzliche graphische Informationen, die nur mittels einer Positionsangabe aus der Zeichenfläche erlangt werden können, benötigt werden. Die Aktion wird zwar mit einer Tastenkombination ausgewählt, trotzdem wird für die Positionsangaben eine Maus gebraucht. Der Sachverhalt wurde nochmals in Abbildung 3.23 anhand des syntaxgerichteten Editors des Projektes *KIEL* verdeutlicht.

In dem vorgeschlagenen Verfahren, die Bearbeitungsreihenfolge umzudrehen, kann komplett auf die Maus verzichtet werden, da alle nötigen Angaben auf eine schnelle Art und Weise mit der Tastatur zu bestimmen sind. Das Suchen und Selektieren

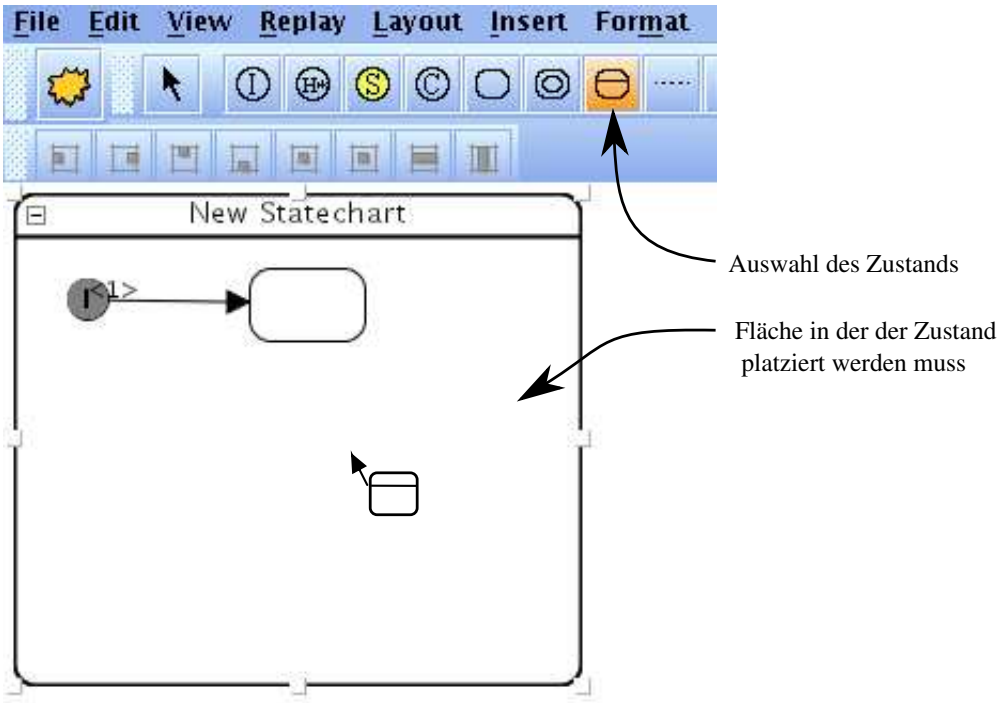


Abbildung 3.23.: Positionierung von Zuständen

findet dabei über die neu vorgestellten Ansätze aus Abschnitt 3.3.1 statt.

3.4. Beschreibungssprachen

Alle in den vorherigen Abschnitten vorgestellten Änderungen arbeiten auf der Topologie des Statecharts. Sie entstanden durch die Optimierung der vorhandenen Konzepte in den bekannten Statechart-Editoren. Findet die Arbeit alleine auf der Struktur des Statecharts statt, ist es ebenfalls denkbar, auf einer textuellen Repräsentation der Statechart-Struktur zu arbeiten. Eine textuelle Repräsentation beschreibt die Struktur des Statecharts und wird daher Statechart-Beschreibungssprache genannt.

In der Literatur gibt es bereits verschiedene Ansätze zu unterschiedlichen Beschreibungssprachen, eine entsprechende Übersicht befindet sich im Anhang A. Folgende Anforderungen werden an die Beschreibungssprache gestellt:

1. Beschreibung der Topologie
2. kompakte Darstellung
3. gute Editierbarkeit
4. gute Nachvollziehbarkeit

Die Sprache soll die Topologie eines Statecharts beschreiben und die Darstellung, bezogen auf Länge der Beschreibung, kompakt sein. Die wichtigsten Anforderungen für eine Beschreibungssprache sind: einfache Editierbarkeit und Lesbarkeit. Eine entsprechende Spezifikation ist in der *Human-Usable Textual Notation Specification* (HUTN) [15] zu finden. Diese Spezifikation beschreibt unter anderem Kriterien zur Benutzerfreundlichkeit von textuellen Notationen und die automatische Umsetzung einer vorhandenen Sprache in eine menschenlesbare Notation.

> „HUTN is a mapping from MOF models to human-usable textual notations, based on a number of usability principles."
> *(DSTC Pegamento Projekt)* [25]

Da keine der in Anhang A aufgezählten Beschreibungssprachen alle Anforderungen erfüllt, wurde eine neue Statechartbeschreibungssprache entwickelt. Grundlage der neuen Beschreibungssprache ist die Spezifikationssprache für *dot* aus dem *Graphviz* Projekt [10], mit deren Hilfe sich Graphen beschreiben lassen. Von Nachteil ist, dass weder die für Statecharts nötige Hierarchie, noch Statechart spezifische Eigenschaften wie Pseudo-Zustände oder Signal Deklarationen unterstützt werden. Die Sprache wird *KIT* (***KI****el statechart extension of do****T***) genannt. Ein Beispiel einer *KIT*-Datei ist in Auflistung 3.1 und das entsprechende Statechart in Abbildung 3.24 zu finden.

Auflistung 3.1: Beispiel einer *KIT* Statechart-Beschreibung

```
statechart absync[model="Esterel Studio";version="5.0";]{
 output AB;
 input A;
 input B;
 input P;
 input R;
 [localEvent="Disarm"]{
  ABSync[localEvent="Arm"]{
    WaitAB{
      ->wA;
      wA->dA[type=sa;label="A / Arm"];
      dA[type=final];
     ||
      ->wB;
      wB->dB[type=sa;label="B / Arm"];
      dB[type=final];
    };
    WaitAB->done[type=nt;label="/ AB"];
    ->WaitAB;
   ||
    ->idle;
    idle->Counting[type=sa;label="Arm"];
    Counting{
     ->count;
     count->count[type=sa;label="2 tick / Disarm"];
    };
    s1[type=suspend];
    s1->Counting[label="P"];
  };
  ABSync->ABSync[type=sa;label="R"];
  ABSync->ABSync[priority="2";type=wa;label="Disarm"];
  ->ABSync;
 };
};
```

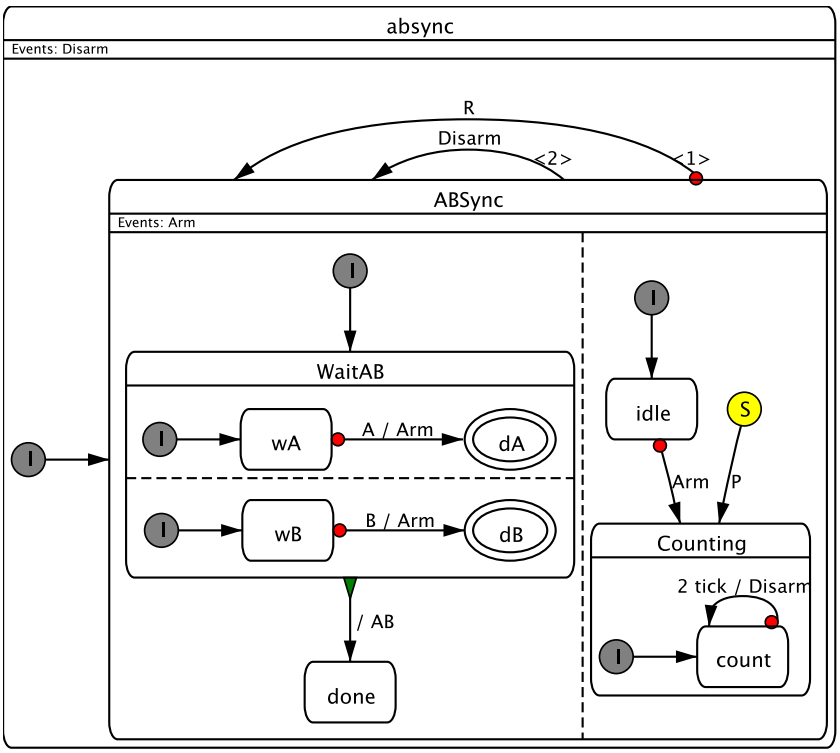


Abbildung 3.24.: Visualisierung von Auflistung 3.1

Die Navigation in der Beschreibungssprache gestaltet sich besonders einfach, da hierfür der benutzte Texteditor zuständig ist. Eine textuelle Darstellung ist linear, daher ist das Editieren und Suchen einfach. Nachteile einer textuellen Darstellung sind, dass im Text schnell die Übersicht verloren geht und die Hierarchie oder Nebenläufigkeit schwer ersichtlich ist. Die Vorteile einer graphischen Sicht sind, dass Hierarchie und Nebenläufigkeit besonders gut zu erkennen sind. Daher wird ein optimales Ergebnis erzielt, wenn sich textuelle und graphische Sicht gegenüberstehen.

Nachdem der Modellierungsprozess genau analysiert und zahlreiche Vorschläge zur Optimierung gemacht wurden, sollen im nächsten Kapitel das klassische syntaxgerichtete Editieren und das Editieren der Struktur gegenübergestellt werden, um etwaige Vor- und Nachteile zu evaluieren.

4. Gegenüberstellung und Analyse der Modellierungsverfahren

In diesem Kapitel werden die drei Modellierungsverfahren, das syntaxgerichtete Editieren, das Strukturediteren und das Editieren mit *KIT* einander gegenübergestellt. Dabei werden die Vor- und Nachteile der verschiedenen Ansätze dargestellt. Als graphischer Editor wurde der in *KIEL* integrierte Editor gewählt. Diese Wahl erlaubt einen direkteren Vergleich, da in diesem Fall alle drei Verfahren auf derselben Datenstruktur arbeiten. Die benötigten Schritte lassen sich direkt auf dem Statechart-Editor von *Esterel Studio* umsetzen. Das Strukturediteren wird anhand der neu implementierten Eigenschaften des *KIEL*-Statechart-Browsers untersucht. Hierbei werden zwei verschiedene Ansätze betrachtet: Zum Einen das Editieren der Struktur mit Hilfe von Änderungsaktionen, siehe Abschnitt 5.2.2 und zum Anderen das Editieren mit Hilfe einer Beschreibungssprache, siehe Abschnitt 5.2.4. Details zur Funktionsweise von *KIT* befinden sich in Abschnitt 5.2.4. Als graphischer Editor wurde

4.1. Gegenüberstellung: Ein ausführliches Beispiel

Zum Vergleich der drei Verfahren wird der Editiervorgang anhand eines Beispiels erläutert und jeder Schritt in den einzelnen Verfahren nachvollzogen. Abschließend findet eine Analyse der einzelnen Schritte nach verschiedenen Kriterien statt. Das zu erstellende Modell, das dazu dienen soll, die Verfahren einander gegenüber zu stellen, beschreibt eine vereinfachte Ampelanlage. Dieses ist in Abbildung 4.1 zu sehen. Um die Anzahl der Schritte nicht unnötig zu vergrößern, werden Schritte, die in allen drei Verfahren ähnlich ablaufen, zusammengefasst. Die folgenden Abschnitte erläutern den kompletten Modellierungsprozess des gesamten Ampelmodells, vom Erzeugen bis zum Hinzufügen der letzten Zustände.

4.1.1. Erzeugen eines neuen Statecharts

Zu Beginn des Editiervorgangs muss ein neues Statechart erzeugt werden. Das lässt sich im Browser und im Editor mit Hilfe eines Tastenkürzels oder über das *File*-Menü erledigen und führt in beiden Modulen zu unterschiedlichen Ergebnissen:

Editor: Es wird ein leeres Statechart, das aus dem *Root*-Zustand besteht, erzeugt. Der Name des Zustandes lautet „New Statechart“.

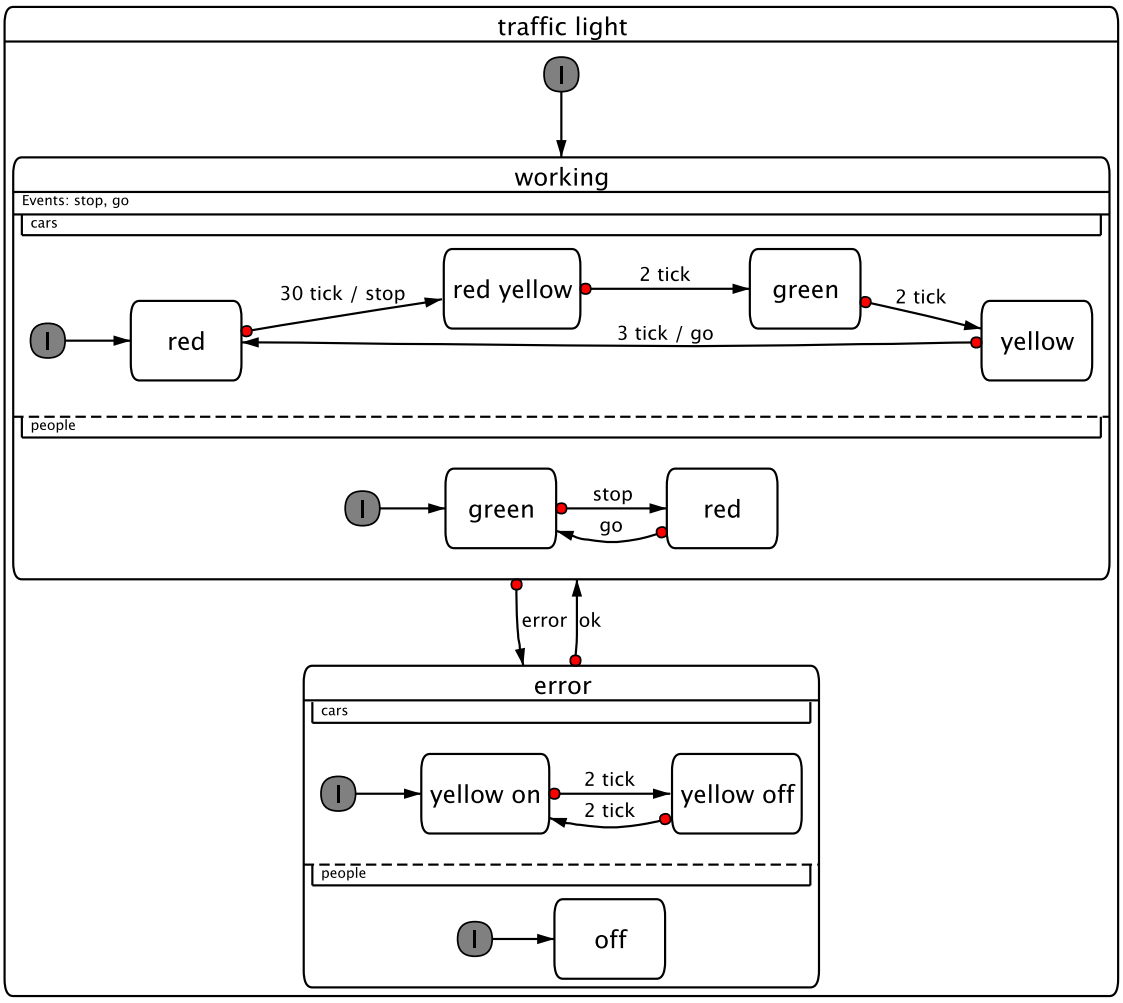
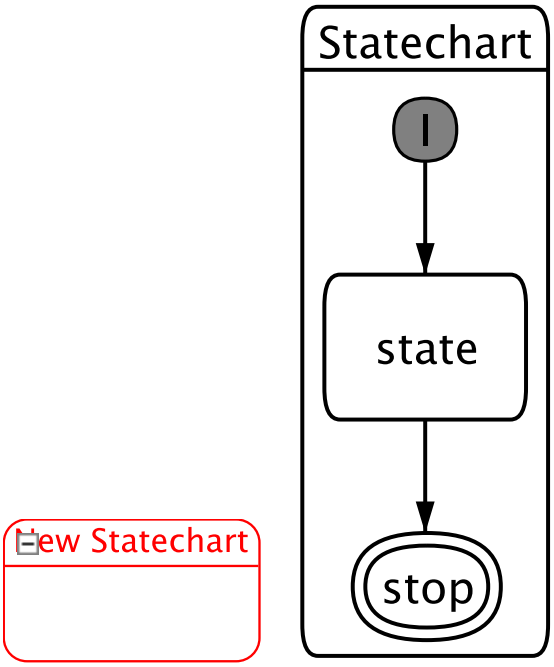


Abbildung 4.1.: Modell einer Ampelanlage



(a) im Editor (b) im Browser

```
statechart Statechart[model="Esterel Studio";version="5.1";]{
 {
  ->state;
  state->stop;
  stop[type=final;];
 };
};
```

(c) in *KIT*

Abbildung 4.2.: Ergebnisse nach dem Erzeugen eines neuen Statecharts

Browser: Es wird ein Statechart erzeugt, das aus einem *Root*-Zustand „Statechart“ besteht. Der *Root*-Zustand enthält drei Zustände: einem Startzustand, der auf einen einfachen Zustand „state“ zeigt, der wiederum auf einen Endzustand „stop“ hinweist.

In Abbildung 4.2 sind die Ergebnisse der Schritte zusammengefasst. Nachdem ein neues Statechart erzeugt wurde, müssen die nötigen Komponenten zum *Root*-Zustand hinzugefügt werden.

4.1.2. Erzeugen der Komponenten auf höchster Ebene

Jetzt werden die Komponenten auf höchster Ebene, dem *Root*-Zustand, erstellt. Dafür müssen ein Startzustand, zwei OR-States und entsprechende Transitionen erzeugt werden. Die dafür nötigen Schritte in den einzelnen Verfahren werden in

Tabelle 4.1 einander gegenübergestellt. Hierbei werden Aktionen, die einen ähnlichen Aufwand wie ihr Editor-Pendant verursachen zusammengefasst. Auf gleicher Ebene befindliche Schritten entsprechen nicht gleichen Aktionen. Die Ergebnisse der Änderungen von Tabelle 4.1 sind in Abbildung 4.3 zu sehen.

Tabelle 4.1.: Komponentenerzeugung auf höchster Ebene

	Editor	**Browser**	***KIT***
1.	auswählen des Startzustandes aus der Werkzeugleiste	auswählen des einfachen Zustandes „state“ (3 × → oder klicken auf „state“)	entfernen von Zeile 5: `stop[type=final];`
2.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	umwandeln des Zustandes zu einem OR-State	hinzufügen in Zeile 5: `state{};`
3.	auswählen des OR-States aus der Werkzeugleiste	auswählen des „stop“ Zustandes (2 × → oder klicken auf „stop“)	hinzufügen in Zeile 6: `stop{};`
4.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	umwandeln des Zustandes zu einem OR-State	ändern in Zeile 5: `start[label= "working"]{};` und ändern in Zeile 6: `stop[label= "error"]{};`
5.	auswählen des OR-States aus der Werkzeugleiste	ändern der Attribute, um den Zustand nicht mehr *final* zu machen	hinzufügen in Zeile 5: `stop->state;`
6.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	bestätigen der Auswahl des Zustandes mit Space	hinzufügen in Zeile 2: `input er- ror; input ok;`
7.	klicken auf den Startzustand und ziehen auf einen OR-State	auswählen des „state“ Zustandes (2 × ← oder klicken auf „state“) und verbinden der beiden Zustände mit einer Transition	ändern in Zeile 5: `state->stop[type =sa;label="error"];` und ändern in Zeile 6: `stop->state[type =sa;label="ok"];` ändern in Zeile 2: `[label= "traffic light"]{`

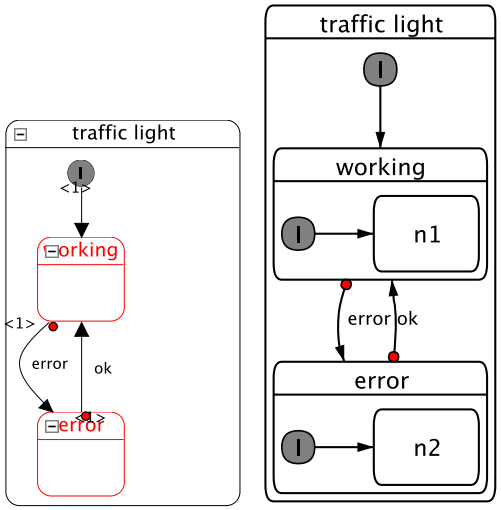
Fortsetzung folgt auf der nächsten Seite

Tabelle 4.1 - Fortsetzung

	Editor	**Browser**	***KIT***
8.	klicken auf den eben verbundenen OR-State und ziehen auf den anderen OR-State	beschriften der Zustände einschließlich des *Root*-States	
9.	klicken auf den letzten OR-State und ziehen auf den vorherigen OR-State	hinzufügen von *Events* zum Statechart: `error` und `ok`	
10.	erzeugen eines Stützpunktes auf einer der Transitionen	beschriften der Transitionen	
11.	verschieben der übereinanderliegenden Transitionen		
12.	beschriften der Zustände, einschließlich des *Root*-States		
13.	hinzufügen von *Events* zum Statechart: `error` und `ok`		
14.	beschriften der Transitionen		

Dabei fällt auf, dass sich das Ergebnis des Browsers von den Ergebnissen der beiden anderen Verfahren, auf Grund der im Browser umgesetzten Änderungsschemata, unterscheidet. Zustände, die für ein Statechart nötigt sind, werden automatisch mit Hilfe der Aktionsregeln erzeugt. In diesem Fall wird ein initialer Zustand mit einem einfachen Zustand verbunden und den neu erzeugten hierarchischen Zuständen hinzugefügt.

Der Browser benötigt im Gegensatz zum Editor weniger Schritte, da schon beim Erzeugen des Statecharts einige Komponenten erstellt werden. Aus diesem Grund sind im Browser keine äquivalenten Schritte zu 1., 2. und 7. zu finden. Einen weiteren Vorteil stellt das automatische Layout dar, das den Schritt 11. beim Editor wegfallen lässt. Die Beschreibungssprache *KIT* bedarf noch weniger Schritte als der Browser oder der Editor. Für *KIT* treffen zwar dieselben Optimierungen wie im Browser zu, jedoch entfällt das Auswählen der Zustände. Dadurch sind im Vergleich zum Browser die Schritte 1.,2. und 6. nicht nötig.



(a) im Editor (b) im Browser

```
statechart Statechart[model="Esterel Studio";version="5.1";]{
 input error; input ok;
 [label="traffic light"]{
  ->state;
  state->stop[label="error"];
  stop->state[label="ok"];
  state[label="working"]{};
  stop[label="error"]{};
 };
};
```

(c) in *KIT*

Abbildung 4.3.: Ergebnisse nach dem Erzeugen der Komponenten auf höchster Ebene

In einem nächsten Schritt werden die OR-States um die Zuständen und Transitionen aus der Region `cars` erweitert.

4.1.3. Erzeugen von Komponenten in OR-States

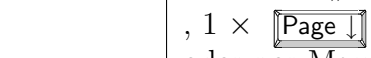
In den im vorherigen Schritt erstellten OR-States sollen als nächstes zusätzliche Komponenten erzeugt werden. Dabei wird bewusst auf das Umwandeln zu AND-States und das Hinzufügen von mehreren Regionen verzichtet, da dieses eher dem Entwicklungsprozess eines Modells entspricht. In einem Entwicklungsprozess ist das komplette Statechart vorher nicht bekannt, daher wird zunächst nur ein Teilaspekt, der im nächsten Schritt erweitert wird, modelliert. Die benötigten Schritte sind in Tabelle 4.2 dargestellt und auf gleicher Ebene befindliche Schritten entsprechen nicht gleichen Aktionen. Die Ergebnisse der Änderungen sind in Abbildung 4.4 dargestellt.

Tabelle 4.2.: Komponentenerzeugung der Hierarchischen Zustände

	Editor	**Browser**	***KIT***
1.	vergrößern des OR-States „working“ auf die benötigte Fläche	erweitern von „working“: auswählen des Zustandes „n1“ (3 × [→] , 1 × [Page ↓] , 2 × [→] oder per Mausauswahl)	hinzufügen in Zeile 8: `->red;`
2.	vergrößern des OR-States „error“ auf die benötigte Fläche	ändern des Namens zu „red“	hinzufügen in Zeile 9: `red->red_yellow [type=sa];`
3.	erweitern von „working“: auswählen des Startzustandes aus der Werkzeugleiste	hinzufügen eines neuen Zustandes „n3“ mit einer Transition, beginnend bei „red“	hinzufügen in Zeile 10: `red_yellow->green [type=sa];`
4.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	umbenennen des Zustandes „n3“ zu „red yellow“	hinzufügen in Zeile 11: `green->yellow [type=sa];`
5.	auswählen des einfachen Zustandes aus der Werkzeugleiste	hinzufügen eines neuen Zustandes „n4“ mit einer Transition, beginnend bei „red yellow“	hinzufügen in Zeile 12: `yellow->red [type=sa];`
6.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	umbenennen des Zustandes „n4“ zu „green“	hinzufügen in Zeile 13: `red_yellow[label= "red yellow"];`
7.	umbenennen des Zustandes in „red“	hinzufügen eines neuen Zustandes „n5“ mit einer Transition, beginnend bei „green“	ändern in Zeile 7: `state[label= "working"; localEvent= "stop, go"]{`
8.	auswählen des einfachen Zustandes aus der Werkzeugleiste	umbenennen des Zustandes „n5“ zu „yellow“	hinzufügen von Beschriftungen an den Transitionen
9.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	zusätzliches auswählen vom Zustand „red“ und Verbinden von „yellow“ nach „red“	hinzufügen in Zeile 16: `->yon;`
10.	umbenennen des Zustandes in „red yellow“	hinzufügen von lokalen Events: `go` und `stop`	hinzufügen in Zeile 17: `yon->yoff [type=sa];`

Fortsetzung folgt auf der nächsten Seite

Tabelle 4.2 - Fortsetzung

	Editor	**Browser**	***KIT***
11.	Zustand vergrößern, damit der Name komplett sichtbar wird	hinzufügen von Beschriftungen an den Transitionen	hinzufügen in Zeile 18: `yoff->yon [type=sa];`
12.	auswählen des einfachen Zustandes aus der Werkzeugleiste	erweitern von „error“: auswählen des Zustandes „n2“ (2 ×  , 1 ×  , 2 × oder per Mausauswahl)	hinzufügen in Zeile 19: `yon[label= "yellow on"];`
13.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	ändern des Namens zu „yellow on“	hinzufügen in Zeile 20: `yoff[label= "yellow off"];`
14.	umbenennen des Zustandes in „green“	hinzufügen eines neuen Zustandes „n6“ mit einer Transition, beginnend bei „yellow on“	hinzufügen von Beschriftungen an den Transitionen
15.	auswählen des einfachen Zustandes aus der Werkzeugleiste	umbenennen des Zustandes „n6“ zu „yellow off“	
16.	klicken auf die Zeichenfläche, um den Zustand zu platzieren	zusätzliches auswählen vom Zustand „yellow on“ und verbinden von „yellow off“ nach „yellow on“	
17.	umbenennen des Zustandes in „yellow“	hinzufügen von Beschriftungen an den Transitionen	
18.	erzeugen einer Transition vom Startzustand nach „red“		
19.	erzeugen einer Transition von „red“ nach „red yellow“		
20.	erzeugen einer Transition von „red yellow“ nach „green“		
21.	erzeugen einer Transition von „green“ nach „yellow“		

Fortsetzung folgt auf der nächsten Seite

Tabelle 4.2 - Fortsetzung

	Editor	**Browser**	***KIT***
22.	erzeugen einer Transition von „yellow“ nach „red“		
23.	hinzufügen von lokalen Events: `go` und `stop`		
24.	hinzufügen von Beschriftungen an den Transitionen		
25.	erweitern von „error“: auswählen des Startzustandes aus der Werkzeugleiste		
26.	klicken auf die Zeichenfläche, um den Zustand zu platzieren		
27.	auswählen des einfachen Zustandes aus der Werkzeugleiste		
28.	klicken auf die Zeichenfläche, um den Zustand zu platzieren		
29.	umbenennen des Zustandes in „yellow on“		
30.	vergrößern des Zustandes, damit der Name komplett sichtbar wird		
31.	auswählen des einfachen Zustandes aus der Werkzeugleiste		
32.	klicken auf die Zeichenfläche, um den Zustand zu platzieren		
33.	umbenennen des Zustandes in „yellow off“		
34.	vergrößern des Zustandes, damit der Name komplett sichtbar wird		

Fortsetzung folgt auf der nächsten Seite

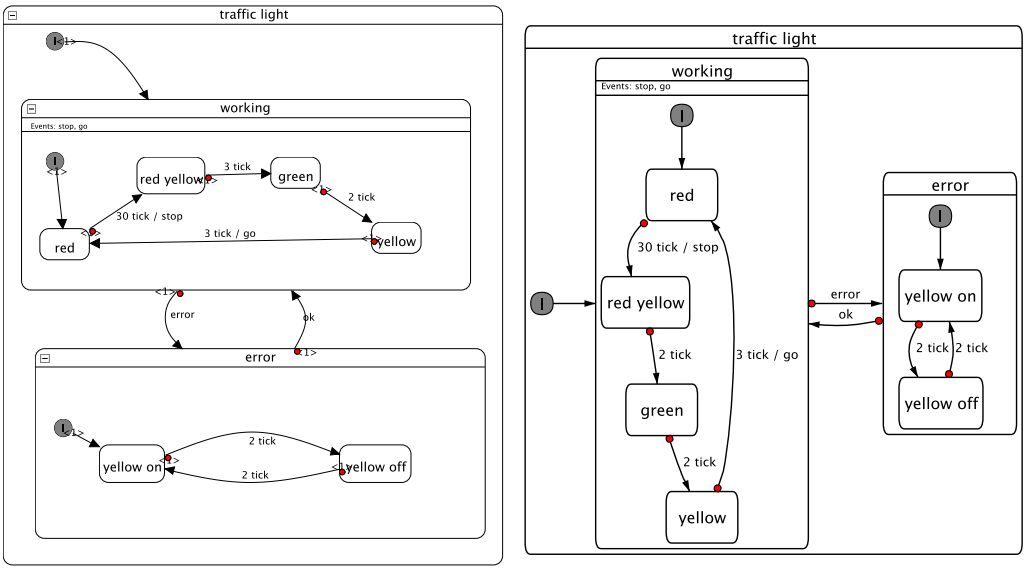
Tabelle 4.2 - Fortsetzung

	Editor	Browser	*KIT*
35.	erzeugen einer Transition vom Startzustand nach „yellow on“		
36.	erzeugen einer Transition von „yellow off“ nach „yellow off“		
37.	erzeugen einer Transition von „yellow on“ nach „yellow off“		
38.	erzeugen eines Stützpunktes auf einer der Transitionen		
39.	verschieben übereinanderliegender Transitionen		
40.	hinzufügen von Beschriftungen an den Transitionen		

Nach dem Erzeugen der Komponenten in OR-States liefern alle drei Verfahren ein identisches Ergebnis. Der Unterschied in der Anzahl der benötigten Schritte fällt in diesem Abschnitt wesentlich deutlicher aus als zuvor.

Im Browser sind im Gegensatz zum Editor nur 42% der Schritte notwendig. Das hat zwei Ursachen: Zum Einen sind Änderungen am Layout erforderlich, bspw. in den Schritten 1., 2., 30. und 34., zum Anderen müssen wesentlich mehr Komponenten erzeugt werden als mit dem Browser. Bei jedem Hinzufügen eines Zustandes wird schon eine passende Transition erzeugt. Hier sind im Editor jeweils drei Schritte notwendig (z. B. 31., 32. und 36.). Im Browser hingegen sind nur zwei Schritte nötig (entsprechend 12. und 14.). Durch die schon im vorherigen Abschnitt erzeugten Zustände innerhalb der hierarchischen Zustände werden im Browser noch zusätzliche Schritte gespart. *KIT* benötigt noch weniger Schritte als der Browser oder der Editor. Durch das implizite Erzeugen von Zuständen, ist es in *KIT* möglich, in nur einem Schritt zwei Zustände, die miteinander verbunden sind, zu erzeugen. Zusätzlich dienen die Namen der Zustände, vorausgesetzt sie enthalten keine Leerzeichen, als Bezeichner und das Benennen der Zustände ist überflüssig.

Damit die Ampelanlage fertig modelliert ist, müssen im letzten Schritt die parallelen Zweige hinzugefügt werden.



(a) im Editor

(b) im Browser

```
1  statechart Statechart[model="Esterel Studio";version="5.0";]{
2   input error; input ok;
3   [label="traffic light"]{
4    ->state;
5    state->stop[type=sa;label="error"];
6    stop->state[type=sa;label="ok"];
7    state[label="working"; localEvent="stop, go"]{
8      ->red;
9      red->red_yellow[type=sa;label="30 tick/stop"];
10     red_yellow->green[type=sa;label="2 tick"];
11     green->yellow[type=sa;label="2 tick"];
12     yellow->red[type=sa;label="3 tick/go"];
13     red_yellow[label="red yellow"];
14   };
15   stop[label="error"]{
16     ->yon;
17     yon->yoff[type=sa;label="2 tick"];
18     yoff->yon[type=sa;label="2 tick"];
19     yon[label="yellow on"];
20     yoff[label="yellow off"];
21   };
22  };
23 };
```

(c) in *KIT*

Abbildung 4.4.: Ergebnisse nach dem Erzeugen der Komponenten in den OR-States

4.1.4. Hinzufügen der parallelen Zweige

In diesem Abschnitt soll das Modell komplettiert werden. Dafür werden die noch fehlenden parallelen Zweige `people` zu den Zuständen „working“ und „error“ hinzugefügt. Die genaue Vorgehensweise ist in Tabelle 4.3 illustriert. Auf gleicher Ebene befindliche Schritten entsprechen nicht gleichen Aktionen.

Tabelle 4.3.: Gegenüberstellung vom Hinzufügen der parallelen Zweige

	Editor	**Browser**	***KIT***
1.	Platz-Schaffen: *Root*-Zustand , „working“ und „error“ vergrößern und neu positionieren, Transitionsverlauf anpassen	auswählen des Zustandes „working“ (2 ×  oder per Mausauswahl))	hinzufügen in Zeile 14: `||`
2.	erweitern von „working“: vorhandene Zustände neu platzieren, um Platz für eine parallelen Zweig zu schaffen	umwandeln in ein ANDState	hinzufügen in Zeile 15: `->pgreen;`
3.	auswählen vertikaler Trennlinien aus der Werkzeugleiste	benennen des Zustandes „n1“ um in „green“	hinzufügen in Zeile 16: `pgreen->pred [type=sa];`
4.	positionieren der Trennlinie durch klicken auf die Zeichenfläche	hinzufügen eines neuen Zustandes „n2“ mit einer Transition, beginnend bei „green“	hinzufügen in Zeile 17: `pred->pgreen [type=sa];`
5.	auswählen des Startzustandes aus der Werkzeugleiste	benennen des Zustandes „n2“ um in „red“	hinzufügen in Zeile 18: `pgreen[label= "green"];`
6.	positionieren des Startzustandes durch klicken auf die Zeichenfläche	zusätzliches auswählen vom Zustand „green“ und verbinden von „red“ nach „green“	hinzufügen in Zeile 19: `pred[label="red"];`
7.	auswählen des einfachen Zustandes aus der Werkzeugleiste	beschriften der Transitionen	hinzufügen von Beschriftungen an den Transitionen
8.	positionieren des Zustandes durch klicken auf die Zeichenfläche	benennen der parallelen Zweige mit „cars“ und „people“	hinzufügen in Zeile 8: `<cars>` und in Zeile 16: `<people>`

Fortsetzung folgt auf der nächsten Seite

Tabelle 4.3 - Fortsetzung

	Editor	**Browser**	***KIT***
9.	umbenennen des Zustandes zu „green“	auswählen des Zustandes „error“ (Page ↑ und 2 × → oder per Mausauswahl))	hinzufügen in Zeile 29: `||`
10.	auswählen des einfachen Zustandes aus der Werkzeugleiste	umwandeln in ein ANDState	hinzufügen in Zeile 30: `->pgreen;`
11.	positionieren des Zustandes durch klicken auf die Zeichenfläche	benennen des Zustandes „n1“ um in „off“	hinzufügen von Beschriftungen an den Transitionen
12.	umbenennen des Zustandes zu „red“	benennen der parallelen Zweige mit „cars“ und „people“	hinzufügen in Zeile 24: `<cars>` und in Zeile 31: `<people>`
13.	verbinden des Startzustandes mit dem Zustand „green“		
14.	verbinden des Zustandes „green“ mit dem Zustand „red“		
15.	verbinden des Zustandes „red“ mit dem Zustand „green“		
16.	erzeugen eines Stützpunktes für eine der Transitionen		
17.	verschieben übereinanderlappender Transitionen		
18.	beschriften der Transitionen		
19.	benennen der parallelen Zweige mit „cars“ und „people“		
20.	erweitern von „error“: vorhandene Zustände neu platzieren, um Platz für einen parallelen Zweig zu schaffen		

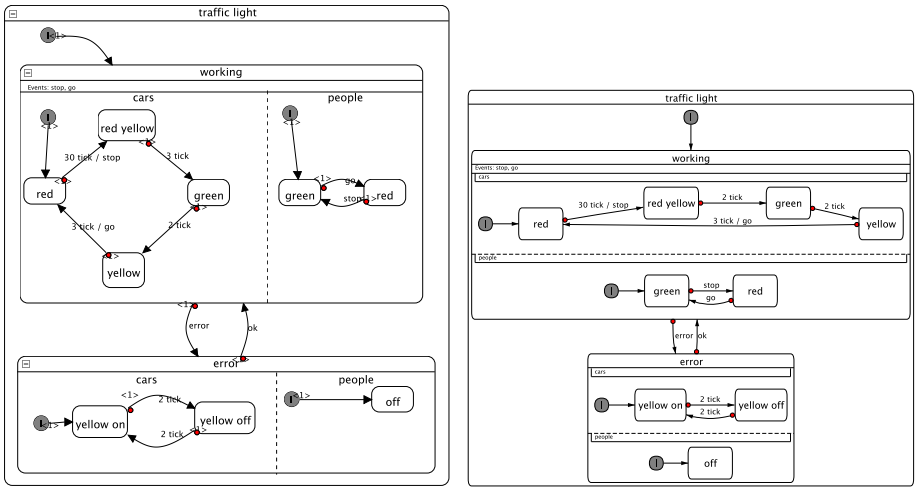
Fortsetzung folgt auf der nächsten Seite

Tabelle 4.3 - Fortsetzung

	Editor	Browser	*KIT*
21.	auswählen vertikaler Trennlinien aus der Werkzeugleiste		
22.	positionieren der Trennlinien durch klicken auf die Zeichenfläche		
23.	auswählen des Startzustandes aus der Werkzeugleiste		
24.	positionieren des Startzustandes durch klicken auf die Zeichenfläche		
25.	auswählen des einfachen Zustandes aus der Werkzeugleiste		
26.	positionieren des Zustandes durch klicken auf die Zeichenfläche		
27.	umbenennen des Zustandes zu „off“		
28.	verbinden des Startzustandes mit dem Zustand „off“		
29.	benennen der parallelen Zweige mit „cars“ und „people“		
30.	verbessern des Layouts: verkleinern der zu großen Zustände		

Das gesamte Modell ist nach den vorgestellten Schritten erstellt worden. Für die Erstellung waren insgesamt 85 Schritte im Editor, 39 Schritte mit Hilfe der Änderungsaktionen im Browser und 34 Änderungen in der *KIT* Beschreibung nötig, siehe Abbildung 4.5. In diesem Abschnitt erfordern die Änderungsaktionen im Gegensatz zum Editor 40% der Schritte. Durch das Hinzufügen der parallelen Zweige, werden im Editor Layoutveränderungen nötig, um Platz für die zu erzeugenden parallelen Zweige zu schaffen. Diese Änderungen fallen im Browser und im *KIT* weg. Die Gründe für das bessere Abschneiden vom Browser und *KIT* sind in diesem Abschnitt identisch mit denen der vorherigen Abschnitte. Insbesondere die Layoutänderungen und

das automatische Hinzufügen von Zuständen beim Erzeugen der parallelen Zweige sorgen im Browser für die geringere Anzahl von Schritten. *KIT* bedarf in diesem Abschnitt genausoviele Schritte wie die Änderungsaktionen, da ein wichtiger Vorteil, das gleichzeitige Benennen der Zustände beim Erzeugen, in diesem Abschnitt nicht möglich ist (siehe dazu z. B. Schritt 5. und 6. bei *KIT*).



(a) im Editor (b) im Browser

```
statechart Statechart[model="Esterel Studio";version="5.0";]{
 input error; input ok;
 [label="traffic light"]{
  ->state;
  state->stop[type=sa;label="error"];
  stop->state[type=sa;label="ok"];
  state[label="working"; localEvent="stop, go"]{
   <cars>
    ->red;
    red->red_yellow[type=sa;label="30 tick/stop"];
    red_yellow->green[type=sa;label="2 tick"];
    green->yellow[type=sa;label="2 tick"];
    yellow->red[type=sa;label="3 tick/go"];
    red_yellow[label="red yellow"];
   ||
    <people>
    ->pgreen;
    pgreen->pred[type=sa;label="stop"];
    pred->pgreen[type=sa;label="go"];
    pgreen[label="green"];
    pred[label="red"];
  };
  stop[label="error"]{
    <cars>
    ->yon;
    yon->yoff[type=sa;label="2 tick"];
    yoff->yon[type=sa;label="2 tick"];
    yon[label="yellow on"];
    yoff[label="yellow off"];
   ||
    <people>
    ->off;
  };
 };
};
```

(c) in *KIT*

Abbildung 4.5.: Ergebnisse nach dem Hinzufügen der parallelen Zweige

Die vorgestellten Schritte sollen im nächsten Abschnitt analysiert werden.

Editierschritt	**Editor**	**Editor(ohne Layout)**	**Browser**	***KIT***
Abschnitt 4.1.2	14	9	9	7
Abschnitt 4.1.3	40	25	17	14
Abschnitt 4.1.4	30	17	12	12
Summe	85	51	39	34

Tabelle 4.4.: Bearbeitungsschritte im Vergleich

4.2. Analyse der Bearbeitungsschritte

In dem Teil des Kapitels sollen die Bearbeitungschritte aus den vorherigen Abschnitten genau analysiert und bewertet werden. Wird zunächst die Anzahl der Aktionen betrachtet, fällt auf, dass im Gegensatz zum Editor im Browser nur 46% der Schritte notwendig sind. Auch wenn alle Schritte, die mit dem Layout des Statecharts zusammenhängen, entfernt werden, benötigt der Browser im Gegensatz zum Editor 76% der Schritte. Im Vergleich mit der Beschreibungssprache *KIT* ist der Unterschied noch etwas gravierender. Mit allen Schritten des Editors verglichen, werden mit Hilfe von *KIT* nur noch 40% der Schritte benötigt. Lässt man die Schritte, die das Layout des Statecharts betreffen weg, sind im Gegensatz zum Editor nur noch 66% der Schritte erforderlich. Die geringere Anzahl an Bearbeitungsschritten entsteht somit nicht alleine durch das automatisierte Layout, sondern durch einen zusätzlichen Faktor. Der zusätzliche Faktor ist das automatische Hinzufügen von Komponenten durch die Änderungsaktionen. Wenn z. B. ein Zustand hinzugefügt wird, findet die automatische Erzeugung einer Transition statt. Im Falle von *KIT* werden durch Transitionen implizit Zustände erzeugt und die gewählten Bezeichner als Namen für die Zustände gesetzt. Durch diese Maßnahmen werden Schritte zusammengefasst, die im Editor einzeln gelöst werden müssen. In der Tabelle 4.4 ist die Anzahl der Schritte im direkten Vergleich aufgelistet.

Inwieweit sich jedoch die deutlich geringe Anzahl an Bearbeitungsschritten positiv auf die Gesamtgeschwindigkeit auswirkt, hängt von verschiedenen Faktoren ab:

1. Expertise
2. Feinmotorik
3. Anzahl der Anschläge
4. Art der Änderungen
5. Komplexität des Modells

Ist der Anwender in der Benutzung graphischer Editoren geschult bzw. ein Experte, wird er schneller als ein Anfänger auf dem Gebiet die einzelnen Schritte umsetzen können. Außerdem kann die Selektion von kleinen Strukturen mit dem Editor von jedem Anwender unterschiedlich schnell umgesetzt werden. Im vorgestellten Fallbeispiel ist die Geschwindigkeit beim Tippen ausschlaggebend, da jeder

	Editor		**Browser**	
Beschriftungen:	mit	ohne	mit	ohne
Gesamtdauer Editiervorgang:	∼ 358,71s	∼ 83,2s	∼ 310,75s	∼ 35,26s

Tabelle 4.5.: Abhängigkeit der Messung von der Tippgeschwindigkeit

Editierschritt	**Editor**	**Browser**	***KIT***
Abschnitt 4.1.2	∼ 61,28s	∼ 43,00s	∼ 55,45s
Abschnitt 4.1.3	∼ 153,93s	∼ 124,75s	∼ 150,25s
Abschnitt 4.1.4	∼ 143,50s	∼ 143,00s	∼ 157,00s
Summe	∼ 358,71s	∼ 310,75s	∼ 362,70

Tabelle 4.6.: Geschwindigkeitsvergleich der Verfahren

Zustand benannt und der zugehörige Name in allen drei Modellierungesverfahren eingegeben werden muss. Diese Anwendung nimmt in allen Verfahren einen großen Teil der Gesamtdauer ein, siehe Tabelle 4.5. Die Änderungsarten sind ebenfalls für die Geschwindigkeit entscheidend. Werden hauptsächlich Änderungen durchgeführt, die im Browser automatisch alle Komponenten erzeugen und im Editor mehrfaches Auswählen und Positionieren erfordern, wird der Geschwindigkeitsvorteil durch die geringere Anzahl der Schritte deutlich. Ist das zu ändernde Modell sehr komplex, sind die nötigen Arbeiten, um Platz für neue Zustände zu gewinnen, so aufwendig, dass allein durch das automatische Layout im Browser ein Geschwindigkeitsvorteil entsteht. In dem vorgestellten Fallbeispiel ist die Dauer, um das gesamte Modell im Browser oder in *KIT* zu erstellen, nur geringfügig kürzer als im Editor. Da es sich hier nur um die Geschwindigkeit beim Tippen bzw. Selektieren und Kopieren handelt, ist der geringe Unterschied stark Anwenderabhängig. Die Ergebnisse der Geschwindigkeitsmessungen sind in Tabelle 4.6 veranschaulicht.

Neben den Geschwindigkeitsbetrachtungen und der Komplexität des Editiervorgangs kann zusätzlich die Qualität analysiert werden. Die Qualität des Editiervorgangs beschreibt das subjektive Empfinden des Editierenden beim Durchführen der Änderungen. Das im graphischen Editor nötige Anpassen des Layouts wird von vielen Modellierenden als unangenehm empfunden, da es viel Zeit erfordert und nicht zur Problemlösung beiträgt. Damit ist das automatische Layout ein Schlüssel zur Qualität des Editiervorgangs. Das als angenehm empfundene automatische Layout, sollte positive Effekte auf die Qualität des Editiervorgangs haben. Untersuchungen zu diesem Thema stehen noch aus. Als große Vorteile des Browsers, im Bezug auf die Qualität des Editiervorgangs, haben sich die vielen Ansichten auf ein Statechart herausgestellt. Der graphischen Sicht auf das Statechart wurde eine textuelle Sicht zur Seite gestellt, was als sehr angenehm empfunden wird. Die Strukturansicht über einen Baum trägt ebenfalls zur Übersichtlichkeit bei.

Nachdem der Modellierungsprozess genau analysiert und die Verfahren einander gegenübergestellt wurden, soll im nächsten Kapitel ein Überblick über die Umset-

zung dieser Arbeit gegeben werden.

5. Umsetzung in KIEL

In diesem Abschnitt soll die Umsetzung der Ideen aus den vorherigen Kapiteln vorgestellt werden. Die Umsetzung erfolgte im Rahmen des Projektes *KIEL* [**?**] (***K****iel* ***I****ntegrated* ***E****nvironment for* ***L****ayout*). Das Modellierungswerkzeug soll im nächsten Abschnitt näher erläutert werden, um schließlich auf die umgesetzten Ideen im Abschnitt 5.2 eingehen zu können.

5.1. Projekt KIEL

KIEL ist ein Modellierungswerkzeug, das am Lehrstuhl für Eingebettete- und Echtzeitsysteme entwickelt wird. Es besteht aus einer Anzahl von Modulen, die jeweils über genau definierte Schnittstellen kommunizieren. Eine Übersicht über die bestehenden Module befindet sich in Abbildung 5.1. Aus Sicht des Benutzers gibt es zwei zentrale Module:

Editor: Der Editor [**?**] ist ein syntaxgerichteter Statechart Editor, wie er im Kapitel 3.2.1 beschrieben wurde.

Browser: Der Browser [**?**] ist ein Programm, um Statecharts anzuzeigen und zu simulieren. Er stellt eine graphische Benutzeroberfläche für die Simulator Module [**?**] [**?**] dar.

Alle Module arbeiten auf einer Datenstruktur für Statecharts, die die Topologie des Statecharts abbildet. Die graphischen Informationen dagegen, wie Position und Größe der einzelnen Komponenten, werden durch ein anderes Modul den *GraphicalInformations* abgebildet. Ein weiteres zentrales Modul ist das Modul Layouter, das alleine auf der Basis der Datenstruktur die graphischen Informationen, die für eine Visualisierung benötigt werden, erzeugt. Der Layout-Prozess lässt sich dabei sowohl über den Browser, als auch über den Editor anstoßen. Das Modul *FileInterface* [28] stellt eine Schnittstelle zur Verfügung, um Statechart-Strukturen mit oder ohne graphische Informationen einzulesen und zu speichern. Die Schnittstellen zu anderen Modellierungswerkzeugen [**?**] werden dabei über Zusatzmodule(*Plugins*) zur Verfügung gestellt. Nach diesem kurzen Überblick zum Aufbau des Projektes *KIEL* werden im nächsten Abschnitt einige Details der anderen Module erläutert.

5.1.1. Implementatorische Details von KIEL

Die *KIEL* Datenstruktur unterscheidet drei Hauptkomponenten für Statecharts:

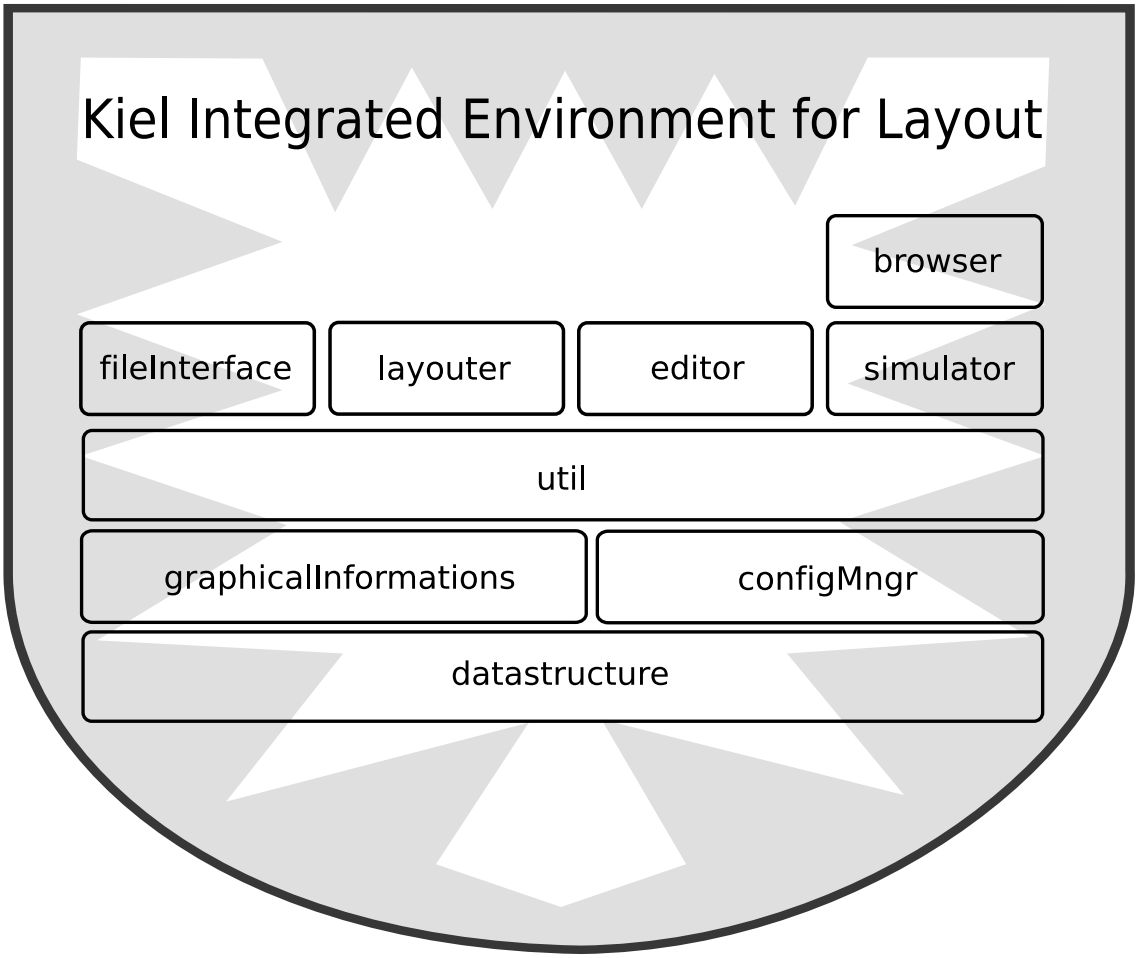


Abbildung 5.1.: Übersicht der Module im Projekt *KIEL*

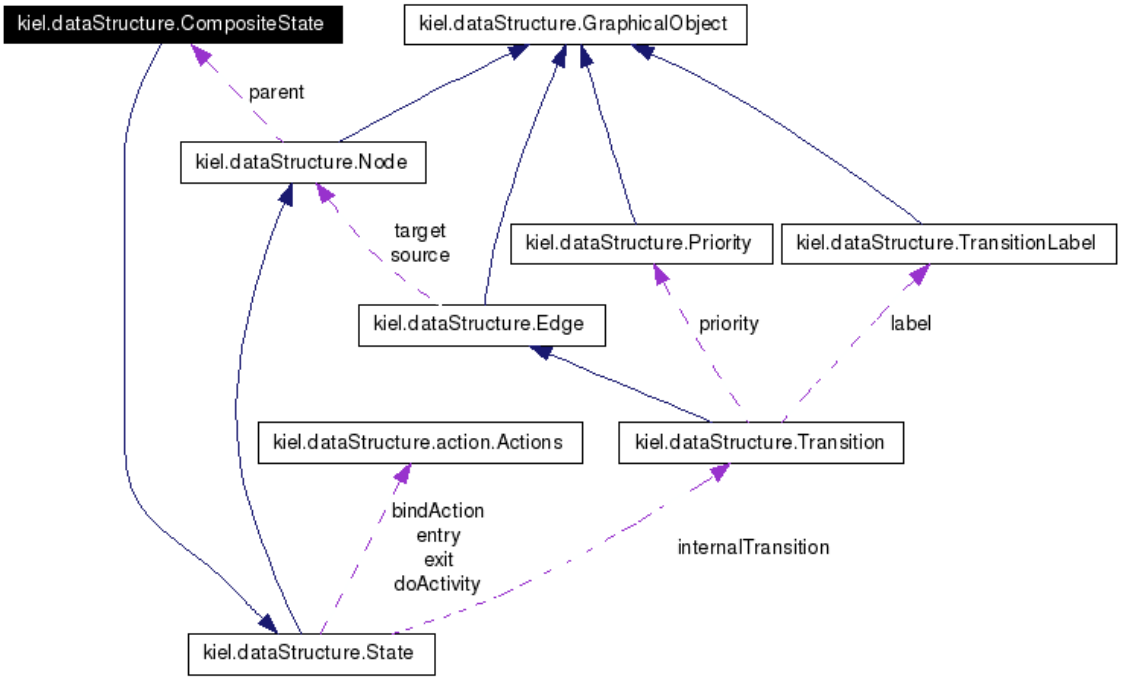


Abbildung 5.2.: Kollaborationsdiagramm der wichtigsten Klassen in *KIEL*

1. Zustände
2. Transitionen
3. Trennlinien

Der Basistyp von Zuständen ist `Node`, von Transitionen `Edge` und Trennlinien sind immer vom Typ `DelimiterLine`. Alle diese Klassen sind von der Klasse `GraphicalObject` abgeleitet. Zusätzlich leiten sich die Klassen `Priority` und `TransitionLabel` von `GraphicalObject` ab, d.h. alle Objekte die visualisierbar sind. Jedes `GraphicalObject` besitzt einen eindeutigen Bezeichner (Englisch:*id* oder *identifier*), der automatisch vergeben oder manuell bestimmt wird. Die bisherigen Zusammenhänge sind in Abbildung 5.2 verdeutlicht.

Weitere Details zur Implementierung der einzelnen Module befinden sich in der Dokumentation des Projektes [22].

5.2. Umgesetzte Ideen

In diesem Abschnitt werden alle umgesetzten Ideen vorgestellt und ihre Implementation erklärt. Die vorgestellten Optimierungen wurden innerhalb des Statechart-Browsers umgesetzt. Aus diesem Grund muss dieser so erweitert werden, dass es möglich ist, Statecharts zu verändern.

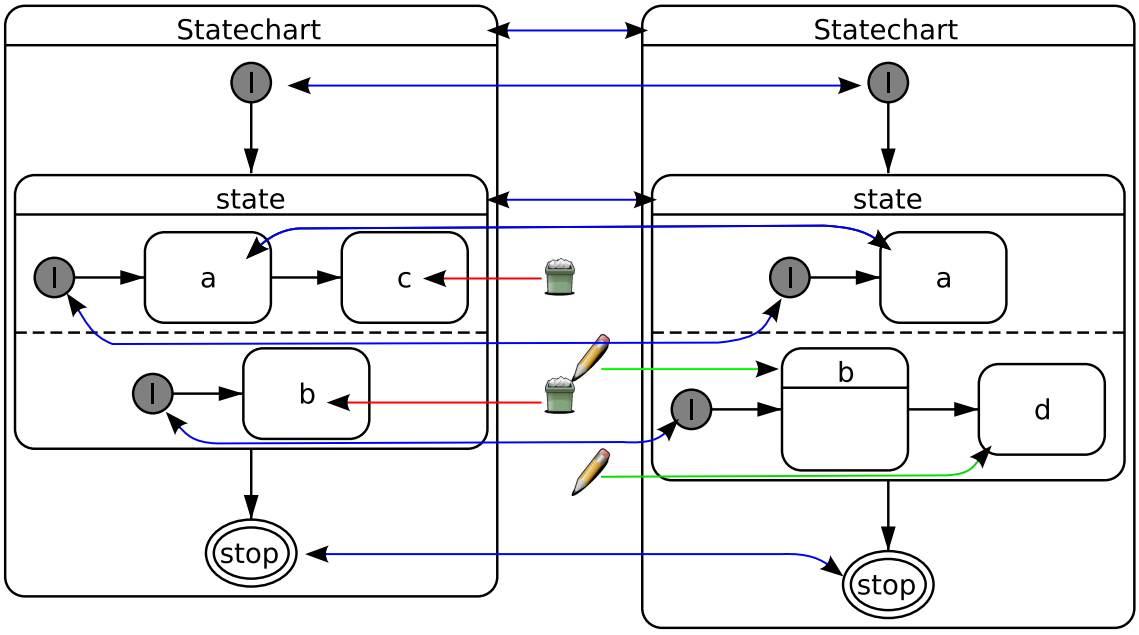


Abbildung 5.3.: herausfinden gleicher Komponenten

5.2.1. Browser-Erweiterungen zur Editierbarkeit

Der Browser wurde, um die Möglichkeit Statecharts zu editieren, erweitert. Dabei wurden zahlreiche neue Methoden hinzugefügt. Eine zentrale Aufgabe ist es vorhandene Statechart zu vergleichen.

Dabei findet eine Zuordnung zwischen den jeweiligen Komponenten des einen und des anderen Statecharts anhand der *ids* statt. Das Ziel ist es, Listen über alle hinzugefügten und entfernten Komponenten zu erhalten. Ein Beispiel hierzu befindet sich in Abbildung 5.3. Anschließend werden alle anzeigenden Komponenten benachrichtigt, damit sie eine Aktualisierung vornehmen können.

Leider nahm die Geschwindigkeit der Darstellung mit der Zunahme der Komponenten immer drastischer ab. Um jedoch Aktionen auf dem Statechart ausführen zu können, darf die Dauer einer Aktualisierung nicht mehr als eine Sekunde betragen. Dieses Ziel wurde durch das Ersetzen des bestehenden Anzeige-Toolkits erreicht. Für den Browser wurden eine Reihe von Schnittstellen definiert, um ihn unabhängiger vom gewählten Toolkit zu machen. Diese Schnittstellen ermöglichen eine Kapselung der Zeichen- und Animationsroutinen von den Aufrufen und Datenstrukturen des Toolkits, mit dessen Hilfe es gelang, die Geschwindigkeit deutlich zu erhöhen. Siehe dazu Abbildung 5.4.

[1]Gemessen wurden die Zeiten die benötigt werden graphische Elemente zu Erstellung und Darzustellen.

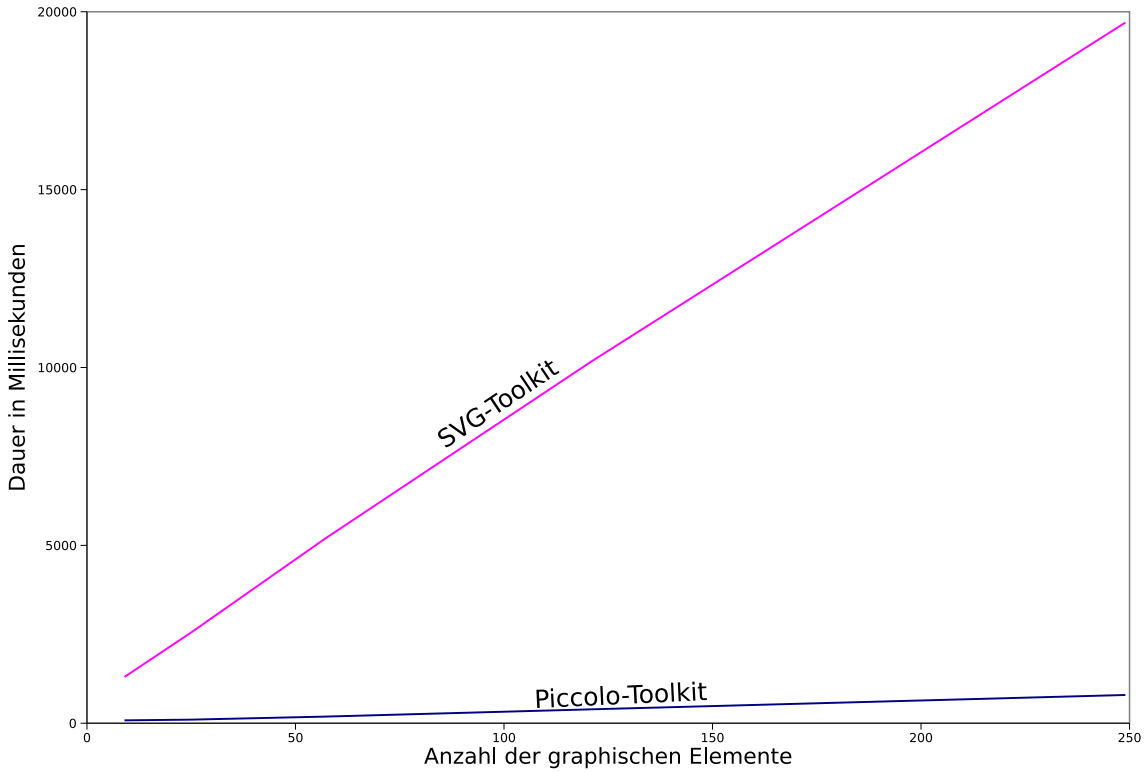


Abbildung 5.4.: Geschwindigkeitsvergleich der Toolkits[1]

5.2.2. Aktionen auf Statecharts

Um die im Kapitel 3.1 vorgestellten Aktionen umzusetzen, wurde zunächst ein Fokus definiert. Er besteht aus folgenden Komponenten:

1. einen Zustand
2. einem weiteren optionalen Zustand
3. einer Transition

Die Komponenten des Fokus sind in den folgenden Auflistungen durch ein ∗ gekennzeichnet. Mit Hilfe dieser drei Komponenten können alle Aktionen beschrieben werden. Die Aktionen lassen sich entweder mit Hilfe von schematischen Abbildungen wie in Abbildungen 3.3 bis 3.5 oder mit Hilfe einer Sprache definieren. Die Sprache, die für die Definition benutzt wurde, ist in Abbildung 5.5 in einer EBNF Form dargestellt. Definition folgender Aktionen für die Implementation:

Folgezustand hinzufügen: Die Aktion fügt zu einem ausgewählten Zustand einen Folgezustand hinzu und verbindet die beiden Zustände mit Hilfe einer Transition. Der Fokus wandert dabei auf den neu erstellten Zustand, siehe Abbildung 3.1(a) und Auflistung 5.1.

⟨*action*⟩ ::= ‘`action`’ ‘`[`’ ⟨*actionid*⟩ ‘`;`’ ⟨*actionname*⟩ ‘`:`’ ⟨*actionkey*⟩ ‘`]`’ ‘`{`’ ⟨*actionrule*⟩* ‘`}`’

⟨*actionname*⟩ ::= ‘`name`’ ‘`=`’ ⟨*string*⟩

⟨*actionkey*⟩ ::= ‘`key`’ ‘`=`’ ⟨*keystroke*⟩

⟨*actionid*⟩ ::= ‘`id`’ ‘`=`’ ⟨*identifier*⟩

⟨*actionrule*⟩ ::= ⟨*focus*⟩ ‘`::=`’ ⟨*stateortransition*⟩+ ‘`;`’

⟨*focus*⟩ ::= ⟨*selectedstate*⟩ (‘`,`’ ⟨*selectedstate*⟩)? (‘`,`’ ⟨*selectedtransition*⟩)?
| ⟨*selectedtransition*⟩

⟨*selectedstate*⟩ ::= ⟨*identifier*⟩ ‘`*`’ ⟨*targ*⟩?

⟨*selectedtransition*⟩ ::= ⟨*identifier*⟩? ‘`->`’ ‘`*`’ ⟨*identifier*⟩ ⟨*ttarg*⟩?

⟨*targ*⟩ ::= ‘`[`’ ⟨*typeargument*⟩ ‘`]`’

⟨*ttarg*⟩ ::= ‘`[`’ ⟨*ttypeargument*⟩ ‘`]`’

⟨*typeargument*⟩ ::= ‘`type`’ ‘`=`’ ⟨*types*⟩

⟨*stateortransition*⟩ ::= ⟨*state*⟩ ‘`;`’ | ⟨*transition*⟩ ‘`;`’ | ‘`||`’ | ‘`...`’

⟨*state*⟩ ::= ⟨*identifier*⟩ ‘`*`’? ⟨*targ*⟩?
| ⟨*identifier*⟩ ‘`*`’? ⟨*modifiers*⟩? ⟨*targ*⟩? ‘`{`’ ⟨*stateortransition*⟩* ‘`}`’

⟨*modifiers*⟩ ::= ‘`.`’ (‘`getParent`’ | ‘`forAllChilds`’)

⟨*types*⟩ ::= ‘`history`’ | ‘`deephistory`’ | ‘`initial`’
| ‘`fork`’ | ‘`join`’ | ‘`junction`’
| ‘`sync`’ | ‘`choice`’ | ‘`dynamicchoice`’
| ‘`suspend`’ | ‘`final`’ | ‘`simple`’ | ‘`composite`’
| ‘`or`’ | ‘`and`’ | ‘`region`’

⟨*transition*⟩ ::= ⟨*selectedident*⟩? ‘`->`’ ‘`*`’? ⟨*selectedident*⟩ ⟨*ttarg*⟩?

⟨*selectedident*⟩ ::= ⟨*identifier*⟩ ‘`*`’?

⟨*ttypeargument*⟩ ::= ‘`type`’ ‘`=`’ ⟨*ttypes*⟩

⟨*ttypes*⟩ ::= ‘`weakabortion`’
| ‘`strongabortion`’
| ‘`normaltermination`’

Abbildung 5.5.: Statechart-Aktions Sprache in EBNF Form

Auflistung 5.1: Aktionsregel zum Hinzufügen von Folgezuständen

```
Action[id=AddState; name="Add State"; keyStroke=Browser.KeyStroke.AddState] {
  A*[type=initial] ::= A->B*[type=initial;];;
  A*[type=choice] ::= A->B*[type=conditional;];;
  A* ::= A->B*;;
}
```

Erzeugen einer Transition: Die Aktion verbindet zwei ausgewählte Zustände mit einer Transition. Ist nur ein Zustand selektiert, wird eine Selbsttransition (engl.:*self loop*) erzeugt, d.h eine Transition in der Quelle und Senke identisch sind. Der Fokus wandert dabei auf die neu erzeugte Transition. Siehe dazu Abbildung 3.5 und Auflistung 5.2.

Auflistung 5.2: Aktionsregel zum Erzeugen einer Transition

```
Action[id=CreateTrans; name="New Transition";
       keyStroke=Browser.KeyStroke.NewTransition] {
  A*    ::= A->*A;;
  A*,B* ::= A->B;;
}
```

Umwandeln eines Zustandes: Die Aktion ändert den Typ eines ausgewählten Zustandes. Ein einfacher Zustand wird in ein OR-State umgewandelt, dabei werden im OR-State ein initialer und ein einfacher Zustand angelegt. Die Zustände werden miteinander verbunden. Handelt es sich bereits um einen OR-State, wird der Zustand zu einem AND-State konvertiert. Der Inhalt des alten OR-States wird in eine neu erzeugte Region des AND-States kopiert. Eine zusätzliche Region, die einen initialen und einen einfachen Zustand enthält, wird erstellt. Die Zustände sind miteinander verbunden. Ist der Zustand ein AND-State, wird der Typ nicht geändert, sondern eine neue Region wird erstellt. Sie enthält einen initialen und einen einfachen Zustand, die miteinander verbunden sind. Vorhandene Regionen werden kopiert. Hierbei bleibt der Fokus auf dem umgewandelten Zustand, siehe Abbildung 3.3 und Auflistung 5.3.

Auflistung 5.3: Aktionsregel zum Umwandeln eines Zustandes

```
Action[id=UpgradeState; name="Upgrade State";
       keyStroke=Browser.KeyStroke.UpgradeState] {
  A*[type=SimpleState] ::= A*{
                             ->a;
                           };;
  A*[type=finalor] ::= A*[type=finaland;]{
                         ...
                       ||
                         ->a;
                     };;
  A*[type=ORState] ::= A*{
                         ...
                       ||
                         ->a;
                     };;
  A*[type=finaland] ::= A*[type=finaland;]{
                         ...
                       ||
                         ->a;
```

```
20                          };;
21   A*[type=ANDState] ::= A*{
22                               ...
23                               ||
24                                ->a;
25                           };;
26 }
```

Entfernen von Komponenten: Die Aktion entfernt die sich gerade im Fokus befindenden Komponenten. Im Falle von hierarchischen Zuständen werden ebenfalls alle enthaltenen Komponenten entfernt. Der Fokus ist nach dieser Aktion leer, siehe Abbildung 3.4 und Auflistung 5.4.

Auflistung 5.4: Aktionsregel zum Entfernen von Komponenten

```
1  Action[id=RemoveObject; name="Remove␣Objects";
2         keyStroke=Browser.KeyStroke.RemoveObjects;
3         options="nolayout"] {
4    A* ::= ...;
5    A*,B* ::= ...;
6    A*,B*,C->*D ::= ...;
7    A->*B ::= ...;
8  }
```

Hinzufügen von History-Zuständen: Die Aktion fügt History-Zustände zu einem ausgewählten Zustand hinzu. Ist der selektierte Zustand ein AND-State, wird zu allen seinen Kindern ein History-Zustand hinzugefügt. Ist der selektierte Zustand eine Region, wird zu allen parallelen Regionen ein History-Zustand hinzugefügt. Ist der selektierte Zustand ein OR-State, wird auch hier ein History-Zustand hinzugefügt, siehe Abbildung 3.7 und Auflistung 5.5.

Auflistung 5.5: Aktionsregel zum Hinzufügen von History-Zuständen

```
1  Action[id=AddHistory; name="Add␣History";
2         keyStroke=Browser.KeyStroke.AddHistory] {
3    A*[type=ANDState] ::= A*.allChilds[type=ANDState;]{
4                            h1[type=history;];
5                          };;
6    A*[type=Region] ::= A*[type=Region;]{
7                          h1[type=history;];
8                        };;
9    A*[type=ORState] ::= A*{
10                         h1[type=history;];
11                       };;
12 }
```

Vertauschen von Quelle und Senke: Die Aktion vertauscht die Quelle und Senke einer selektierten Transition. Der Fokus liegt danach auf der umgedrehten Transition, siehe Abbildung 3.6 und Auflistung 5.6.

Auflistung 5.6: Aktionsregel zum Vertauschen von Quelle und Senke

```
1  Action[id=SwapTransition; name="Swap␣Source␣and␣Target";
2         keyStroke=Browser.KeyStroke.SwapTransition] {
3    A->*B ::= B->*A;;
4  }
```

Hinzufügen von OR-States: Die Aktion erzeugt einen OR-State auf gleicher Ebene eines selektierten Zustandes und verbindet die beiden Zustände. Der OR-State enthält bereits einen initialen und einen einfachen Zustand, die miteinander in Verbindung stehen. Der Fokus liegt danach auf dem neuen OR-State, siehe Abbildung 3.1(b)und Auflistung 5.7.

Auflistung 5.7: Aktionsregel zum Hinzufügen von OR-States

```
1 Action[id=AddMacroState; name="Add␣ORState";
2        keyStroke=Browser.KeyStroke.AddORState] {
3   A* ::= B*{
4             ->a;
5           };
6           A->B;;
7 }
```

Hinzufügen von Suspend-Zuständen: Die Aktion erzeugt einen Suspend-Zustand auf gleicher Ebene eines selektierten Zustandes und verbindet die beiden Zustände miteinander. Der Fokus liegt danach auf der verbindenden Transition, siehe Abbildung 3.8 und Auflistung 5.8.

Auflistung 5.8: Aktionsregel zum Hinzufügen von Suspend-Zuständen

```
1 Action[id=AddSuspend; name="Add␣Suspend";
2        keyStroke=Browser.KeyStroke.AddSuspend] {
3   A* ::= S[type=suspend;];
4           S->*A;;
5 }
```

Hinzufügen von Choice-Zuständen: Die Aktion benötigt zwei selektierte Zustände. Die Zustände werden mit einem Choice-Konstrukt verbunden. Der Fokus liegt danach auf dem neuen Choice-Zustand, um weitere Zustände hinzufügen zu können. Die Aktion gibt es analog für Dynamic-Choice-Zustände und Junction-Zustände, siehe Abbildung 3.9 und Auflistung 5.9.

Auflistung 5.9: Aktionsregel zum Hinzufügen von Choice-Zuständen

```
1 Action[id=AddChoice; name="Add␣Choice";
2        keyStroke=Browser.KeyStroke.AddChoice] {
3   A*,B* ::= C[type=choice;];
4             A->C[type=conditional;];
5             C->*B[type=conditional;];;
6   A->*B ::= C[type=choice;];
7             A->C[type=conditional;];
8             C->B[type=conditional;];;
9 }
```

Die vorgestellten Aktionen stellen nur einen Ausschnitt dar. Mit Hilfe der dargestellten Sprache lassen sich noch weit mehr Aktionen definieren. Die eingeführte Sprache dient nicht nur zur Definition der Aktionen, sondern auch zur Laufzeitinterpretation. Dafür wurde der Browser um einen Parser und einen Interpreter für die Regeln erweitert. Der Parser wurde dabei mit Hilfe eines Parser-Generators erzeugt. Die erstellte Grammatik, ein Überblick über Parser-Generatoren und deren Funktionsweise, befindet sich im Anhang E und Anhang B.

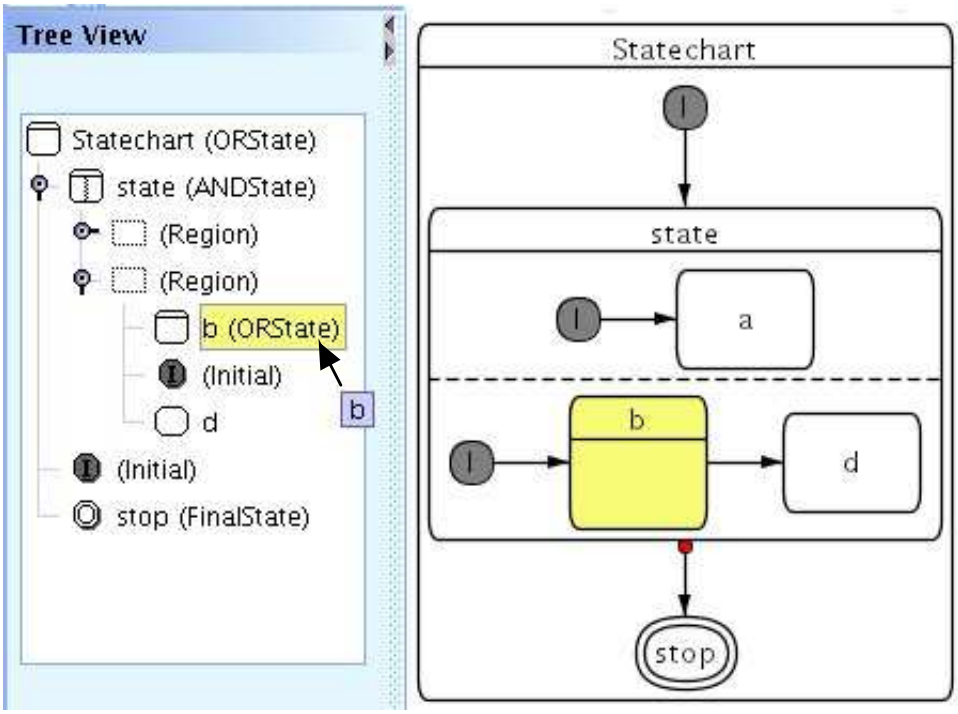


Abbildung 5.6.: Selektion über die Baumansicht des Browsers

5.2.3. Selektieren von Komponenten

Um den im vorherigen Abschnitt definierten Fokus zu bestimmen, muss ein Selektionsmechanismus implementiert werden. Im Browser war bereits eine Auswahl von Komponenten über die Baumansicht des Statecharts möglich, siehe Abbildung 5.6. Über die Baumansicht ist nur die Auswahl von Zuständen gegeben, jedoch nicht die Auswahl von Transitionen. Es wurden folgende Mechanismen zur Auswahl implementiert:

1. Mit Hilfe der Maus und Klicken in die Zeichenfläche können Transitionen und Zustände selektiert werden.

2. Mit Hilfe der Tastatur, insbesondere der Cursortasten, ist eine Navigation durch das Statechart möglich. Hierbei können Transitionen und Zustände selektiert werden.

Um über die Zeichenfläche Zustände und Transitionen auswählen zu können, wurde ein Verfahren implementiert, das feststellt, welche Komponente sich unter dem Mauszeiger befindet. Somit können zwei Zustände und eine Transition selektiert werden, was dem Fokus entspricht.

Zusätzlich wurde das im Abschnitt 3.3.1 vorgestellte Konzept zur Navigation umgesetzt. Die dort vorgestellten Tastenkombinationen, die nicht auf der Tastatur zu finden sind (z. B. Search), können im Browser konfiguriert werden. Siehe dazu Anhang D.

5.2.4. Die Statechart-Beschreibungssprache KIT

In Kapitel 3.4 wurde neben den umgesetzten Vorschlägen zusätzlich das Editieren mit Hilfe einer Beschreibungssprache vorgeschlagen. Die im Rahmen dieser Arbeit entwickelte Statechart-Beschreibungssprache wird *KIT* (***KI****el statechart extension of do****T***) genannt. Die komplette Grammatik der erstellten Beschreibungssprache befindet sich im Anhang E.

Funktionsweise von KIT

In diesem Abschnitt werden, um die Funktionsweise von *KIT* zu erläutern, Teile der Grammatik näher vorgestellt. Die Erstellung von Zuständen wird in Abbildung 5.7 und die der Transitionen in Abbildung 5.8 beschrieben. Folgende Einschränkung geht dabei nicht aus der Grammatik hervor: Die angegebenen Bezeichner, auch *identifier* genannt, sind in einer *KIT*-Datei eindeutig zu wählen.

Eine sehr praktische Eigenschaft ist das implizite Erzeugen von Zuständen. Beispielsweise betrachte man die Deklaration einer Transition in Auflistung 5.10. Die Transition verläuft von einem Zustand mit der "A" zu einem anderen Zustand mit der "f1". Sind die Zustände zum Zeitpunkt des Einlesens dieser Zeile noch nicht erstellt, werden sie implizit als einfache Zustände durch die Auflistung 5.10 erzeugt.

Auflistung 5.10: *KIT* Code zur Erzeugung einer Transition

```
1    A->f1[type=sa;label="A";];
```

Mit Hilfe eines *identifiers* lässt sich der Typ des Zustandes ändern. Um Startzustände besonders einfach notieren zu können, wurde hierfür eine besondere Notation eingeführt: Das Auslassen des ersten *identifiers* bei der Transitionsdeklaration erzeugt einen Startzustand, der mit dem angegeben Zustand verbunden ist. Daher ist Auflistung 5.11 identisch mit Auflistung 5.12.

Auflistung 5.11: *KIT* Code für einen initialState mit Transition (implizit))

```
1    ->A;
```

Auflistung 5.12: *KIT* Code für einen initialState mit Transition (explizit))

```
1    i1[type=initial];
2    i1->A;
```

Implementierung in KIEL

Zur Einbettung von *KIT* in *KIEL* wurde zum Einen ein *Plugin* für das FileInterface entwickelt, mit dem es möglich ist, Dateien einzulesen und abzuspeichern und zum Anderen wurde ein Texteditor implementiert. Dadurch besteht die Möglichkeit, parallel die textuelle und visuelle Repräsentation eines Statecharts im Auge zu behalten. Das wird in Abbildung 5.9 verdeutlicht.

Die Editiervorgänge beeinflussen sich jeweils gegenseitig: Wird im Texteditor die *KIT*-Repräsentation geändert, aktualisiert sich die graphische Sicht automatisch.

⟨*state*⟩ ::= ⟨*identifier*⟩ ⟨*sargument*⟩? ‘`;`’
| ⟨*identifier*⟩ ⟨*sargument*⟩? ‘`{`’ ⟨*region*⟩? ⟨*element*⟩* ‘`}`’ ‘`;`’

⟨*region*⟩ ::= ‘`<`’ ⟨*identifier*⟩ ‘`>`’ ⟨*rargument*⟩?

⟨*element*⟩ ::= ⟨*state*⟩
| ⟨*transition*⟩
| ‘`||`’ ⟨*region*⟩?

⟨*rargument*⟩ ::= ‘`[`’ ⟨*regionargument*⟩+ ‘`]`’

⟨*sargument*⟩ ::= ‘`[`’ ⟨*stateargument*⟩+ ‘`]`’

⟨*regionargument*⟩ ::= ‘`label`’ ‘`=`’ ⟨*string*⟩ ‘`;`’
| ⟨*event*⟩ ‘`;`’

⟨*stateargument*⟩ ::= ‘`label`’ ‘`=`’ ⟨*string*⟩ ‘`;`’
| ‘`type`’ ‘`=`’ ⟨*pseudo*⟩ ‘`;`’
| ⟨*event*⟩ ‘`;`’
| ⟨*entry*⟩ ‘`=`’ ⟨*string*⟩ ‘`;`’
| ⟨*do*⟩ ‘`=`’ ⟨*string*⟩ ‘`;`’
| ⟨*exit*⟩ ‘`=`’ ⟨*string*⟩ ‘`;`’

⟨*event*⟩ ::= ‘`localEvent`’ ‘`=`’ ‘`"`’ ⟨*declaration*⟩ ‘`"`’

⟨*pseudo*⟩ ::= ⟨*history*⟩
| ⟨*deephistory*⟩
| ⟨*initial*⟩
| ⟨*fork*⟩
| ⟨*join*⟩
| ⟨*junction*⟩
| ⟨*sync*⟩
| ⟨*choice*⟩
| ⟨*dynamicchoice*⟩
| ⟨*suspend*⟩
| ⟨*final*⟩

Abbildung 5.7.: Grammatik zum Erstellen von Zuständen mit *KIT*

⟨*transition*⟩ ::= ⟨*identifier*⟩? '`->`' ⟨*identifier*⟩ ⟨*targument*⟩? '`;`'

⟨*targument*⟩ ::= '`[`' ⟨*transargument*⟩+ '`]`'

⟨*transargument*⟩ ::= '`label`' '`=`' ⟨*string*⟩ '`;`'
| '`type`' '`=`' ⟨*transtype*⟩ '`;`'
| '`priority`' '`=`' ⟨*int*⟩ '`;`'

⟨*transtype*⟩ ::= ⟨*weakabortion*⟩
| ⟨*strongabortion*⟩
| ⟨*normaltermination*⟩

Abbildung 5.8.: Grammatik zum Erstellen von Transitionen mit *KIT*

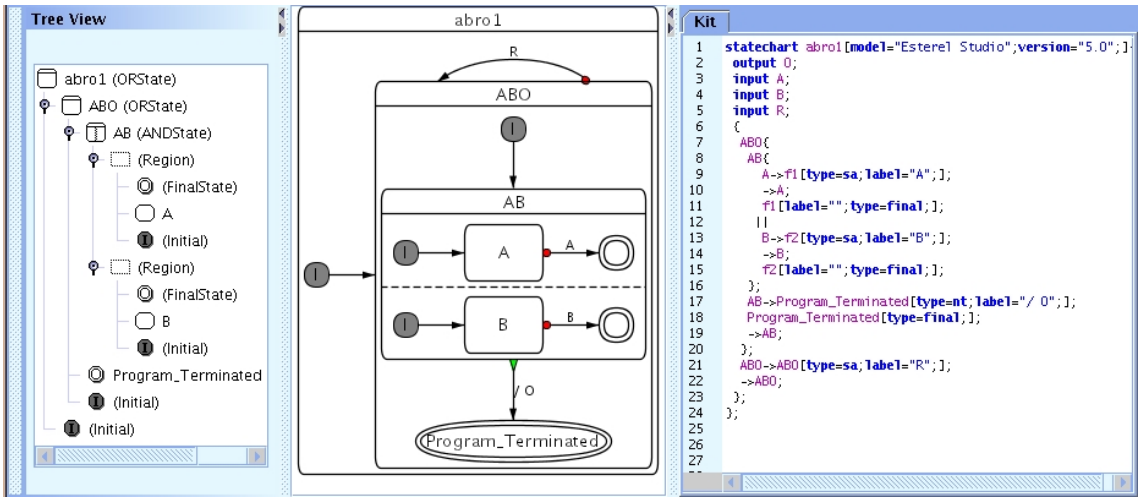
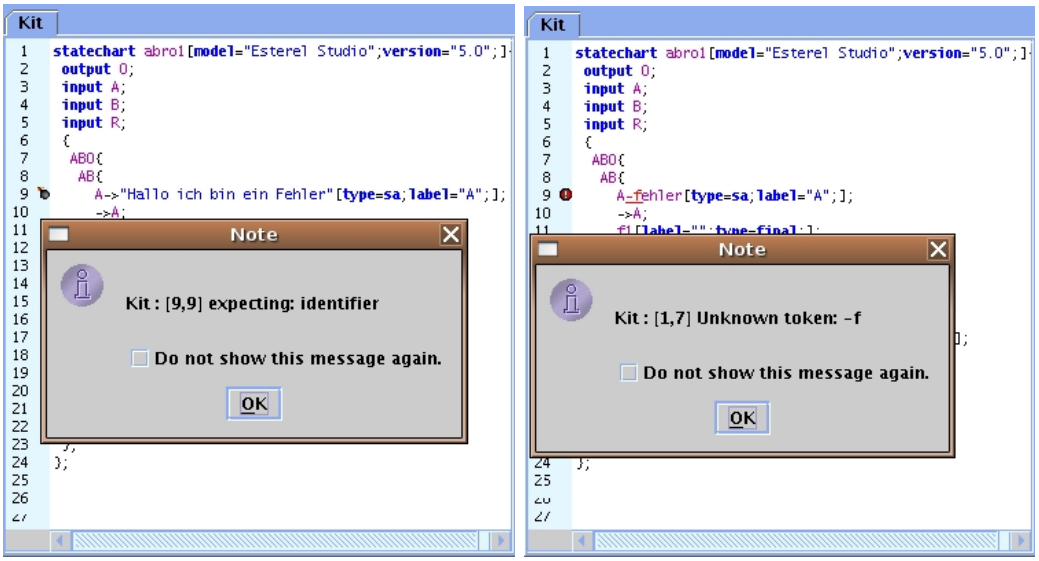


Abbildung 5.9.: verschiedene Ansichten auf ein Statechart



(a) Parser Fehler (b) Lexer Fehler

Abbildung 5.10.: Fehler in der *KIT*-Struktur

Wird andersherum mit Hilfe von Tastenkürzeln die graphische Repräsentation geändert, kommt es zu einer Aktualisierung der *KIT*-Darstellung. Der Aktualisierungsmechanismus findet über fest definierte Schnittstellen statt und ermöglicht dadurch die Einbindung anderer textueller Beschreibungen.

Der implementierte Text-Editor für *KIT* verfügt über folgende Eigenschaften: *Syntax-Highlighting*, d.h. farbliches Markieren von Schlüsselworten und anderen syntaktischen Strukturen und automatisches Testen der *KIT*-Struktur. Wird ein Fehler in der *KIT* Struktur gefunden, kommt es zur Markierung der entsprechenden Stelle und eine Hilfe zur Lösung des Problems wird verfügbar. Das ist in Abbildung 5.10 verdeutlicht.

Für das Aktualisieren der graphischen Darstellung wurde folgendes Verfahren umgesetzt: Eine Aktualisierung findet statt, nachdem x ms nicht mehr im Editor geändert worden ist. Der genaue Wert wird dabei über die Konfigurationsdatei des Browsers festgelegt. Siehe dazu Anhang D.

Nachdem alle umgesetzten Eigenschaften erläutert wurden folgt im nächsten Kapitel eine Zusammenfassung und ein Ausblick auf mögliche weitere Arbeiten.

6. Zusammenfassung und Ausblick

Die vorliegende Arbeit beschreibt ein Modul für das Projekt *KIEL*, das es ermöglicht, ein Statechart mit Hilfe von Maßnahmen, die die Struktur des Statecharts verändern, zu editieren. Eine graphische Sicht auf das Statechart wird durch den Einsatz eines automatischen Layouts erzeugt. Die unterschiedlichen vorgestellten Ansätze sollen die Komplexität des Editiervorgangs optimieren. Aus diesem Grund wurde der Modellierungsprozess in syntaxgerichteten Editoren analysiert und Möglichkeiten zur Optimierung aufgezeigt. Die umgesetzten Verfahren wurden erläutert und an einem Fallbeispiel verdeutlicht.

Die implementierten Verfahren verbessern die Komplexität des Editiervorgangs deutlich. Im Fallbeispiel sind im Vergleich mit dem Browser im Editor mehr als doppelt so viele Schritte nötig, um dasselbe Modell zu erzeugen. Im Vergleich mit der *KIT*-Sprache werden sogar 2,5 mal so viele Schritte benötigt. Bezüglich der Geschwindigkeit konnten im Fallbeispiel nur geringe Verbesserungen gemessen werden, unter Berücksichtigung der Tippgeschwindigkeit konnte jedoch ein erhebliches Verbesserungspotential aufgezeigt werden. Zusätzlich wurde eine bessere Qualität des Editiervorgangs, durch die Anwendung des automatischen Layouts und die Gegenüberstellung von textueller und graphischer Sicht auf ein Statechart, festgestellt. Darüberhinaus gibt es weitere Ansätze, um die Modellierungsarbeit von Satecharts noch weiter zu optimieren. An dieser Stelle werden daher mögliche Erweiterungen vorgestellt.

Erweiterung der Aktions-Sprache: Mit Hilfe der in Kapitel 5 vorgestellten Aktions-Sprache lassen sich komfortable Aktionen auf der Statechart-Datenstruktur beschreiben. Sie ist jedoch auf Änderungen, die sich auf ein oder zwei Zustände beziehen, beschränkt. Es ist nicht möglich die Umgebung der Zustände, auf die sich die Änderung beziehen soll, zu beschreiben. Daher können nicht alle denkbaren Aktionen mit dieser Sprache erklärt werden.

Verbesserung des internen Editors: Der *KIT* Editor ist ein sehr einfacher Texteditor, der mit zusätzlichen Tastaturkommandos für einen Geschwindigkeitszuwachs beim Editieren von *KIT* sorgen könnte.

Zusätzliches graphisches Editieren im Browser: Um einen optimalen Editierprozess zu erhalten, sollte der Browser um die Möglichkeit des graphischen Editierens erweitert werden. Dadurch erhöht sich die Flexibilität des Editiervorgangs.

Geschwindigkeit beim Editieren verbessern: Trotz der deutlich geringeren Anzahl von nötigen Schritten ist die Gesamtgeschwindigkeit zum Erzeugen eines Mo-

dells nur geringfügig verbessert worden, deshalb sollten weitere Untersuchungen zur Optimierung des Vorgangs angesetzt werden.

7. Literaturverzeichnis

[1] Java compiler compiler - the java parser generator. URL `https://javacc.dev.java.net/`.

[2] Charles André. Semantics of S.S.M (Safe State Machine). Technischer bericht, Esterel Technologies, Sophia-Antipolis, France, April 2003. `http://www.esterel-technologies.com`.

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

[4] Rodolfo Castelló, Rym Mili und Ioannis G. Tollis. A Framework for the Static and Interactive Visualization for Statecharts. *Journal of Graph Algorithms and Applications*, 6(3):313–351, 2002.

[5] Esterel Technologies. Esterel studio. `http://www.esterel-technologies.com/products/esterel-studio/overview.html`.

[6] Thomas Huining Feng. Case study: Consistency problems in a UML model of a chat room. In *Sixth International Conference on the Unified Modelling Language (UML 2003), Workshop on Consistency Problems in UML-based Software Development II*, Oktober 2003. San Francisco, USA.

[7] Thomas Huining Feng. An extended semantics for a Statechart Virtual Machine. In A. Bruzzone und Mhamed Itmi, editors, *Summer Computer Simulation Conference (SCSC 2003), Student Workshop*, Seiten S147 – S166. The Society for Computer Modelling and Simulation, Juli 2003. Montréal, Canada.

[8] Etienne Gagnon. SableCC: Java parser generator. `http://sablecc.org/`.

[9] Etienne Gagnon. Sablecc, an object-oriented compiler framework. Diplomarbeit, McGill University, Montreal, 1998. URL `http://sablecc.org/documentation.html`.

[10] GraphViz. Graphviz—graph drawing tools. `http://graphviz.org/`.

[11] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, Juni 1987.

[12] F. Maraninchi und Y. Rémond. Argos: an automaton-based synchronous language. *Computer Languages*, (27):61–92, 2001.

[13] Franco Mazzanti. *UMG User Guide (Version 2.5)*. Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI), Pisa, Italy, 2003.

[14] Object Management Group. Unified Modeling Lanugage—UML Resource Page. `http://www.uml.org`.

[15] *Human-Usable Textual Notation (HUTN) Specification.* Object Management Group, Inc., August 2004. `http://www.omg.org/technology/documents/formal/hutn.htm`.

[16] Terence Parr. Antlr, another tool for language recognition. URL `http://www.antlr.org/`.

[17] Piccolo. Piccolo Homepage, 2005. `http://www.cs.umd.edu/hcil/piccolo`.

[18] Rational Software. Rational rose technical developer. `http://www-306.ibm.com/software/awdtools/developer/technical/`.

[19] Jason Robbins und David Redmiles. Cognitive support, uml adherence, and xmi interchange in argo/uml. *Journal of Information and Software Technology. Special issue: The Best of COSET '99*, 42(2):79–89, 2000. `http://citeseer.nj.nec.com/robbins99cognitive.html`.

[20] F. Mazzanti S. Gnesi. On the fly model checking of communicating uml state machines. In *Second ACIS International Conference on Software Engineering Research Management and Applications (SERA2004)*, 2004.

[21] Stefan Schiffer. *Visuelle Programmierung. Grundlagen und Einsatzmöglichkeiten.* Addison-Wesley, 1998.

[22] The KIEL Project. Project API Documentation, 2004. `http://www.informatik.uni-kiel.de/~rt-kiel/kiel/kiel/doc/`.

[23] The KIEL Project (Kiel Integrated Environment for Layout). Project Homepage, 2006. `http://www.informatik.uni-kiel.de/rtsys/kiel/`.

[24] The Mathworks. Stateflow—Design and simulate event-driven systems. `http://www.mathworks.com/products/stateflow/`.

[25] The University of Queensland. DSTC Pegamento project. `http://www.dstc.edu.au/Research/Projects/Pegamento/hutn/`.

[26] W3C. State Chart XML (SCXML): State Machine Notation for Control Abstraction, Februar 2007. `http://www.w3.org/TR/scxml/`.

[27] Wikipedia. Wikipedia, the free encyclopedia. URL `http://en.wikipedia.org/`.

[28] Mirko Wischer. Ein FileInterface für das KIEL Projekt. Internship Report, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, 2005.

[29] World Wide Web Consortium (W3C) . XML Homepage. `http://www.w3.org/XML/`.

A. Überblick über vorhandene Statechart-Beschreibungssprachen

In diesem Abschnitt werden einige Statechart-Beschreibungssprachen, die im Rahmen dieser Arbeit betrachtet wurden,vorgestellt. Die Vorstellung der Sprachen erfolgt nach folgenden Kriterien:

(i) Verständnis
Wie gut ist die Sprache zu verstehen auch ohne eine Anleitung konsultiert zu haben?

(ii) Editierbarkeit
Wie einfach lässt sich die Sprache editieren? Gibt es besondere Eigenschaften, die beim Editieren beachtet werden müssen?

(iii) Länge des Textes
Wie viel Text muss geschrieben werden, um ein Statechart zu beschreiben? Gibt es viele zusätzliche Angaben neben der Statechart-Topologie?

(iv) Struktur
Lässt sich die Struktur des Statecharts auf die Struktur der Beschreibungssprache übertragen?

Die Vorstellung beschränkt sich auf Sprachen, die den Kriterien nicht wiedersprechen. So beschreiben die Dateiformate von verschiedenen Modellierungswerkzeugen ebenfalls Statecharts. Da diese Dateiformate allein zur Datenhaltung geschaffen worden sind, ist der Umfang an Zusatzinformationen so groß, dass schon ein einfaches Statechart aus mehreren hundert Zeilen Beschreibung besteht. Um eine Vorstellung von den verschiedenen Sprachen zu bekommen, soll das einfache Beispiel aus Kapitel 2 umgesetzt werden, siehe Abbildung A.1. Nachfolgende Sprachen wurden betrachtet:

SCXML: Die Abkürzung SCXML [26] steht für **S**tate**C**hart **XML**. Das W3C (**W**orld **W**ide **W**eb **C**onsortium) hat im Juli 2005 SCXML als einen Entwurf vorgestellt. Die Sprache wird vom W3C folgendermaßen beschrieben:

> „It is intendet to be a general-purpose state machine language that can be used in multiple contexts, ...“ [26]

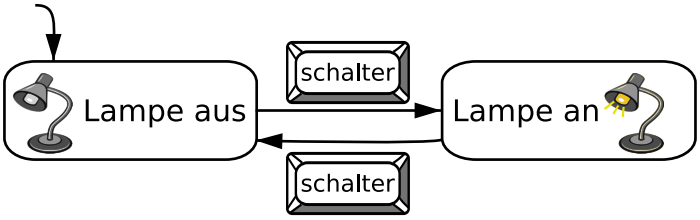


Abbildung A.1.: einfaches Statechart

Das Beispielchart ist in Auflistung A.1 zu finden.

Auflistung A.1: Beispiel im SCXML-Format

```
1  <?xml version="1.0" encoding="us-ascii"?>
2  <scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml">
3    <initial>
4      <transition>
5        <target next="S11"/>
6      </transition>
7    </initial>
8
9    <state id="Lampe aus">
10     <transition event="schalter">
11       <target next="Lampe an"/>
12     </transition>
13   </state>
14
15   <state id="Lampe an">
16     <transition event="schalter">
17       <target next="Lampe aus"/>
18     </transition>
19   </state>
20 </scxml>
```

(i) Verständnis
Das Format basiert auf XML [29] und besitzt dadurch eine bekannte Darstellung. Hierarchie und Parallelität sind einfach zu erkennen, Transitionen werden innerhalb der Zustände definiert.

(ii) Editierbarkeit
Das Format besitzt das Problem, was alle XML Formate haben: Auf Grund der benutzten *Tags* und besonders der hierarchischen Abhängigkeiten einzelner *Tags* ist die Darstellung zu lang und gleichermaßen umständlich manuell zu editieren.

(iii) Länge des Textes
In diesem Format wird nichts implizit definiert, daher müssen alle Angaben explizit gemacht werden. Dieses wirkt sich ungünstig auf die Länge der Darstellung aus.

(iv) Struktur
Alle Kinder eines Zustandes müssen innerhalb der Zustandsdefinition stehen, parallele Zustände werden auf derselben Ebene definiert. Daher ist die Struktur des Statecharts gut nachvollziehbar.

SVM: Der Name SVM [7] ist eine Abkürzung für **S**tatechart **V**irtual **M**achine. Es handelt sich dabei, um einen von Huining Feng entwickelten Simulator für Statecharts. Um Statecharts zu repräsentieren, wird eine textuelle Beschreibung, die hier vorgestellt werden soll, genutzt. Das Beispielchart ist in Auflistung A.2 zu finden.

Auflistung A.2: Beispiel im SVM-Format

```
1  STATECHART:
2    I [DS]
3    Lampe_An
4    Lampe_Aus
5
6  TRANSITION:
7    S: I
8    N: Lampe_Aus
9
10 TRANSITION:
11   S: Lampe_Aus
12   N: Lampe_An
13   E: schalter
14
15 TRANSITION:
16   S: Lampe_An
17   N: Lampe_Aus
18   E: schalter
```

(i) Verständnis
Zustände und Transitionen werden in getrennten Abschnitten definiert. Hierarchie und Parallelität wird mit Hilfe von Einrückungen erzeugt. Diese ist sehr fehleranfällig, denn ein Leerzeichen zu viel bedeutet, dass ein Zustand sich eine Hierarchieebene zu tief befindet. Die Benutzung der Tab -Taste ist dadurch nicht zu empfehlen, da diese zu vielen Problemen führen kann. Für die Transitionsdefinition müssen die Zustände inklusive ihrer Hierarchie angegeben werden. Auch die Symbole, die für die Definition von Eigenschaften genutzt werden, um bspw. *Trigger* anzugeben, sind nicht selbsterklärend.

(ii) Editierbarkeit
Der Mechanismus zur Erzeugung der Hierarchie ist sowohl beim Lesen als auch beim Schreiben problematisch. Durch die getrennte Definition von Zuständen und Transitionen muss beim Editieren gesprungen werden, was den Editiervorgang umständlich macht.

(iii) Länge des Textes
Alle Transitionen und Zustände müssen getrennt definiert werden, da viele Eigenschaften in einer neuen Zeile stehen müssen. Aus diesem Grund ist die Darstellung lang.

(iv) Struktur
Die Struktur des Statecharts ist schwer auszumachen, da Transitionen und Zustände in getrennten Bereichen definiert werden.

UMC: *UMC* [20] ist ein von Franco Mazzanti [13] entwickelter Modelchecker, der in der Lage ist Statecharts mit einer kleinen Anzahl von Zuständen zu verifizieren. Ähnlich wie bei SVM wird eine textuelle Beschreibung genutzt, um die Statecharts zu spezifizieren. Diese Beschreibungssprache wird hier betrachtet. Das Beispielchart ist in Auflistung A.3 zu finden.

Auflistung A.3: Beispiel im *UMC*-Format

```
1 Class Lampe is
2 Signals: schalter
3 State top = I,  Lampe_an,  Lampe_aus
4 Transitions:
5  I -> Lampe_an
6  Lampe_an -(schalter)-> Lampe_aus
7  Lampe_aus -(schalter)-> Lampe_an
8 end
9 Object my_lampe : Lampe
```

(i) Verständnis
Auch hier werden Zustände und Transitionen getrennt definiert. Wobei nur hierarchische und parallele Zustände explizit definiert werden müssen. Kinder von hierarchischen Zuständen werden als Kommaseparierte Listen angegeben. Transitionen werden über Listen von Ursprungszuständen und Listen von Zielzuständen angegeben, wobei optional eine Transitionsbeschriftung definiert werden kann. Die Darstellung ist durch die gewählten Symbole einfach zu verstehen.

(ii) Editierbarkeit
Das Format ist relativ gut zu editieren, bis auf die Tatsache, dass Zustände und Transitionen getrennt definiert werden müssen. Wobei zumindest auf die Definition von einfachen Zuständen verzichtet werden kann.

(iii) Länge des Textes
Dadurch, dass einfache Zustände nicht explizit definiert werden müssen, ist die Darstellung kompakter als die vorherigen Formate.

(iv) Struktur
Die Struktur des Statecharts ist schwer auszumachen, da Transitionen und Zustände in getrennten Bereichen definiert werden.

Argos: *Argos* [12] ist eine von Florence Maraninchi entwickelte synchrone Sprache. Das Beispielchart ist in Auflistung A.4 zu finden.

Auflistung A.4: Beispiel im *Argos*-Format

```
1  targos
2  main lampe (schalter) ()
3
4  process lampe (ischalter) () {
5    controller{
6      init i
7
8      Lampe_an {}
9      -> Lampe_aus with ischalter;
10
```

```
11      Lampe_aus {}
12      -> Lampe_an with ischalter;
13    }
14  }
15  endtargos
```

(i) Verständnis
Bei diesem Format werden Zustände und Transitionen nicht getrennt definiert. Parallelität wird über den Parallel-Operator "||" gekennzeichnet. Hierarchie spiegelt sich in Form von Verschachtelungen wider. Das erlaubt eine einfache Lesbarkeit.

(ii) Editierbarkeit
Zustände werden durch ihren Namen angegeben und können innerhalb der folgenden geschweiften Klammern genauer spezifiziert werden. Nach Definition eines Zustandes können ausgehende Transitionen definiert werden. Diese werden durch einen Pfeil eingeleitet. Das Format ist gut zu editieren, da Transitionen und Zustände nicht getrennt definiert werden müssen und nach jedem Zustand seine ausgehenden Transitionen beschrieben werden.

(iii) Länge des Textes
Die benutzten Schlüsselworte wie `targos`, `process` oder `controller` blähen das Format zwar etwas auf, sorgen aber für eine gute Lesbarkeit.

(iv) Struktur
Die Struktur des Statecharts spiegelt sich direkt in der Darstellung wider, da Hierarchie und Parallelität sich einfach erkennen lassen.

Ein Vergleich aller vorgestellten Sprachen befindet sich in Tabelle A.1.

	SCXML	**SVM**	***UMC***	***Argos***
Verständnis	+	−	+/−	+
Editierbarkeit	−	−	−	+/−
Länge	−	−	+/−	+/−
Struktur	+	−	+/−	+

Tabelle A.1.: Beschreibungssprachen im Vergleich[1]

[1]Kriterium erfüllt: +, Kriterium teilweise erfüllt: +/−, Kriterium nicht erfüllt: −

B. Parser-Generatoren

In diesem Abschnitt soll ein kurzer Überblick über das Themengebiet der Parser-Generatoren gegeben werden. Zusätzlich wird der in dieser Arbeit gewählte Parser-Generator vorgestellt. Der Begriff Parser-Generator wird in der Wikipedia [27] folgendermaßen definiert:

> „A compiler-compiler or parser generator is a utility for generating the source code of a parser, interpreter or compiler from an annotated language description in the form of a grammar (usually in BNF) plus code that is associated with each of the rules of the grammar that should be executed when these rules are applied by the parser. These pieces of code are sometimes referred to as semantic action routines since they define the semantics of the syntactic structure that is analysed by the parser. Depending upon the type of parser that should be generated, these routines may construct a parse tree (or AST), or generate executable code directly."

Ein Parser-Generator-Ablauf besteht aus zwei Schritten: Einem Lexer- oder auch *Scanner*-Generator Schritt und dem eigentlichen Parser-Generator Schritt. Für klassische Parser-Generatoren sind daher zwei verschiedene Programmen notwendig. Im Rahmen dieser Arbeit wurde jedoch *Java* eingesetzt, hier existieren hauptsächlich kombinierte Programme, d.h. sie vereinen folgende Schritte:

1. Das in diesem Schritt generierte Programm erzeugt einen Lexer. Ein solches Programm ist für die lexikalische Analyse der Eingabe zuständig. Das bedeutet, dass die Symbole der Sprache erkannt und in so genannte *Tokens* gruppiert werden. *Whitespace* wird in diesem Schritt aus der Eingabe eliminiert.

2. Die Ausgabe des ersten Schrittes ist die Eingabe für den im Parser-Generator-Schritt erzeugten Parser. Dieser Parser fasst die erkannten *Tokens* zu grammatikalischen Sätzen zusammen und stellt diese in einem Parse-Baum dar. Der Baum kann dann durchwandert werden. Der komplette Ablauf wird in Abbildung B.1 dargestellt.

Folgende Parser-Generatoren haben im *Java*-Umfeld die größte Verbreitung:

JavaCC: Der *JavaCC* [1] Parser-Generator ist ein reiner *Java* Parser-Generator. Es handelt sich um einen LL(k) Parser, da er von links nach rechts arbeitet, linksreduktion betreibt und k *Token* vorausschauen kann. Lexer und Parser Definition finden in derselben Datei statt. Auch Aktionen auf dem Parse-Baum werden in der Spezifikation eingebettet. Dadurch erhält man eine Datei, die sowohl die Grammatik der Sprache beschreibt, als auch *Java*-Code für die Einbettung in ein Programm enthält.

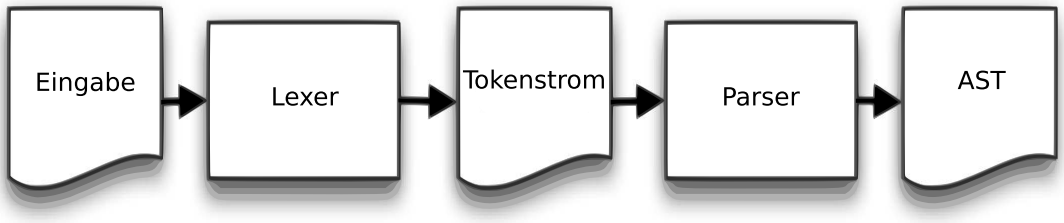


Abbildung B.1.: Ablaufschema eines Lexer-Parser-Durchlaufs

ANTLR: Der *ANTLR* [16] Parser-Generator kann neben *Java*-Code auch *C++*, *Python* und *C#* Parser erzeugen. Er arbeitet ebenfalls nach dem LL(k) Prinzip. Auch hier befindet sich in der Spezifikation eingebetteter *Java*-Code.

SableCC: *SableCC* [8] ist ein reiner *Java* Parser-Generator. Es handelt sich um einen LALR Parser, der eine modifizierte Variante des LR(1) Parsers ist, also ein Parser der von links nach rechts arbeitet, rechtsreduktion betreibt und k *Token* vorausschauen kann. Dabei kann dieser mit mehr Kontext-Freien Grammatiken umgehen als ein einfacher LR Parser, aber weniger als ein allgemeiner LR(1) Parser. Lexer und Parser Definition finden in derselben Datei statt. Im Gegensatz zu den anderen Parser-Generatoren erzeugt *SableCC* eine komplette Klassenhierarchie inklusive einer Reihe von Analyseklassen für den Parse-Baum. Dadurch ist es nicht nötig, *Java*-Code in die Spezifikation einzubetten.

In dieser Arbeit kam *SableCC* zum Einsatz. Zum Einen da es sich um einen LARL Parser handelt und zum Anderen da es nur in diesem Parser-Generator möglich war, auf den eingebetteten *Java*-Code zu verzichten.

C. Bedienungsanleitung

In diesem Abschnitt wird die Bedienung des Browser anhand der zur Verfügung gestellten Tastaturkommandos erläutert. Die Tastensymbole entsprechen dabei den dargestellten Tasten, siehe Tabelle C.1.

Tabelle C.1.: Tastaturkommandos im Browser

Kommando	Aktion
Datei-Operationen	
Ctrl + O	Statechart-Dateien laden
Ctrl + E	Trace-Datei laden
Ctrl + S	Statechart-Dateien speichern
Ctrl + D	*KIT*-Datei speichern
Simulator-Steuerung	
Alt + A	simuliere einen MacroStep
Alt + I	simuliere einen MicroStep
Alt + P	spiele einen kompletten Trace ab
Alt + U	halte den Trace-Ablauf an
Alt + W	spule zum Anfang des Traces zurück
Alt + Q	spule zum Ende des Trace vor
Alt + B	gehe ein Schritt im Trace zurück
Alt + R	gehe ein Schritt im Trace vor
Alt + X	setze die Simulation zurück
Sichtweise	
Ctrl + F	immer automatisch auf die Zeichenfläche skalieren
Ctrl + H	Zoom zurücksetzen
Ctrl + G	Größe auf aktuelle Zeichenfläche skalieren
Navigation im Statechart	
Ctrl + ⏎	schaltet zwischen Text-Editor und Navigation hin und her
↑ , ↓	durchlaufen von Auswahlmöglichkeiten
← , →	durchlaufen von Sequenzen
Page ↑ , Page ↓	Hierarchie auf- und abwandern
⏎	bestätigen der Zustandsauswahl
Home	gehe zu Beginn der Sequenz zurück
Ctrl + ↑ , ↓	Historie durchlaufen
Ctrl + -	nächsten Zustand suchen

Fortsetzung folgt auf der nächsten Seite

Tabelle C.1 - Fortsetzung

Kommando	Aktion
Ctrl + .	nächste Transition suchen
Ctrl + ,	nächsten Pseudo-Zustand suchen
Ctrl + W	einen Zustand oder Transition suchen
Änderungen am Statechart	
Ctrl + I	fügt einen einfachen Folgezustand hinzu (Schema 3.1(a))
Ctrl + J	fügt einen hierarchischen Folgezustand hinzu (Schema 3.1(b))
Ctrl + K	fügt ein Choice-Konstrukt hinzu (Schema 3.9)
Ctrl + L	fügt ein Dynamic-Choice-Konstrukt hinzu (Schema 3.9)
Ctrl + M	fügt ein History-Zustand hinzu (Schema 3.7)
Ctrl + N	fügt ein Suspend-Zustand hinzu (Schema 3.8)
Ctrl + P	ändert den Zustandstyp von einem einfachen Zustand zu einem hierarchischen Zustand und bei nochmaliger Betätigung zu einem parallelen Zustand (Schema 3.3)
Ctrl + R	erzeugt eine Transition zwischen zwei ausgewählten Zuständen (Schema 3.5)
Ctrl + S	vertauscht Quelle und Senke einer Transition (Schema 3.6)
Ctrl + T	entfernt die selektierten Komponenten (Schema 3.4)

D. Konfiguration

In diesem Abschnitt werden die Konfigurationsmöglichkeiten des Browsers vorgestellt. Dazu werden die vorgegebenen Schlüssel mit ihren Vorgabewerten präsentiert und die akzeptierten Werte beschrieben. Siehe dazu Tabelle D.1.

Tabelle D.1.: Schlüssel-Wert-Paare für die Browser Konfiguration

Schlüssel	**Werte**
Beliebige Zeichenketten	
`Browser.Title`	Vorgabewert: `Kiel Statechart Browser 2.0` beschreibt den Titel des Programms, der dem `KielFrame` mitgeteilt wird.
Wahrheitswerte	
`Browser.ShowInterface`	Vorgabewert: `false` gibt an, ob das Interface als Tooltip in der Zeichenfläche angezeigt werden soll.
`Browser.ShowTooltip`	Vorgabewert: `true` gibt an, ob Zustände und Transitionen einen Tooltip anzeigen sollen.
`Browser.AnimateConfigInMicroSteps`	Vorgabewert: `false` gibt an, ob während einer MicroStep-Animation die Konfigurationen wechseln sollen.
`Browser.AlwaysStepThrough`	Vorgabewert: `true` gibt an, ob ein MacroStep durch eine Animation von MicroSteps dargestellt wird.
`Browser.Animation`	Vorgabewert: `true` gibt an, ob zwischen zwei Ansichten eine Animation stattfinden soll.
Ganzzahlige Werte	
`Browser.LogLevel`	Vorgabewert: `2` Erwartet wird ein Wert zwischen `0,...,10`. Dieser Wert gibt in Kombination mit Browser.LogCompare an, welche Mitteilungen des Browsers im Browser-Log erscheinen sollen.
`Animation.Duration`	Vorgabewert: `750` Dauer der Animation in Millisekunden.

Fortsetzung folgt auf der nächsten Seite

Tabelle D.1 - Fortsetzung

Schlüssel	Werte
`MicroStep.SleepTime`	Vorgabewert: `100` Dauer der MicroStep-Ansicht in Millisekunden, wenn über alle MicroSteps animiert wird.
`KitEditor.FontSize`	Vorgabewert: 10. Größe der Schrift im eingebetteten Editor.
`ParserThread.Threshold`	Vorgabewert: `1000` Dauer in Millisekunden, die verstreichen muss, ohne dass neuen Text im Editor eingegeben werden, um eine Aktualisierung der Zeichenfläche zu erzwingen.
Hexadezimale Farbtripel	
`BrowserCanvas.StringLabelColor`	Vorgabewert: `#ff0000` Farbe einer Transitionsbeschriftung, die nicht ausgewertet werden konnte.
`BrowserTree.MarkNodeColor`	Vorgabewert: `#f8fb78` Farbe eines markierten Zustandes.
`BrowserTree.MarkEdgeColor`	Vorgabewert: `#a8a000` Farbe einer markierten Transition.
`BrowserTables.InputEventColor`	Vorgabewert: `#0000ff` Farbe für Eingabe-Signale.
`BrowserTables.OutputEventColor`	Vorgabewert: `#ff0000` Farbe für Ausgabe-Signale.
`BrowserTables.LocalEventColor`	Vorgabewert: `#808080` Farbe für Lokale-Signale.
`BrowserTables.VariablesColor`	Vorgabewert: `#00ff00` Farbe für Variablen.
CSS konforme Zeichenketten	
`MicroStep.ExecuteTransition`	Vorgabewert: `fill:none;stroke:green;stroke-width:2.0px;` legt die Markierung einer ausgeführten Transition fest.
`MicroStep.TestTransition`	Vorgabewert: `fill:none;stroke:blue;stroke-width:1.0px;` legt die Markierung einer getesteten Transition fest.
`MicroStep.OnInside`	Vorgabewert: `fill:gray;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der eine *OnInside* Aktion ausführt,fest.

Fortsetzung folgt auf der nächsten Seite

Tabelle D.1 - Fortsetzung

Schlüssel	Werte
`MicroStep.OnExit`	Vorgabewert: `fill:green;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der eine *OnExit* Aktion ausführt,fest.
`MicroStep.OnEntry`	Vorgabewert: `fill:blue;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der eine *OnEntry* Aktion ausführt,fest.
`MicroStep.StateSuspended`	Vorgabewert: `fill:yellow;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der gerade suspendiert wird,fest.
`MicroStep.StateActivated`	Vorgabewert: `fill:none;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der gerade betreten wird,fest.
`MicroStep.StateDeactivated`	Vorgabewert: `fill:white;stroke:black;stroke-width:2.0px;` legt die Markierung eines Zustandes, der gerade verlassen wird,fest.
`MicroStep.ActualConfigState`	Vorgabewert: `fill:#ddddff;stroke:black;stroke-width:2.0px;` legt die Markierung der gerade aktuellen Zustände fest.
Vergleichsoperatoren	
`Browser.LogCompare`	Vorgabewert: `<=` mögliche Werte sind `<=`,`==` oder `>=`. Sie geben die Vergleichsoperation, mit der der Browser.LogLevel mit dem jeweiligen Wert der auszugebenden Zeichenkette verglichen wird, an.
Tastenkombinationen	
`Browser.KeyStroke.CountElements`	Vorgabewert: `ctrl B`
`Browser.KeyStroke.NewStatechart`	Vorgabewert: `ctrl C`
`Browser.KeyStroke.SaveKit`	Vorgabewert: `ctrl D`
`Browser.KeyStroke.LoadTrace`	Vorgabewert: `ctrl E`
`Browser.KeyStroke.AlwaysFitScreen`	Vorgabewert: `ctrl F`
`Browser.KeyStroke.FitScreen`	Vorgabewert:`ctrl G`

Fortsetzung folgt auf der nächsten Seite

Tabelle D.1 - Fortsetzung

Schlüssel	Werte
Browser.KeyStroke.ResetZoom	Vorgabewert: ctrl H
Browser.KeyStroke.AddState	Vorgabewert:ctrl I
Browser.KeyStroke.AddORState	Vorgabewert: ctrl J
Browser.KeyStroke.AddChoice	Vorgabewert: ctrl K
Browser.KeyStroke.AddDynamicChoice	Vorgabewert: ctrl L
Browser.KeyStroke.AddHistory	Vorgabewert: ctrl M
Browser.KeyStroke.AddSuspend	Vorgabewert: ctrl N
Browser.KeyStroke.UpgradeState	Vorgabewert: ctrl P
Browser.KeyStroke.NewTransition	Vorgabewert: ctrl R
Browser.KeyStroke.SwapTransition	Vorgabewert: ctrl S
Browser.KeyStroke.RemoveObjects	Vorgabewert: ctrl T
Tastenkürzel: Alt + ...	
Browser.Mnemonic.MicroStep	Vorgabewert: I
Browser.Mnemonic.MacroStep	Vorgabewert:A
Browser.Mnemonic.EditMode	Vorgabewert: D
Browser.Mnemonic.Statechart	Vorgabewert: C
Browser.Mnemonic.View	Vorgabewert: V
Browser.Mnemonic.Transition	Vorgabewert: T
Browser.Mnemonic.State	Vorgabewert: S
Browser.Mnemonic.TraceStepBack	Vorgabewert: B
Browser.Mnemonic.TraceStepForward	Vorgabewert: R
Browser.Mnemonic.TracePlay	Vorgabewert: P
Browser.Mnemonic.TraceStop	Vorgabewert: U
Browser.Mnemonic.TraceRewind	Vorgabewert: W
Browser.Mnemonic.TraceFastForward	Vorgabewert: Q
Browser.Mnemonic.Reset	Vorgabewert: X

E. Spezifikationen der Sprachen im SableCC Format

In diesem Abschnitt werden die Spezifikationen für die erstellten Sprachen im SableCC Format [9] vorgestellt. Aus diesen Quelldateien erzeugt SableCC den Code, der nötig ist, um die Arbeit kompilieren zu können.

E.1. Kit

```
/**
  *   Kit Sablecc file $Id: kit.grammar,v 1.5 2006/05/14 23:35:50 miwi Exp $
  *
  */
Package kiel.util.kit;

Helpers
   all = [0 .. 255];
   digit = ['0' .. '9'];
   nondigit = ['_' + ['#' + ['.' + [['a' .. 'z'] + ['A' .. 'Z']]]]];
   nonzero_digit = ['1' .. '9'];
   decimal_constant = digit+;
   c_char = [all - ['"' + [10 + 13]]];
   c_char_sequence = c_char+;
   cr = 13;
   lf = 10;
   tab = 9;
   not_star = [all - '*'];
   not_cr_lf = [[all -  cr] - lf];
   not_star_slash = [not_star - '/'];
   comma = ',';
   minus = '-';

Tokens
   dot = '.';
   colon = ':';
   semicolon = ';';
   l_par = '(';
   r_par = ')';
   l_bracket = '[';
   r_bracket = ']';
   l_brace = '{';
   r_brace = '}';
   l_reg = '<';
   r_reg = '>';
   parallel = '||';
   equal = '=';
   add = '+';
   mult = '*';
   edge = '->';

   statechart = 'statechart';
   model = 'model';
   version = 'version';
   input = 'input';
   output = 'output';
   var = 'var';
   label = 'label';
   type = 'type';
   history = 'history' | 'hi';
   deephistory = 'deephistory' | 'dh';
   initial = 'initial' | 'in';
   fork = 'fork' | 'fk';
   join = 'join' | 'jn';
   junction = 'junction' | 'jc';
   sync = 'sync' | 'sy';
   choice = 'choice' | 'ch';
```

```
    dynamicchoice = 'dynamicchoice' | 'dc';
    suspend = 'suspend' | 'sd';
    final  = 'final';
    do = 'doActivity' | 'do';
    entry = 'entryActivity' | 'entry';
    exit = 'exitActivity' | 'exit';
    bind = 'bindActivity' | 'bind';
    localevent = 'localEvent';
    localvariable = 'localVariable';
    priority = 'priority';
    weakabortion = 'weakAbortion' | 'wa';
    strongabortion = 'strongAbortion' | 'sa';
    normaltermination = 'normalTermination' | 'nt';
    suspension = 'suspension' | 'sp';
    conditional = 'conditional' | 'co';
    internal = 'internalTransition' | 'it';
    integer = 'integer';
    float = 'float';
    double = 'double';
    boolean = 'boolean';
    combine = 'combine';
    with = 'with';
    pos = 'pos';
    priopos = 'pp';
    labelpos = 'lp';
    width = 'width';
    height = 'height';
    collapsed = 'collapsed';
    true = 'true';
    false = 'false';

    identifier = nondigit (digit | nondigit)*;
    int = '"' minus? digit+ '"';
    dim = '"' digit+ comma digit+ '"';
    number = minus? decimal_constant ('.' digit+)?;
    string = '"' c_char_sequence? '"';
    blank = (cr | lf | tab | ' ')+;
    comment =  '//' not_cr_lf* (cr|lf);

Ignored Tokens
    blank,
    comment;

Productions
    chart = statechart identifier cargument?  l_brace iodeclaration* cstate r_brace semicolon;

    chartargument  = {model} model equal string
                   | {version} version equal string;

    firstcarguments = chartargument semicolon;

    lastcargument = chartargument semicolon?;

    cargument = l_bracket firstcarguments* lastcargument r_bracket;

    iodeclaration = decl_typ declaration semicolon;

    declaration = identifier initialvalue? vtype? ;

    initialvalue = colon equal number;

    vtype = colon var_event_type;

    var_event_type = {boolean} boolean
                   | {double} double
                   | {float} float
                   | {integer} integer
                   | {integer_add} combine integer with add
                   | {integer_mult} combine integer with mult;

    decl_typ = {input} input | {output} output | {var} var;

    cstate = sargument? l_brace region? element* r_brace semicolon;

    state = {simple} identifier sargument? semicolon
          | {composite} identifier cstate;

    element = {state} state | {transition} transition | {new_region} parallel region?;

    region = l_reg identifier r_reg rargument?;

    firstrarguments = regionargument semicolon;

    lastrargument = regionargument semicolon?;

    rargument =  l_bracket firstrarguments* lastrargument r_bracket;

    firstsarguments = stateargument semicolon;

    lastsargument = stateargument semicolon?;
```

```
        sargument =  l_bracket firstsarguments* lastsargument r_bracket;

        t_boolean = {true} true | {false} false;

        sview = {width} width equal int
                | {height} height equal int
                | {pos} pos equal dim
                | {collapsed} collapsed equal t_boolean;

        regionargument = {label} label equal string
                       | {new_event} event
                       | {new_variable} variable
                       | {view} sview;

        event = localevent equal string;

        variable = localvariable equal string;

        pseudo =  {phistory} history
               | {pdeephistory} deephistory
               | {pinitial} initial
               | {pfork} fork
               | {pjoin} join
               | {pjunction} junction
               | {psync} sync
               | {pchoice} choice
               | {pdynamic} dynamicchoice
               | {psuspend} suspend
               | {final} final;

        stateargument = {label} label equal string
                      | {type} type equal pseudo
                      | {new_event} event
                      | {new_variable} variable
                      | {do_action} do equal string
                      | {entry_action} entry equal string
                      | {exit_action} exit equal string
                      | {view} sview;

        transition = {initial} edge [target_state]:identifier targument? semicolon
                   | {other} [source_state]:identifier edge [target_state]:identifier targument? semicolon;

        firsttransarguments = transargument semicolon;

        lasttransargument = transargument semicolon?;

        tview = {path} pos equal string
                | {priopos} priopos equal dim
                | {labelpos} labelpos equal dim;

        targument =  l_bracket firsttransarguments* lasttransargument r_bracket;

        transargument = {label} label equal string
                      | {type} type equal transtype
                      | {priority} priority equal int
                      | {view} tview;


        transtype = {weak} weakabortion
                  | {strong} strongabortion
                  | {normal} normaltermination
                  | {suspension} suspension
                  | {conditional} conditional
                  | {internal_transition} internal;
```

E.2. Deklarationen

```
    /**
      *   Declaration Sablecc file $Id: declaration.grammar,v 1.4 2006/05/22 19:21:03 miwi Exp $
      *
      */
    Package kiel.util.declaration;

    Helpers
       all = [0 .. 127];
       digit = ['0' .. '9'];
       nondigit = ['_' + ['#' + ['.' + [['a' .. 'z'] + ['A' .. 'Z']]]]];
       nonzero_digit = ['1' .. '9'];
       decimal_constant = digit+;
       cr = 13;
       lf = 10;
       tab = 9;
       minus = '-';

    Tokens
```

```
    dot = '.';
    comma = ',';
    colon = ':';
    semicolon = ';';
    equal = '=';
    add = '+';
    mult = '*';

    integer = 'integer';
    int16 = 'int16';
    int8 = 'int8';
    uint32 = 'uint32';
    uint16 = 'uint16';
    uint8 = 'uint8';
    float = 'float';
    double = 'double';
    boolean = 'boolean';
    combine = 'combine';
    with = 'with';

    identifier = nondigit (digit | nondigit)*;
    number = minus? decimal_constant ('.' digit+)?;
    blank = (cr | lf | tab | ' ')+;

Ignored Tokens
    blank;

Productions
    declarations = declaration another_decl* semicolon?;

    another_decl = comma declaration;

    declaration = identifier initialvalue? vtype? ;

    initialvalue = colon equal number;

    vtype = colon var_event_type;

    var_event_type = {boolean} boolean
                   | {double} double
                   | {float} float
                   | {integer} integer
                   | {int16} int16
                   | {int8} int8
                   | {uint32} uint32
                   | {uint16} uint16
                   | {uint8} uint8
                   | {integer_add} combine integer with add
                   | {integer_mult} combine integer with mult;
```

E.3. Änderungsaktionen

```
/**
  *   Action Interpreter Sablecc file
  *   $Id: ActionInterpreter.grammar,v 1.3 2006/05/12 21:04:39 miwi Exp $
  *
  */
Package kiel.browser.controller.interpreter;

Helpers
    all = [0 .. 127];
    digit = ['0' .. '9'];
    nondigit = ['_' + ['#' + [['a' .. 'z'] + ['A' .. 'Z']]]];
    nonzero_digit = ['1' .. '9'];
    decimal_constant = digit+;
    c_char = [all - ['"' + ['\' + [10 + 13]]]] ;
    c_char_sequence = c_char+;
    cr = 13;
    lf = 10;
    tab = 9;
    not_star = [all - '*'];
    not_cr_lf = [[all -  cr] - lf];
    not_star_slash = [not_star - '/'];
    comma = ',';
    minus = '-';

Tokens
    dot = '.';
    colon = ':';
    semicolon = ';';
    comma = ',';
    l_par = '(';
    r_par = ')';
    l_bracket = '[';
    r_bracket = ']';
    l_brace = '{';
```

```
        r_brace = '}';
        parallel = '||';
        equal = '=';
        add = '+';
        select = '*';
        edge = '->';
        converts = '::=';
        separator = '--';
        copy = '...';

        type = 'type';
        label = 'label';
        options = 'options';
        composite = 'CompositeState' | 'compi';
        region = 'Region' | 'region';
        and = 'ANDState' | 'and';
        or = 'ORState' | 'or';
        simple = 'SimpleState' | 'simple';
        history = 'history' | 'hi';
        deephistory = 'deephistory' | 'dh';
        initial = 'initial' | 'in';
        fork = 'fork' | 'fk';
        join = 'join' | 'jn';
        junction = 'junction' | 'jc';
        sync = 'sync' | 'sy';
        choice = 'choice' | 'ch';
        dynamicchoice = 'dynamicchoice' | 'dc';
        suspend = 'suspend' | 'sd';
        final  = 'final';
        finalor  = 'finalor';
        finaland  = 'finaland';
        browserproperty = 'Browser.KeyStroke.';
        weakabortion = 'weakAbortion' | 'wa';
        strongabortion = 'strongAbortion' | 'sa';
        normaltermination = 'normalTermination' | 'nt';
        conditional = 'conditional';
        internal = 'internalTransition' | 'it';
        parent = 'parent';
        allchilds = 'allChilds';
        new_action = ('A'|'a') 'ction';
        new_menu = ('M'|'m') 'enu';
        name = 'name';
        id = 'id';
        key = 'keyStroke';

        identifier = nondigit (digit | nondigit)*;
        int = '"' minus? digit+ '"';
        string = '"' c_char_sequence? '"';
        blank = (cr | lf | tab | ' ')+;
        comment =  '//' not_cr_lf* (cr|lf);

    Ignored Tokens
        blank,
        comment;

    Productions
        actionfile = action* menu*;

        action =  new_action l_bracket actionid [first]:semicolon actionname [sec]:semicolon actionkey
         actionoptions? r_bracket l_brace actionrule* r_brace;

        menu = new_menu l_bracket menuargument r_bracket l_brace menuitem* r_brace;

        menuargument = name equal string;

        menuitem = {action_ident} identifier semicolon
                 | {separator} separator;

        actionname = name equal string;

        actionkey = key equal keystroke;

        actionid = id equal identifier;

        actionoptions = semicolon options equal string;

        keystroke = {keystring} string | {keyproperty} browserproperty identifier;

        actionrule = lvalue converts [first]:stateortransition [second]:stateortransition* semicolon ;

        lvalue = {stateandtrans} selectedstate anotherstate? anotherselectedtransition?
                | {transonly} selectedtransition;

        selectedstate = identifier select targ?;

        selectedtransition = [source]:identifier? edge select [target]:identifier ttarg?;

        anotherstate = comma selectedstate;

        anotherselectedtransition = comma selectedtransition;
```

```
    targ = l_bracket typeargument r_bracket;

    ttarg = l_bracket ttypeargument r_bracket;

    typeargument = type equal types;

    stateortransition = {state} state semicolon
                      | {transition} transition semicolon
                      | {region} parallel
                      | {copyall} copy;

    state = {simple} identifier select? stateattributes?
          | {composite} identifier select? modifiers? stateattributes? l_brace stateortransition* r_brace;

    modifiers = [first]:modifier [sec]:modifier*;

    modifier = {parent} dot parent
             | {forallchilds} dot allchilds;

    stateattributes = l_bracket stateattrib* r_bracket;

    stateattrib =  {label} label equal string semicolon
                 | {type} typeargument semicolon;


    types = {phistory} history
          | {pdeephistory} deephistory
          | {pinitial} initial
          | {pfork} fork
          | {pjoin} join
          | {pjunction} junction
          | {psync} sync
          | {pchoice} choice
          | {pdynamic} dynamicchoice
          | {psuspend} suspend
          | {final} final
          | {finalor} finalor
          | {finaland} finaland
          | {simple} simple
          | {compo} composite
          | {or} or
          | {and} and
          | {region} region;


    transition = [source]:selectedident? edge select? [target]:selectedident tattributes?;

    selectedident = identifier select?;

    tattributes = l_bracket transattrib* r_bracket;

    transattrib =  {label} label equal string semicolon
                 | {type} ttypeargument semicolon;

    ttypeargument = type equal ttypes;

    ttypes = {weak} weakabortion
             | {strong} strongabortion
             | {initial} initial
             | {conditional} conditional
             | {normal} normaltermination;
```

F. Java Code

F.1. Überblick

In diesem Abschnitt werden die im Rahmen dieser Arbeit erstellte *Java*-Klassen Paketweise vorgestellt.

F.1.1. `kiel.browser`

Das Paket `kiel.browser` ist das Basis-Paket des Browsers, es enthält die allgemeinen Klassen und die Schnittstelle zum `KielFrame`, dem Anwendungsfenster des Projektes *KIEL*.

`Browser`: Die Klasse implementiert ein `KielComponent` und stellt damit die Schnittstelle zum `KielFrame` dar.

`BrowserException`: Diese Klasse wird von Komponenten des Browsers im Fehlerfall geworfen.

`BrowserProperties`: Diese Klasse enthält statische Methoden, mit deren Hilfe es möglich ist, auf die Konfigurationsdatei des Browser zuzugreifen. Alle Zugriffe auf die Konfiguration des Browsers laufen über die Klasse.

F.1.2. `kiel.browser.model`

Der Browser ist nach dem MVC-Prinip (*Model-View-Controller*) aufgebaut. Die Klassen dieses Paketes stellen das Datenmodell des Browsers dar.

`BrowserModel`: Die Klasse ist das wichtigste Modell des Browsers. Hier findet die Datenhaltung des gesamten Browser-Moduls statt. Änderungen im Modell werden registrierten Komponenten über einen Nachrichtenmechanismus mitgeteilt.

`ExpInformation`: Die *KIEL*-Datenstruktur enthält Klassen für Signale und Variablen, aber keine Informationen über die Aktualität der Signale und Variablen oder über das Emitieren eines Signals.

`ModelHelper`: Die Klasse BrowserModel enthält Daten des Browsers und Methoden, um die Klasse zu verändern oder abzufragen. Abhängigkeiten zwischen einzelnen Daten werden erst aufgelöst, wenn sie elementar sind. Auf diese Weise wird durch das Setzen einer neuen `Configuration` eine neue `View` gesetzt.

Komplizierte Abhängigkeiten, die viele Daten aus dem Modell verändern oder von anderen Komponenten benötigt werden, dienen als Hilfsmethoden und werden in diese Klasse ausgelagert.

`ModelMessage`: Instanzen dieses Typs werden bei Änderungen im `BrowserModel` verschickt.

`SignalTableModel`: Diese Klasse implementiert ein Tabellenmodell für Signal- und Variablentabellen.

`ParserThread`: Bei Änderungen im integrierten Text-Editor wird ein neues Statechart, das von der `ParserThread`-Klasse entgegengenommen wird, erzeugt. Erfolgt eine Zeitlang keine Texteingabe, wird das letzte übergebene Statechart, dem `BrowserModel` über die `updateStatechart`-Methode übergeben.

F.1.3. `kiel.browser.model.languages`

Das Paket `kiel.browser.model.languages` dient der Integration textueller Repräsentationen von Statecharts in den Browser.

`Handler`: Mit Hilfe dieser Klasse ist es möglich, verschiedene Statechart-Beschreibungssprachen in den Browser zu integrieren. Dazu müssen die Sprachen das Interface `StatechartLanguage` implementieren und sich über die Methode `registerStateLanguages` dieser Klasse beim Browser registrieren.

`KitLanguage`: Die Klasse implementiert das Interface `StatechartLanguage`, sorgt für die Darstellung eines Statecharts in der *KIT*-Sprache und übernimmt die Funktion des *Syntax-Highlighting* für die *KIT*-Sprache. Der Editiervorgang in einem *KIT*-Dokument wird über einen `DocumentFilter` realisiert, der sich im Paket `kiel.browser.view`, befindet.

F.1.4. `kiel.browser.view`

Das Paket `kiel.browser.view` enthält alle Klassen, die etwas auf dem Bildschirm darstellen oder von solchen Klassen benötigt werden.

`BrowserGUI`: Alle Klassen, die über Änderungen am `BrowserModel` informiert werden müssen, sind von dieser Basisklasse abgeleitet. Sie müssen daher alle eine `update`-Methode, mit der sie Nachrichten vom Browser empfangen können, implementieren.

`BrowserCanvas`: Die Klasse enthält die Zeichenfläche und ist von `BrowserGUI` abgeleitet. Ihre Aufgabe ist es drei Klassen, die das Zeichnen, das Animieren und das Markieren übernehmen, zu koordinieren. Der Zusammenhang wird in Abbildung F.1 verdeutlicht.

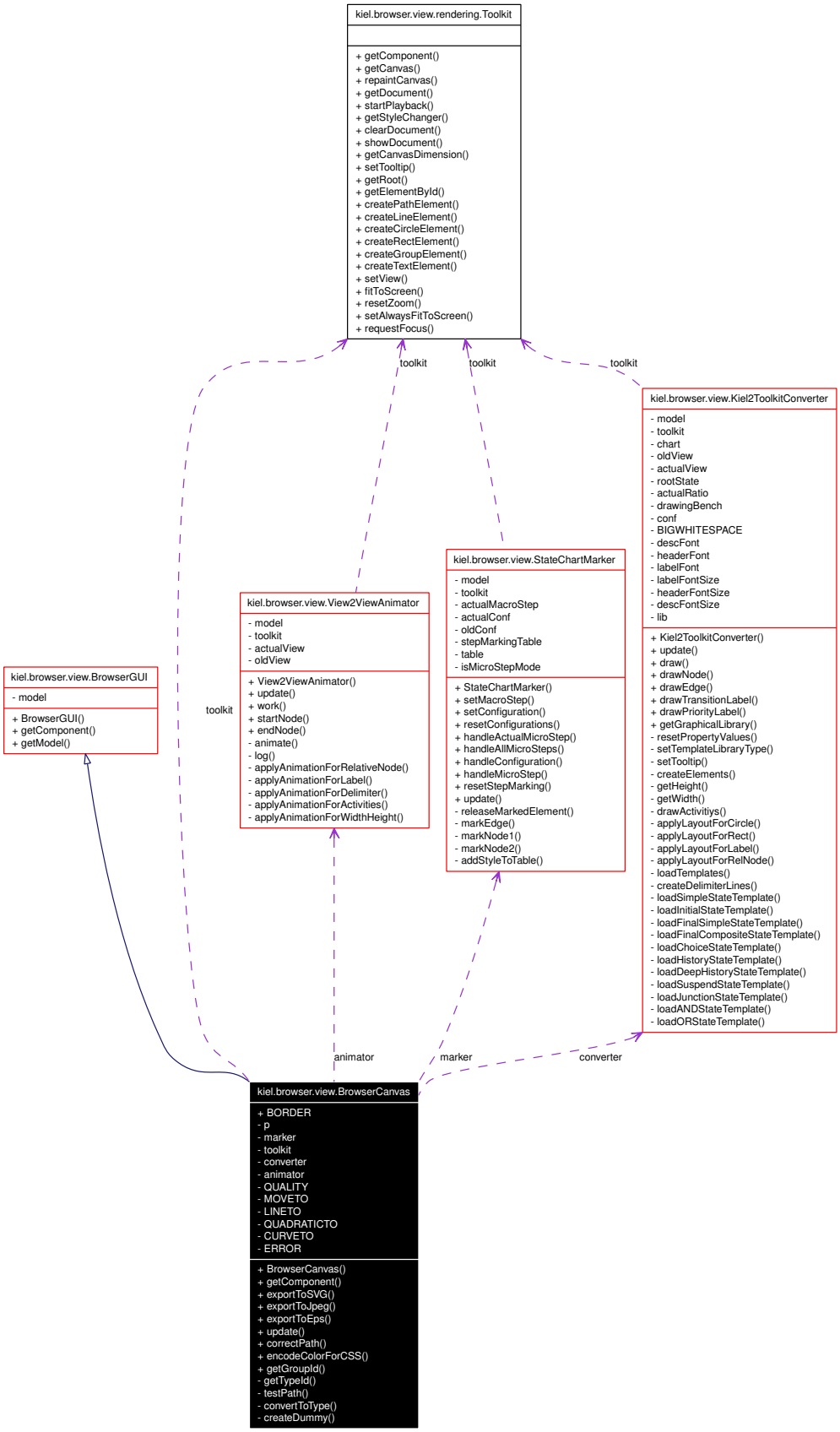


Abbildung F.1.: Kollaborations-Diagramm der `BrowserCanvas`-Klasse

`BrowserToolbar`: Die Klasse erzeugt die Toolbar des Browsers und ist von `BrowserGUI` abgeleitet. Sie enthält zusätzliche eingebettete Klassen, die einfache Aktionen aus der Toolbar übernehmen.

`BrowserTree`: Diese Klasse erzeugt die Baumansicht des Browsers, sie ist von `BrowserGUI` abgeleitet.

`EdgePreferences`: Die Klasse erzeugt den Eigenschaftsdialog für Transitionen, sie ist von `BrowserGUI` abgeleitet.

`NodePreferences`: Diese Klasse erzeugt den Eigenschaftsdialog für Zustände, sie ist von `BrowserGUI` abgeleitet.

`GraphicalObjectsLibrary`: Die Klasse wertet die Methode `getModelSource` der `StateChart`-Klasse aus. Dadurch ist es möglich, unterschiedliche Zeichengebung für Statecharts zu realisieren.

`PlusMinusComboBox`: Komponenten dieses Typs werden in den Eigenschaftsdialogen verwendet, sie dienen dazu, Elemente aus Listen zu löschen und hinzuzufügen.

`ResourceLoader`: Diese Utility-Klasse lädt Resourcen, die sich in einer `jar`-Datei befinden.

`SignalTable`: Das ist eine Signal- und Variablen-Tabellen, mit deren Hilfe Signale und Variablen für die Simulation gesetzt werden können.

`TextEditor`: Die Klasse erzeugt für jede der registrierten Beschreibungssprachen einen einfachen Text-Editor, sie ist von `BrowserGUI` abgeleitet.

`Kiel2ToolkitConverter`: Diese Klasse zeichnet ein Statechart, dazu gebraucht sie ausschließlich Methoden aus dem Interface `Toolkit`.

`View2ViewAnimator`: Die Klasse animiert von einer `View` zur nächsten, dafür ruft sie ausschließlich Methoden aus dem Interface `Toolkit` auf.

`StateChartMarker`: Die Klasse markiert Zustände oder Transitionen, die Ausprägung der Markierung wird dabei über CSS-konforme Zeichenketten definiert.

`BrowserStatusLine`: Diese Klasse erzeugt den Statustext, der im `KielFrame` dargestellt wird. Sie leitet sich von `BrowserGUI` ab.

`PaintUtils`: Die Klasse enthält Hilfsmethoden, um das Aussehen des `KielFrame` auf den Browser zu übertragen.

`LineNumber`: Diese Klasse erzeugt die Zeilenkomponente, die in den Text-Editor integriert wird. Es ist möglich, Zeilen mit beliebigen Bildern zu markieren.

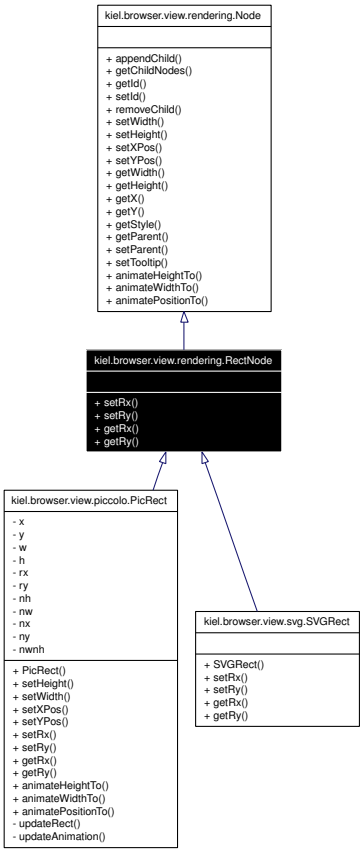


Abbildung F.2.: Klassendiagramm aus dem `kiel.browser.view.rendering` Paket

`MyDocumentFilter:` Diese Klasse implementiert Editoreigenschaften für *KIT*. Wird das *KIT*-Dokument editiert, werden die Änderungen in dieser Klasse verarbeitet. Dazu gehört neben dem *Syntax-Highlighting* des geänderten Dokuments auch das Parsen mit Hilfe des erzeugten *KIT*-Parsers.

F.1.5. `kiel.browser.view.rendering`

In dem Paket `kiel.browser.view.rendering` werden Schnittstellen definiert, um den Browser unabhängig von dem gewählten Toolkit zu machen,siehe Abbildung F.2.

`Node:` Dieses Interface definiert einen allgemeinen Knoten in einem Toolkit.

`GroupNode:` Dieses Interface definiert eine zusammengehörige Gruppe von Knoten.

`PathNode:` Dieses Interface definiert einen Knoten, der einen allgemeinen Pfad repräsentiert.

`CircleNode:` Dieses Interface definiert einen kreisförmigen Konten.

`RectNode`: Dieses Interface definiert einen rechteckigen Knoten mit abgerundeten Kanten.

`TextNode`: Dieses Interface definiert einen Text-Knoten.

`LineNode`: Dieses Interface definiert einen Knoten, der eine Linie darstellt.

`Toolkit`: Dieses Interface stellt Methoden zur Verfügung, die benötigt werden, um die oben genannten Knoten zu erzeugen und die Zeichenfläche zu verwalten.

`StyleChanger`: Dieses Interface enthält eine Methode, die CSS-Zeichenketten auf die oben definierten Knoten anwendet.

F.1.6. `kiel.browser.view.piccolo`

Das Paket `kiel.browser.view.piccolo` enthält die Implementation der Schnittstellen aus dem Paket `kiel.browser.view.rendering` für das Piccolo-Toolkit [17]. Dafür wurden folgende Klassen implementiert: `PiccoloToolkit`, `PicNode`, `PicGroup`, `PicLine`, `PicCircle`, `PicPath`, `PicRect`, `PicStyleCanger`, `PicText`.

F.1.7. `kiel.browser.view.svg`

Das Paket `kiel.browser.view.svg` enthält die Implementation der Schnittstellen aus dem Paket `kiel.browser.view.rendering` für das *CSIRO-SVG*-Toolkit. Dazu wurden folgende Klassen implementiert: `SVGText`, `SVGCircle`, `SVGGroup`, `SVGLine`, `SVGNode`, `SVGPath`, `SVGRect`, `SVGToolkit`, `CSSStyleChanger`.

F.1.8. `kiel.browser.controller`

Das Paket `kiel.browser.controller` enthält den Controller-Anteil des *Model-View-Controller* Design des Browsers.

`StateChartWalker`: Diese Klasse setzt das Konzept zur Navigation in einem Statechart aus Abschnitt 3.3.1 um.

`MacroStepAction`: Die Klasse steuert einen Simulator an und stellt den berechneten `MacroStep` dar.

`MicroStepAction`: Diese Klasse stellt analog zur `MacroStepAction`-Klasse einen `MicroStep` dar.

`PriorityComparator`: Die Klasse wird benötigt, um eine Liste von Transitionen anhand ihrer Prioritäten zu sortieren.

`TracePlayback`: Diese Klasse animiert eine Simulation anhand eines vorher angelegten Traces.

`LayoutController`: Die Klasse ist die Schnittstelle, um den aktuellen Layouter zu bestimmen.

F.1.9. `kiel.browser.controller.interpreter`

Das Paket `kiel.browser.controller.interpreter` enthält einen Interpreter für die neu erstellte Sprache zur Definition von Änderungen in einem Statechart.

`ActionInterpreter`: Diese Klasse lädt die Konfigurationsdatei der Aktionssprache und stellt die Methoden zur Verfügung, um die generierten Menüs auszulesen.

`Translation`: Die Klasse erstellt die Menüs aus der Konfigurationsdatei und legt die einzelnen Regeln in einer Tabelle ab.

`InterpretingAction`: Diese Klasse ist der eigentliche Interpreter und `actionPerformed` wird bei jedem Aufruf einer Aktion aufgerufen.

F.1.10. `kiel.fileInterface.kit`

Das Paket `kiel.browser.controller.interpreter` stellt ein *FileInterface-Plugin* zur Verfügung, um *KIT*-Dateien einzulesen und zu speichern.

`Kit`: Diese Klasse erweitert die abstrakte Klasse `FileInterface`, damit stellt sie eine Schnittstelle für das FileInterface zur Verfügung. Diese Klasse lädt und schreibt eine *KIT*-Datei.

`KitFileFilter`: Die Klasse beschreibt, welche Dateien von dem *KIT*-Parser akzeptiert werden.

`XKit`: Diese Klasse erweitert die abstrakte Klasse `FileInterface`, damit stellt sie eine Schnittstelle für das FileInterface zur Verfügung. Sie schreibt zusätzlich zur Topologie graphische Informationen über das Statechart in eine *KIT*-Datei. Die Dateiendung wird dabei auf „xkit“ (e**X**tended ***KIT***) gesetzt.

`XKitFileFilter`: Diese Klasse beschreibt welche Dateien von dem *KIT*-Parser akzeptiert werden. Die Dateiendung lautet in diesem Fall jedoch “xkit”.

`KitProperties`: Die Klasse verwaltete Einstellungen des *KIT*-Moduls.

`KitGenerator`: Die Klasse erzeugt die gesamte *KIT*-Datei mit Hilfe von `EdgeGenerator` und `NodeGenerator`.

`EdgeGenerator`: Diese Klasse erzeugt Transitionen im *KIT*-Format.

`NodeGenerator`: Die Klasse erzeugt Zustände im *KIT*-Format.

`KitParser`: Diese Klasse stellt ein Interface zum generierten Parser zur Verfügung.

`Translation`: Mit Hilfe dieser Klasse wird der vom Parser generierte AST (***A****bstract* ***S****yntax* ***T****ree*) in die Kiel-Datenstruktur übersetzt.

F.1.11. `kiel.util.declaration`

Das Paket `kiel.util.declaration` stellt einen Parser für die Deklaration von Signalen und Variablen zur Verfügung.

`DeclarationParser:` Die Klasse stellt die Schnittstelle zum generierten Parser dar.

`DeclarationException:` Diese Exception wird geworfen, wenn die Deklaration fehlerhaft war.

F.1.12. `kiel.util.languages`

Das Paket `kiel.util.languages` stellt Schnittstellen zur Verfügung, um verschiedene Sprachen im Browser-Text-Editor darstellen zu können.

`UpdateThread:` Diese Klasse definiert zwei Methoden. Eine Methode, die für das Aktualisieren aller Browser-Komponenten zuständigen *Thread*, ein Statechart und ein entsprechendes Dokument zu übergeben. Die andere Methode sollte aufgerufen werden, um dem *Thread* zu signalisieren, dass noch im Dokument geschrieben wird.

`ILineNumber:` Die Klasse stellt ein Interface für die Zeilennummern-Komponente des Text-Editors zur Verfügung, um Bilder in bestimmte Zeilen einzublenden.

`StatechartLanguage:` Diese Klasse stellt eine Schnittstelle zur Verfügung, um zusätzliche Sprachen im Text-Editor des Browser darzustellen. Wird das Interface implementiert, kann die neue Sprache dem Browser bekannt gemacht werden. Das geschieht über das `KielFrame`, mit Hilfe der `getLanguageHandler`-Methode.

F.2. kiel.browser

F.2.1. Browser

```
//£Id: Browser.java,v 1.137 2007/06/04 06:58:18 khe Exp £
package kiel.browser;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.KeyboardFocusManager;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Observable;
import java.util.Observer;

import javax.swing.AbstractAction;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextPane;
import javax.swing.JToolBar;
import javax.swing.KeyStroke;
import javax.swing.SwingUtilities;
import javax.swing.filechooser.FileFilter;
import javax.swing.text.BadLocationException;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;

import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Transition;
import kiel.configMngr.ConfigMngr;
import kiel.graphicalInformations.View;
import kiel.browser.controller.LayoutController;
import kiel.browser.controller.StateChartWalker;
import kiel.browser.controller.interpreter.ActionInterpreter;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.browser.model.SignalTableModel;
import kiel.browser.model.languages.EsterelLanguage;
import kiel.browser.model.languages.Handler;
import kiel.browser.model.languages.KitLanguage;
import kiel.browser.view.BrowserCanvas;
import kiel.browser.view.BrowserMenuBar;
import kiel.browser.view.BrowserStatusLine;
import kiel.browser.view.BrowserToolbar;
import kiel.browser.view.BrowserTree;
import kiel.browser.view.EdgePreferences;
import kiel.browser.view.NodePreferences;
import kiel.browser.view.ResourceLoader;
import kiel.browser.view.SignalTable;
import kiel.browser.view.TextEditor;
import kiel.util.ComponentCounter;
import kiel.util.ComponentCounterResult;
import kiel.util.DocumentReader;
import kiel.util.DocumentWriter;
import kiel.util.KielFrameRefresh;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;
import kiel.util.main.IKielFrame;
import kiel.util.main.KielComponent;
import kiel.util.main.KielLayouter;
import kiel.browser.model.StatechartFocus;
import javax.swing.JOptionPane;


/**
 * <p>
 * Description: This class implements a Main Kiel Component, it is taken over by
 * an initial Version from Florian Luepke.
 * </p>
 * <p>
 * Copyright: Copyright (c) 2004 Florian Luepke, Adrian Posor, Mirko Wischer
 * </p>
 * <p>
```

```
 * Company: Uni Kiel
 * </p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version
 *          <p>
 *          £Revision: 1.137 £ last modified £Date: 2007/06/04 06:58:18 £
 *          </p>
 *
 * <p>
 * £Log: Browser.java,v £
 * Revision 1.137  2007/06/04 06:58:18  khe
 * Make KIEL GUI work if the KIEL directory is read-only.
 *
 * Revision 1.136  2007/04/05 21:52:37  khe
 * added javadocs
 *
 * Revision 1.135  2007/02/23 00:38:22  khe
 * slightly reformatted
 *
 * Revision 1.134  2007/02/08 13:55:17  khe
 * * changed imports from kiel.base to kiel.preferences
 * * fixed classpaths wrt kiel.base
 * * fixed build dependencies wrt kiel.base
 *
 * Revision 1.133  2007/02/06 18:23:19  khe
 * * new xml based preferences system
 * * new preferences dialog
 * * moved preferences and logfile to new module base to resolve circular build
     time
 *   dependency between util.jar and graphicalInformations.jar
 *
 * Revision 1.132  2006/08/04 09:00:05  kbe
 * added templates for uml statecharts
 *
 * Revision 1.131  2006/06/16 07:47:02  spr
 * correct spelling mistake "triggert"
 *
 * Revision 1.130  2006/05/22 19:21:52  miwi
 * *** empty log message ***
 *
 * Revision 1.129  2006/05/21 21:39:33  miwi
 * *** empty log message ***
 *
 * Revision 1.128  2006/05/21 20:24:01  miwi
 * bug fixing<br>
 *
 * Revision 1.127  2006/05/17 15:27:04  kbe
 * added a message after pressing "New Statechart"
 *
 * Revision 1.126  2006/05/12 21:04:39  miwi
 * bug fixing<br>
 *
 * Revision 1.125  2006/05/08 13:06:35  khe
 * * moved property management to kiel.util.preferences.Preferences
 * * added default value getters to Preferences class
 *
 * Revision 1.124  2006/04/30 12:14:13  khe
 * restored accidentially deleted files
 *
 * Revision 1.122  2006/04/18 19:30:50  miwi
 * *** empty log message ***
 *
 * Revision 1.121  2006/04/18 19:28:28  miwi
 * *** empty log message ***
 * *
 * Revision 1.111  2006/03/26 20:21:10  miwi
 * bugfixing...<br>
 *
 * Revision 1.103  2006/02/21 14:39:26  miwi
 * enh 174 added<br>
 *
 * Revision 1.102  2006/02/20 13:31:26  miwi
 * Highlight code changed<br>
 *
 * Revision 1.101  2006/02/13 13:48:56  kbe
 * Integrated the CheckerOutputGUI into the Tabbed Pane within the Kielframe.
 * Removed the drag and drop ability of the Tabs.<br>
 *
 * Revision 1.100  2006/02/13 10:42:37  kbe
 * Modified the output of the StateChartChecker.<br>
 *
 * Revision 1.99  2006/02/11 13:01:57  kbe
 * reorganized imports<br>
 *
 * Revision 1.98  2006/02/11 12:56:07  kbe
 * Restructured the displaying of the results of the robustness checker.
 * The tab of the checker log is neither included nor displayed any longer.
 * The constructor of StateChartCheckerbase needs a new parameter to distinguish
 * between output in a GUI or output to the command line.<br>
 *
 * Revision 1.93 2006/01/31 11:31:15 miwi Iterpreter added
 *
 * Revision 1.92 2006/01/25 14:34:30 kbe Modified the output of the component
 * counter
 *
 * Revision 1.91 2006/01/25 09:49:12 kbe Modified the actionhandler for count
 * elements to use the new ComponentCounter class
 *
 * Revision 1.87 2006/01/16 18:00:15 miwi Optimzed imports
 *
 * Revision 1.86 2006/01/11 13:13:27 miwi SVG Output enabled again
 *
 * Revision 1.82 2005/12/13 14:08:42 miwi New update functions
 *
 * Revision 1.80 2005/12/01 14:11:36 apo Browser now deletes input signals in
 * table when switching to event simulation mode
 *
 * Revision 1.79 2005/12/01 12:40:40 apo fixed several serious bugs
 *
 * Revision 1.78 2005/11/30 12:04:40 miwi Better ParserThread handling
 *
 * Revision 1.77 2005/11/29 16:44:00 miwi Better Status Line
 *
 * Revision 1.76 2005/11/28 17:10:11 miwi Eps Export added again
 *
 * Revision 1.75 2005/11/28 10:41:25 miwi Browser Version 2.0 RC1 import
 *
 * Revision 1.74 2005/11/04 18:09:11 miwi View menu added for zooming
 *
```

```
 * Revision 1.73 2005/11/04 10:57:36 miwi Properties dialogs removeable
 *
 * Revision 1.72 2005/10/17 15:04:29 apo added menu item 'Simulation Mode'
 *
 * Revision 1.69 2005/09/27 10:43:40 apo browser resets trace when new chart is
 * loaded
 *
 * Revision 1.67 2005/09/13 08:16:42 miwi New features: fit-to-pane, edge prefs
 * etc<br>
 *
 * Revision 1.66 2005/09/10 10:50:19 miwi bug fixing<br>
 *
 * Revision 1.63 2005/08/23 15:37:22 apo added trace file saving feature, fixed
 * trace bugs
 *
 * Revision 1.62 2005/08/17 12:27:01 apo completed trace file loading, extended
 * browser toolbar
 *
 * Revision 1.61 2005/08/11 12:41:40 apo added trace file support (not finished)
 *
 * Revision 1.56 2005/07/28 17:18:49 apo added input event simulation support
 * for Matlab/Stateflow-charts
 *
 * Revision 1.55 2005/07/26 13:31:16 apo added support for input and output
 * variables.
 *
 * Revision 1.47 2005/05/24 17:17:48 miwi New SVG Export added<br>
 *
 * Revision 1.46 2005/05/18 10:34:56 miwi added chart comparison
 *
 * Revision 1.45 2005/05/12 15:12:48 miwi bug 57 removed<br>
 *
 * Revision 1.44 2005/04/04 08:50:11 miwi copy defaults/better exception/better
 * log<br>
 *
 * Revision 1.42 2005/03/23 12:17:37 miwi more Properties better
 * color handling<br>
 *
 * Revision 1.39 2005/03/16 09:13:11 miwi Style checked<br>
 *
 * Revision 1.37 2005/03/16 08:17:21 miwi Save log added<br>
 *
 * Revision 1.36 2005/03/10 13:52:45 miwi New name in title<br>
 *
 * Revision 1.35 2005/03/10 13:00:02 miwi context menu added<br>
 *
 * Revision 1.33 2005/03/10 12:45:42 tkl interface of PseudoLayouter changed<br>
 *
 * Revision 1.32 2005/03/10 11:56:05 miwi new LayoutException integrated<br>
 *
 * Revision 1.31 2005/03/01 08:47:14 tkl changed:
 * kiel.layouter.Handler.getHandeler() to instance()<br>
 *
 * Revision 1.28 2005/02/23 17:15:06 miwi Uninitialised signal support<br>
 *
 * Revision 1.26 2005/02/09 10:05:39 tkl fixes due interface changes in layouter
 * module<br>
 *
 * Revision 1.23 2005/02/08 12:10:09 miwi changing to expressionTable<br>
 *
 * Revision 1.21 2005/02/08 11:15:30 miwi corrected refresh problem in tables
 * <br>
 *
 * Revision 1.19 2005/02/01 18:48:28 miwi new signaltable handling<br>
 *
 * Revision 1.10 2004/11/17 13:38:58 flu set Configurations-Menu transparent
 * (blue)<br>
 *
 * Revision 1.9 2004/11/16 18:59:15 miwi Config Handling / ResourceBundle<br>
 *
 * Revision 1.7 2004/11/16 11:44:46 miwi Configurations enabled again<br>
 * </p>
 *
 * @todo FURTHER Make Edge/Node Properties GUI depended on simulator
 *       (browser.properties)
 */
public class Browser implements KielComponent, Observer, ActionListener {

  /**
   * For the default dimensions.
   */
  private static final int DEFAULT_WIDTH = 100;

  /**
   * For the default dimensions.
   */
  private static final int DEFAULT_HEIGHT = 50;

  /**
   * Order for the tabs.
   */
  private static final int INPUT = 1;

  /**
   * Order for the tabs.
   */
  private static final int OUTPUT = 2;

  /**
   * Order for the tabs.
   */
  private static final int IN_VARS = 3;

  /**
   * Order for the tabs.
   */
  private static final int OUT_VARS = 4;

  /**
   * Order for the tabs.
   */
  private static final int IN_EVENTS = 7;

  /**
   * Order for the tabs.
   */
  private static final int STATE_PREF = 8;

  /**
   * Order for the tabs.
```

```
     */
    private static final int TRANS_PREF = 9;

    /**
     * Order for the tabs.
     */
    private static final int INPUTOUTPUT = -1;

    /**
     * Order for the tabs.
     */
    private static final int SIMLOG = 5;

    /**
     * Order for the tabs.
     */
    private static final int BROWSERLOG = 6;

    /**
     * The layouter.
     */
    private KielLayouter layouter;

    /**
     * The config manager manages all informationens (computes them).
     */
    private ConfigMngr configMngr = new ConfigMngr();

    /**
     * The data model needed for all browser views.
     */
    private static BrowserModel model;

    /**
     * The browser main component.
     */
    private static BrowserCanvas canvas;

    /**
     * The browser tree view component.
     */
    private static BrowserTree tree;

    /**
     * The browser node pref component.
     */
    private static NodePreferences nodePref;

    /**
     * The browser edge pref component.
     */
    private static EdgePreferences edgePref;

    /**
     * The browser toolbar component.
     */
    private static BrowserToolbar tbar;

    /**
     * The browser menu component (file menu entrys alone or whole menu).
     */
    private static BrowserMenuBar menu;

    /**
     * The browser status line.
     */
    private static BrowserStatusLine status;

    /**
     * Signal table (input).
     */
    private static SignalTable inTable;

    /**
     * Signal table (events).
     */
    private static SignalTable inETable = null;

    /**
     * Signal table (output).
     */
    private static SignalTable outTable;

    /**
     * Signal table (local).
     */
    private static SignalTable remainingTable;

    /**
     * Signal Table Model for all signals.
     */
    private SignalTableModel allSignalsModel = new SignalTableModel();

    /**
     * Signal table (all).
     */
    private static SignalTable allTable;

    /**
     * Signal table (input variables).
     */
    private static SignalTable inVarTable;

    /**
     * Signal table (output variables).
     */
    private static SignalTable outVarTable;


    /**
     * Kit Text Editor.
     */
    private static TextEditor kitEdit;

    /**
     * The kiel frame (our parent).
     */
    private static IKielFrame kielFrame;

    /**
```

```
     * Writer for the Browser Log.
     */
    private static DocumentWriter writer;

    /**
     * Reader for the Kit Document.
     */
    private static DocumentReader reader;

    /**
     * The "Simulation Mode" menu.
     */
    private static JMenu simMode = new JMenu("Simulation Mode");

    /**
     * The root menus.
     */
    private static JMenu[] rootMenus = null;

    /**
     * The items of the file menu.
     */
    private static JMenuItem[] fileMenuItems = null;

    /**
     * Signal mode selection for matlab/simulink simulation.
     */
    private static JMenuItem signalMode;

    /**
     * Event mode selection for matlab/simulink simulation.
     */
    private static JMenuItem eventMode;

    /**
     * To select focus on text editor (if internal editor is used).
     */
    private static JCheckBoxMenuItem textEdit;

    /**
     * To select focus on canvas.
     */
    private static JCheckBoxMenuItem keyEdit;

    /**
     * Handler for adding new statechart languages.
     */
    private static Handler languageHandler;

    /**
     * Interpreter for actions.
     */
    private static ActionInterpreter interpreter;
    /**
     * The EsterelLanguage.
     */
    private static EsterelLanguage theEsterelLanguage =
      new EsterelLanguage();

    /**
     * The edit modes menu.
     */
    private JMenu editMode;

    /**
     * The popupmenu.
     */
    private JPopupMenu popup =
      new EsterelTabPopupMenu();

    /**
     * constructor creates and connects all browserGUI components. with
     * browsermodel
     *
     * @param aKielFrame
     *            the kielframe that shows the browser components.
     */
    public Browser(final IKielFrame aKielFrame) {
      kielFrame = aKielFrame;
      model = new BrowserModel();
      walker = new StateChartWalker(model);
      languageHandler = new Handler(model);
      model.addObserver(this);
      BrowserProperties.setBrowserModel(model);
      ModelHelper.setBrowserModel(model);
      tree = new BrowserTree(model);
      canvas = new BrowserCanvas(model);
      tbar = new BrowserToolbar(model);
      menu = new BrowserMenuBar(model);
      status = new BrowserStatusLine(model);
      nodePref = new NodePreferences(model);
      edgePref = new EdgePreferences(model);
      kitEdit = new TextEditor(model);
      inTable = new SignalTable(model.getInputSignalTable());
      outTable = new SignalTable(model.getOutputSignalTable());
      inVarTable = new SignalTable(model.getInputVariablesTable());
      outVarTable = new SignalTable(model.getOutputVariablesTable());
      inETable = new SignalTable(model.getInputEventsTable());
      // remainingTable = new SignalTable(model.getRemainingSignals());
      allSignalsModel.addExpressions(model.getInputSignalTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getOutputSignalTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getLocalEventsTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getVariablesTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allTable = new SignalTable(allSignalsModel);
      browserlog = new JTextPane();
      browserlog.setEditable(false);
      writer = new DocumentWriter(browserlog.getStyledDocument(), browserlog);

      //reader = new DocumentReader(new PlainDocument());
      //kielFrame.setCheckerReader(reader);
      languageHandler.registerStateLanguages(new KitLanguage());
      languageHandler.registerStateLanguages(theEsterelLanguage);
```

```
        ((JTextPane) ((JPanel) ((JScrollPane) kitEdit.getTabs()
                                .getComponent(kitEdit.getTabs()
                                              .indexOfTab(theEsterelLanguage.
          getName())))
                      .getViewport().getView()).getComponent(0))
          .addMouseListener(new PopupListener());
      }

      /**
       *
       */
      public static void requestKeyEdit() {
        keyEdit.doClick();
      }

      /**
       * returnes main menus like layouter and configuration menus.
       *
       * @return the root menus
       */
      public final JMenu[] getRootMenus() {
        if (rootMenus != null) {
          return rootMenus;
        }
        ArrayList menus = new ArrayList();

        JMenu layout = kielFrame.createLayouterMenu(true,
                                                    LayoutController.class);
        for (int i = 0; i < layout.getMenuComponents().length; i++) {
          layout.getMenuComponents()[i].setEnabled(true);
        }

        signalMode = new JMenuItem();
        eventMode = new JMenuItem();

        signalMode.setAction(new AbstractAction("Signals") {
          public void actionPerformed(final ActionEvent e) {
            JTabbedPane pane = kielFrame.getCurrentTabbedPane();
            if (pane != null) {
              pane.setEnabledAt(IN_EVENTS, false);
              pane.setEnabledAt(INPUT, true);
              signalMode.setEnabled(false);
              eventMode.setEnabled(true);
              allSignalsModel.restoreInputSignals();
    //          model.getTrace().record();
    //          model.getTrace().startTrace();
              BrowserProperties.reload();
              model.resetSimulator(false);
              model.showStaticView();
            }
          }
        });

        eventMode.setAction(new AbstractAction("Events") {
          public void actionPerformed(final ActionEvent e) {
            JTabbedPane pane = kielFrame.getCurrentTabbedPane();
            if (pane != null) {
              pane.setEnabledAt(IN_EVENTS, true);
              pane.setEnabledAt(INPUT, false);
              signalMode.setEnabled(true);
              eventMode.setEnabled(false);
              allSignalsModel.removeInputSignals();
    //          model.getTrace().record();
    //          model.getTrace().startTrace();
              BrowserProperties.reload();
              model.resetSimulator(false);
              model.showStaticView();
            }
          }
        });
        simMode.add(signalMode);
        simMode.add(eventMode);
        simMode.setEnabled(false);
        eventMode.setEnabled(false);

        JMenu checker = kielFrame.getCheckerMenu();
        /*  for (int i = 0; i < checker.getMenuComponents().length; i++) {
              if (checker.getMenuComponents()[i].getClass() == JMenuItem.class) {
                  ((JMenuItem) checker.getMenuComponents()[i])
                          .addActionListener(new AbstractAction() {
                              public void actionPerformed(final ActionEvent e) {
                                  JTabbedPane pane = kielFrame
                                          .getCurrentTabbedPane();
                                  if (pane != null) {
                                      pane.setSelectedIndex(CHECKERLOG);
                                  }
                              }
                          });
              }
          }*/

        JMenu view = new JMenu("View");
        view.setMnemonic(BrowserProperties.getViewMenuMnemonic());

        JCheckBoxMenuItem alwaysFit = new JCheckBoxMenuItem(new AbstractAction(
          "Always fit-to-screen") {
          private boolean always = false;

          public void actionPerformed(final ActionEvent e) {
            always = !always;
            model.getToolkit().setAlwaysFitToScreen(always);
            if (always) {
              model.getToolkit().fitToScreen();
            }
          }
        });
        alwaysFit.setAccelerator(BrowserProperties.getAlwaysFitScreenAccelerator());

        JMenuItem resetZoom = new JMenuItem(new AbstractAction("Reset zoom") {
          public void actionPerformed(final ActionEvent e) {
            model.getToolkit().resetZoom();
          }
        });
        resetZoom.setAccelerator(BrowserProperties.getResetZoomAccelerator());

        JMenuItem fitZoom = new JMenuItem(new AbstractAction("Fit-to-screen") {
          public void actionPerformed(final ActionEvent e) {
            model.getToolkit().fitToScreen();
          }
```

```
    });
    fitZoom.setAccelerator(BrowserProperties.getFitScreenAccelerator());

    view.add(alwaysFit);
    view.addSeparator();
    view.add(resetZoom);
    view.add(fitZoom);
    editMode = new JMenu("Edit Mode");
    editMode.setMnemonic(BrowserProperties.getEditModeMenuMnemonic());
    textEdit = new JCheckBoxMenuItem(new AbstractAction("Text editor") {
      public void actionPerformed(final ActionEvent e) {
        kitEdit.requestFocus();
        kitEdit.getSelectedTextPane().setBackground(Color.WHITE);
      }
    });
    keyEdit = new JCheckBoxMenuItem(new AbstractAction("Key navigation") {
      public void actionPerformed(final ActionEvent e) {
        kitEdit.getSelectedTextPane().setBackground(Color.lightGray);
        canvas.getComponent().requestFocus();
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_LEFT, 0), "left");
        canvas.getComponent().getActionMap().put("left",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.leftPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_RIGHT, 0), "right");
        canvas.getComponent().getActionMap().put("right",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.rightPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_UP, 0), "up");
        canvas.getComponent().getActionMap().put("up",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.upPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_DOWN, 0), "down");
        canvas.getComponent().getActionMap().put("down",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.downPressed();
          }
        });
        canvas.getComponent().getInputMap()
          .put(KeyStroke.getKeyStroke(KeyEvent.VK_PAGE_UP, 0),
               "pg_up");
        canvas.getComponent().getActionMap().put("pg_up",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.pageUpPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_PAGE_DOWN, 0),
          "pg_down");
        canvas.getComponent().getActionMap().put("pg_down",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.pageDownPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_UP,
                                 KeyEvent.CTRL_DOWN_MASK), "hist_fw");
        canvas.getComponent().getActionMap().put("hist_fw",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.historyForwardPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_DOWN,
                                 KeyEvent.CTRL_DOWN_MASK), "hist_bk");
        canvas.getComponent().getActionMap().put("hist_bk",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.historyBackPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_HOME, 0), "first");
        canvas.getComponent().getActionMap().put("first",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.gotoFirstPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_HOME,
                                 KeyEvent.CTRL_DOWN_MASK), "abs_first");
        canvas.getComponent().getActionMap().put("abs_first",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.gotoAbsolutFirstPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_END, 0), "last");
        canvas.getComponent().getActionMap().put("last",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.gotoLastKeyPressed();
          }
        });
        canvas.getComponent().getInputMap().put(
          KeyStroke.getKeyStroke(KeyEvent.VK_END,
                                 KeyEvent.CTRL_DOWN_MASK), "abs_last");
        canvas.getComponent().getActionMap().put("abs_last",
                                                 new AbstractAction() {
          public void actionPerformed(final ActionEvent e) {
            walker.gotoDeepestLastPressed();
          }
```

```
      });
      canvas.getComponent().getInputMap().put(
        KeyStroke.getKeyStroke(',', KeyEvent.CTRL_DOWN_MASK),
        "next_st");
      canvas.getComponent().getActionMap().put("next_st",
                                               new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          walker.nextStatePressed();
        }
      });
      canvas.getComponent().getInputMap().put(
        KeyStroke.getKeyStroke('.', KeyEvent.CTRL_DOWN_MASK),
        "next_ps");
      canvas.getComponent().getActionMap().put("next_ps",
                                               new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          walker.nextPseudoPressed();
        }
      });
      canvas.getComponent().getInputMap().put(
        KeyStroke.getKeyStroke('-', KeyEvent.CTRL_DOWN_MASK),
        "next_tr");
      canvas.getComponent().getActionMap().put("next_tr",
                                               new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          walker.nextTransitionPressed();
        }
      });
      canvas.getComponent().getInputMap().put(
        KeyStroke.getKeyStroke(KeyEvent.VK_SPACE, 0), "space");
      canvas.getComponent().getActionMap().put("space",
                                               new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          walker.specialKey1Pressed();
        }
      });
      canvas.getComponent().getInputMap().put(
        KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0), "enter");
      canvas.getComponent().getActionMap().put("enter",
                                               new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          walker.specialKey2Pressed();
        }
      });
    }
  });
  editMode.add(textEdit);
  editMode.add(keyEdit);
  canvas.getComponent().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
    .put(KeyStroke.getKeyStroke('␣', KeyEvent.CTRL_DOWN_MASK),
         "toggle_focus");
  canvas.getComponent().getActionMap().put("toggle_focus",
                                           new AbstractAction() {
    private boolean canvasFocused = false;

    public void actionPerformed(final ActionEvent e) {
      if (canvasFocused) {
        textEdit.doClick();
      } else {
        keyEdit.doClick();
      }
      canvasFocused = !canvasFocused;
    }
  });
  ButtonGroup g = new ButtonGroup();
  g.add(textEdit);
  g.add(keyEdit);
  textEdit.doClick();
  KeyboardFocusManager focusManager =
    KeyboardFocusManager.getCurrentKeyboardFocusManager();
  focusManager.addPropertyChangeListener(
    new PropertyChangeListener() {
    public void propertyChange(final PropertyChangeEvent e) {
      String prop = e.getPropertyName();
      if ("focusOwner".equals(prop) && e.getNewValue() != null) {
        if (e.getNewValue().equals(canvas.getTkCanvas())) {
          keyEdit.doClick();
        } else if (e.getNewValue() instanceof JTextPane) {
          JTextPane tpane = (JTextPane) e.getNewValue();
          JTabbedPane tabs = (JTabbedPane) kitEdit.getComponent();
          JComponent root = tabs.getRootPane();
          JComponent start = tpane;
          do {
            start = (JComponent) start.getParent();
          } while (start != root && start != tabs);
          if (start == tabs) {
            textEdit.doClick();
          }
        }
      }
    }
  }
  );
  JMenu esterel = new JMenu("Esterel");
  JMenuItem load = new JMenuItem(
    new AbstractAction("Load␣Statechart␣from␣Esterel␣Tab") {
    public void actionPerformed(final ActionEvent e) {
      try {
        loadStateChartFromEsterelTab();
      } catch (Exception ex) {
        model.getBrowserLogFile().log(
          LogFile.ERROR,
          "Exception␣during␣loading␣of␣esterel␣from␣tab:␣" + ex
          + "␣");
        ex.printStackTrace(new PrintWriter(model
                                           .getBrowserLogFile().getWriter()));
        JTabbedPane pane = kielFrame.getCurrentTabbedPane();
        if (pane != null && pane.getTabCount() >= BROWSERLOG) {
          pane.setSelectedIndex(BROWSERLOG);
        }
      }
    }
  });
  JMenuItem save = new JMenuItem(
    new AbstractAction("Save␣Esterel␣Tab␣to␣file") {
    public void actionPerformed(final ActionEvent e) {
      try {
        JFileChooser chooser = new JFileChooser();
        int returnVal = chooser.showOpenDialog(null);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
```

```
//            System.out.println("You chose to open this file: "
//                               + chooser.getSelectedFile().getName());
              saveEsterelTextFromEsterelTab(chooser.getSelectedFile());
            }
          } catch (Exception ex) {
            model.getBrowserLogFile().log(
              LogFile.ERROR,
              "Exception during saveing of esterel  tab: " + ex
              + " ");
            ex.printStackTrace(new PrintWriter(model
                                               .getBrowserLogFile().getWriter()));
            JTabbedPane pane = kielFrame.getCurrentTabbedPane();
            if (pane != null && pane.getTabCount() >= BROWSERLOG) {
              pane.setSelectedIndex(BROWSERLOG);
            }
          }
        }
      });
      esterel.add(load);
      esterel.add(save);
      JMenu statechart = new JMenu("Statechart");
      statechart.setMnemonic(BrowserProperties.getStatechartMenuMnemonic());
      JMenuItem chartInfo = new JMenuItem(
        new AbstractAction("Count elements") {
        public void actionPerformed(final ActionEvent e) {
          if (model != null && model.getStateChart() != null) {
            ComponentCounter cntr = new ComponentCounter();
            List results = cntr.countComponentsInChart(model
              .getStateChart());
            Iterator iter = results.iterator();
            model.getBrowserLogFile().log(LogFile.INFO,
                                          "Counting components of statechart:");
            String tempstring = "";
            while (iter.hasNext()) {
              ComponentCounterResult result = (ComponentCounterResult) iter
                                              .next();
              tempstring = "";
              for (int i = 0; i < result.getHierarchy(); i++) {
                tempstring += "  ";
              }
              model.getBrowserLogFile().log(
                LogFile.STRUCTURE,
                tempstring + result.getName() + "="
                + result.getValue());
            }
            keyEdit.doClick();
            interpreter.resetEdit();
            JTabbedPane pane = kielFrame.getCurrentTabbedPane();
            if (pane != null
                && pane.getTabCount() >= BROWSERLOG) {
              pane.setSelectedIndex(BROWSERLOG);
            }
          }
        }
      });
      chartInfo.setAccelerator(BrowserProperties
                               .getCountElementsAccelerator());
      statechart.add(chartInfo);

      interpreter = new ActionInterpreter(canvas
                                          .getComponent(), model);
      menus.add(view);
      menus.add(layout);
      menus.add(checker);
      menus.add(simMode);
      menus.add(editMode);
      menus.addAll(interpreter.getMenus());
      menus.add(statechart);

      rootMenus = (JMenu[]) menus.toArray(new JMenu[1]);

      return rootMenus;
    }

    /**
     * Action class that selects states and transitions.
     */
    private StateChartWalker walker;

    /**
     * flag for loading a File or creating a new file.
     */
    private boolean newFile = false;

    /**
     *
     * @return StateChart
     */
    private static StateChart getStateChartTemplate() {
      StateChart chart = new StateChart();
      chart.setModelSource("Esterel Studio");
      chart.setModelVersion("5.0");
      ORState root = new ORState("Statechart");
      chart.setRootNode(root);
      InitialState init = new InitialState();
      SimpleState simple = new SimpleState("state");
      FinalSimpleState end = new FinalSimpleState("stop");
      root.addSubnode(init);
      root.addSubnode(simple);
      root.addSubnode(end);
      InitialArc initArc = new InitialArc();
      initArc.setSource(init);
      initArc.setTarget(simple);
      Transition trans = new StrongAbortion();
      trans.setSource(simple);
      trans.setTarget(end);
      return chart;
    }

    /**
     *
     * @return null
     */
    public final JMenuItem[] getFileMenuItems1() {
      if (fileMenuItems != null) {
        return fileMenuItems;
      }

      final JMenuItem saveAsItem = new JMenuItem();
```

```
final JMenuItem saveItem = new JMenuItem();
final JMenuItem traceItemL = new JMenuItem();
final JMenuItem traceItemS = new JMenuItem();
final JMenuItem newChart = new JMenuItem();
newChart.setAction(new AbstractAction("New Statechart") {
  public void actionPerformed(final ActionEvent e) {
    KielFrameRefresh.kielRefresh();
    StateChart chart = getStateChartTemplate();
    KielFrameRefresh.kielNewStateChartLoaded(chart);
    KielLayouter defaultLayouter = kielFrame
                                   .getKielLayouterByName(kielFrame
                                   .getDefaultLayouterName());
    try {
      defaultLayouter.setStatechart(chart);
      defaultLayouter.layoutView(defaultLayouter.getStaticView());
      newFile = true;
      setCurrentStateChart(chart, defaultLayouter.getStaticView());
      newFile = false;
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
});
newChart
  .setAccelerator(BrowserProperties.getNewStatechartAccelerator());
saveAsItem.setAction(new AbstractAction("Save as...") {
  public void actionPerformed(final ActionEvent e) {
    if (model.getStateChart() != null) {
      try {
        kielFrame.writeStateChartFileWithSaveDialog(model
          .getStateChart(), model.getView());
        model.setCurrentFile(kielFrame.getCurrentFile());
      } catch (Exception ex) {
        model.getBrowserLogFile().log(LogFile.ERROR,
                                      "Exception during writing: " + ex);
        ex.printStackTrace();
      }
    } else {
      model.getBrowserLogFile().log(LogFile.ERROR,
                                    "Statechart is null");
    }
  }
});
saveItem.setAction(new AbstractAction("Save") {
  public void actionPerformed(final ActionEvent e) {
    if (model.getKitDocument() != null) {
      if (model.getCurrentFile() == null) {
        saveAsItem.doClick();
      } else if (!model.getCurrentFile().getName().endsWith(".kit")) {
        try {
          kielFrame.writeStateChartFile(model.getCurrentFile(),
                                        model.getStateChart(),
                                        model.getView(),
                                        false);
        } catch (Exception ex) {
          saveAsItem.doClick();
          model.getBrowserLogFile().log(LogFile.ERROR, ex.getMessage());
        }
      } else {
        try {
          FileWriter saveKitWriter = new FileWriter(
            model.getCurrentFile().getAbsolutePath());
          String kitText = model.getKitDocument().getText(0,
            model.getKitDocument().getLength());
          saveKitWriter.write(kitText);
          model.getBrowserLogFile().log(
            LogFile.INFO,
            "Written Kit to "
            + model.getCurrentFile().getName());
          saveKitWriter.close();
        } catch (IOException ex1) {
          ex1.printStackTrace();
        } catch (BadLocationException ex2) {
          ex2.printStackTrace();
        }
      }
    } else {
      saveAsItem.doClick();
    }
  }
});
saveItem.setAccelerator(BrowserProperties.getSaveKitAccelerator());
traceItemL.setAction(new AbstractAction("Load trace file...") {
  public void actionPerformed(final ActionEvent e) {
    try {
      kielFrame.openTraceWithOpenDialog();
      model.setTrace(kielFrame.getTraceData());
    } catch (Exception ex) {
      model.getBrowserLogFile().log(
        LogFile.ERROR,
        "Exception during loading of trace file: " + ex
        + " ");
      ex.printStackTrace(new PrintWriter(model
                                         .getBrowserLogFile().getWriter()));
      JTabbedPane pane = kielFrame.getCurrentTabbedPane();
      if (pane != null && pane.getTabCount() >= BROWSERLOG) {
        pane.setSelectedIndex(BROWSERLOG);
      }
    }
  }
});
traceItemL.setAccelerator(BrowserProperties.getLoadTraceAccelerator());
traceItemS.setAction(new AbstractAction("Save trace file...") {
  public void actionPerformed(final ActionEvent e) {
    try {
      kielFrame.writeTraceWithSaveDialog(model.getTrace());
    } catch (Exception ex) {
      model.getBrowserLogFile().log(LogFile.ERROR,
                                    "Exception during saving of trace file: "
  "
                                    + ex);
    }
  }
});

fileMenuItems = new JMenuItem[] {newChart, saveItem, saveAsItem,
                                 traceItemL, traceItemS};
return fileMenuItems;
}
```

```
    /**
     * Export file name are added these numbner if file exists.
     */
    private static int nr = 1;

    /**
     *
     * @return file menu items esp. export as jpg, export as eps
     */
    public final JMenuItem[] getFileMenuItems2() {
      JMenuItem exportEps = new JMenuItem(new AbstractAction(
        "Export view as eps") {
        public void actionPerformed(final ActionEvent e) {
          if (model.getStateChart() != null) {
            File f = kielFrame.getCurrentFile();
            if (f == null) {
              f = new File(XMLPreferences.getSafeWriteDirectory() + ""
                      + model.getStateChart().getRootNode() + "_"
                      + model.getLayouterName() + "_" + nr + ".eps");
              nr++;
            }
            JFileChooser chooser =
              new JFileChooser();
            chooser.setSelectedFile(f);
            chooser.setDialogType(JFileChooser.SAVE_DIALOG);
            chooser.setFileFilter(new FileFilter() {
              public boolean accept(final File f) {
                return f.isDirectory() || f.getName().toLowerCase()
                    .endsWith(".eps");
              }
              public String getDescription() {
                return "eps - encapsulated postscript";
              }
            });
            int returnVal = chooser.showSaveDialog(null);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
              f = chooser.getSelectedFile();
              if (f.exists()) {
                returnVal = JOptionPane.showConfirmDialog(kielFrame.getJFrame(),
                    "Should the existing file be overwritten?", "File exists",
                    JOptionPane.YES_NO_CANCEL_OPTION);
                if (returnVal == JOptionPane.CANCEL_OPTION) {
                  // okay save aborted
                  return;
                } else if (returnVal == JOptionPane.NO_OPTION) {
                  int i = 1;
                  do {
                    f = new File(f.getAbsolutePath().substring(0,
                      f.getAbsolutePath().lastIndexOf('.')) +  i + ".eps");
                    i++;
                  } while (f.exists() || i == Integer.MAX_VALUE);
                }
              }
              canvas.exportToEps(f);
              model.setStatus("Wrote: "
                              + f.getAbsoluteFile().
                              toString());
            }
          }
        }
      });
      JMenuItem exportSVG = new JMenuItem(new AbstractAction(
        "Export view as svg") {
        public void actionPerformed(final ActionEvent e) {
          if (model.getStateChart() != null) {
            File f = kielFrame.getCurrentFile();
            if (f == null) {
              f = new File(XMLPreferences.getSafeWriteDirectory() + ""
                      + model.getStateChart().getRootNode() + "_"
                      + model.getLayouterName() + "_" + nr + ".eps");
              nr++;
            }
            JFileChooser chooser =
              new JFileChooser();
            chooser.setSelectedFile(f);
            chooser.setDialogType(JFileChooser.SAVE_DIALOG);
            chooser.setFileFilter(new FileFilter() {
              public boolean accept(final File f) {
                return f.isDirectory() || f.getName().toLowerCase()
                    .endsWith(".eps");
              }
              public String getDescription() {
                return "eps - encapsulated postscript";
              }
            });
            int returnVal = chooser.showSaveDialog(null);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
              f = chooser.getSelectedFile();
              if (f.exists()) {
                returnVal = JOptionPane.showConfirmDialog(kielFrame.getJFrame(),
                    "Should the existing file be overwritten?", "File exists",
                    JOptionPane.YES_NO_CANCEL_OPTION);
                if (returnVal == JOptionPane.CANCEL_OPTION) {
                  // okay save aborted
                  return;
                } else if (returnVal == JOptionPane.NO_OPTION) {
                  int i = 1;
                  do {
                    f = new File(f.getAbsolutePath().substring(0,
                      f.getAbsolutePath().lastIndexOf('.')) +  i + ".eps");
                    i++;
                  } while (f.exists() || i == Integer.MAX_VALUE);
                }
              }
              canvas.exportToSVG(f);
              model.setStatus("Wrote: "
                              + f.getAbsoluteFile().
                              toString());
            }
          }
        }
      });
//      JMenuItem exportTkFormat = new JMenuItem(new AbstractAction(
//              "Print view in toolkit format") {
//          public void actionPerformed(final ActionEvent e) {
//            try {
//              System.out.println(""
//                                 + model.getToolkit().getDocument().
//                                 getText(0,
//                                 model.getToolkit().getDocument().getLength()));
```

```
//              } catch (BadLocationException ex) {
//              }
//            }
//        });
    return new JMenuItem[] {exportEps, exportSVG};
  }

  /**
   *
   * @return the toolbars
   */
  public final ToolBar[] getToolBars() {
    JToolBar toolbar = (JToolBar) tbar.getComponent();
    toolbar.setOpaque(false);
    for (int i = 0; i < toolbar.getComponentCount(); i++) {
      ((JComponent) toolbar.getComponent(i)).setOpaque(false);
    }
    return new ToolBar[] {new ToolBar(toolbar, "MainMenu", true)};
  }

  /**
   * loads a statechart from the esterel tab.
   * @throws Exception any exception.
   */
  public static void loadStateChartFromEsterelTab() throws Exception {
    File esterelTab = File.createTempFile("esterelTab", ".strl");
    Browser.saveEsterelTextFromEsterelTab(esterelTab);
    kielFrame.open(esterelTab);
    esterelTab.deleteOnExit();
  }

  /**
   * writes the content of an esterel tab to a file.
   *
   * @param aFile
   *            the file.
   * @throws Exception
   *             any exception.
   */
  public static void saveEsterelTextFromEsterelTab(final File aFile) throws
    Exception {

    JTabbedPane tabs = kitEdit.getTabs();
    String esterelprog = "No esterel text in tab";
    esterelprog = ((JTextPane) ((JPanel) ((JScrollPane) tabs
                                          .getComponent(tabs.indexOfTab(
                                            theEsterelLanguage.getName())))
                                .getViewport().getView()).getComponent(0)).
                  getText();
    BufferedWriter out = new BufferedWriter(new OutputStreamWriter(
      new FileOutputStream(aFile)));
    out.write(esterelprog, 0, esterelprog.length());
    out.close();
  }

  /**
   * @return the drawing canvas as it is our main component here
   */
  public final JComponent getMainComponent() {
    JSplitPane pane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, canvas
                                     .getComponent(), kitEdit.getComponent());
    // pane.setDividerSize(10);
    final double resizeWeight = 0.8;
    final int location = 500;
    pane.setResizeWeight(resizeWeight);
    pane.setDividerLocation(location);
    pane.setOneTouchExpandable(true);
    pane.setBorder(null);
    // pane.setEnabled(false);
    return pane;
  }

  /**
   *
   * @return the tree view for the actual state
   */
  public final JComponent getTreePanel() {
    return tree.getComponent();
  }

  /**
   * TextPane that contains browser log.
   */
  private JTextPane browserlog;

  /**
   * TextPane that contains sim log.
   */
  private JTextPane simlog;

  /**
   * Table that contains the output of the checker.
   */
  //private JScrollPane checkerPane;

  /**
   * Tabbed Panels for logging and SignalTables.
   *
   * @return the tabbed panels below the main/tree components
   */
  public final JTabbedPane getSubPanels() {
    final JTabbedPane tab = new JTabbedPane();
    tab.setMinimumSize(new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT));
    tab.setSize(new Dimension(DEFAULT_WIDTH, 150));
    model.getBrowserLogFile().enableLog();
    model.getBrowserLogFile().setLogLevel(0);
    SimpleAttributeSet error = new SimpleAttributeSet();
    SimpleAttributeSet opti = new SimpleAttributeSet();
    SimpleAttributeSet warning = new SimpleAttributeSet();
    StyleConstants.setBold(error, true);
    StyleConstants.setForeground(error, Color.red);
    StyleConstants.setBold(opti, true);
    StyleConstants.setForeground(opti, Color.green);
    StyleConstants.setForeground(warning, Color.orange);
    writer.addStyleForRegEx("\\[Browser 10\\].*\n", error);
    //writer.addStyleForRegEx("\\[Checker\\].*\n", opti);
    model.getBrowserLogFile().setWriter(writer);

    simlog = new JTextPane();
    simlog.setEditable(false);
```

```
model.getSimulatorLogFile().enableLog();
model.getSimulatorLogFile().setLogLevel(0);
model.getSimulatorLogFile().setWriter(
  new DocumentWriter(simlog.getStyledDocument(), simlog));
((DocumentWriter) model.getSimulatorLogFile().getWriter())
  .addStyleForRegEx("\\[.*10\\].*\n", error);
tab.insertTab("All Signals/Variables", null, allTable.getComponent(),
              "All Signals/Variables", 0);
tab.insertTab("Input Signals", null, inTable.getComponent(),
              "Input Signals", INPUT);
tab.insertTab("Output Signals", null, outTable.getComponent(),
              "Output Signals", OUTPUT);
tab.insertTab("Input Variables", null, inVarTable.getComponent(),
              "Input Variables", IN_VARS);
tab.insertTab("Output Variables", null, outVarTable.getComponent(),
              "Output Variables", OUT_VARS);
tab.insertTab("Simulator Log", null, new JScrollPane(simlog),
              "Simulator Logs", SIMLOG);
tab.insertTab("Browser Log", null, new JScrollPane(browserlog),
              "Browser Logs", BROWSERLOG);
tab.insertTab("Input Events", null, inETable.getComponent(),
              "Input Events", IN_EVENTS);
tab.insertTab("State Properties", null, nodePref.getComponent(),
              "Preferences of selected state", STATE_PREF);
tab.insertTab("Transition Properties", null, edgePref.getComponent(),
              "Preferences of selected transition", TRANS_PREF);

final JPopupMenu contextMenuTabs = new JPopupMenu();
JMenuItem nodePrefItem = new JMenuItem(new AbstractAction(
  "Edge and Node Properties") {
  private boolean shown = true;

  public void actionPerformed(final ActionEvent e) {
    JTabbedPane pane = kielFrame.getCurrentTabbedPane();
    if (pane != null) {
      if (shown) {
        pane.removeTabAt(TRANS_PREF);
        pane.removeTabAt(STATE_PREF);
      } else {
        pane.insertTab("State Properties", null, nodePref
                       .getComponent(),
                       "Preferences of selected state", STATE_PREF);
        pane.insertTab("Transition Properties", null, edgePref
                       .getComponent(),
                       "Preferences of selected transition",
                       TRANS_PREF);
      }
      shown = !shown;
    }
  }
});
contextMenuTabs.add(nodePrefItem);
// Nesting a MouseListener
tab.addMouseListener(new MouseAdapter() {
  public void mousePressed(final MouseEvent ev1) {
    // Check if it is a right click
    if (SwingUtilities.isRightMouseButton(ev1)) {
      // If it is a right click then show the menu
      contextMenuTabs.show(tab, ev1.getX(), ev1.getY());
      contextMenuTabs.repaint();
    } else {
      contextMenuTabs.setVisible(false);
    }
  }
});
final JPopupMenu contextMenuSim = new JPopupMenu();
JMenuItem menuItem1 = new JMenuItem("Save as...");
menuItem1.addActionListener(this);
menuItem1.setActionCommand("simlog");
contextMenuSim.add(menuItem1);
/*
 * Making the menu heavyweight - this prevents the menu going behind the
 * canvas. This is a bug with Swing menus,other Swing components do not
 * have this issue.
 */
contextMenuSim.setLightWeightPopupEnabled(false);
final JPopupMenu contextMenuBrowser = new JPopupMenu();
JMenuItem menuItem2 = new JMenuItem("Save as...");
menuItem2.addActionListener(this);
menuItem2.setActionCommand("browserlog");
contextMenuBrowser.add(menuItem2);
JMenuItem menuItem3 = new JMenuItem("Clear log");
menuItem3.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    try {
      browserlog.getStyledDocument().remove(
        0,

        browserlog.getStyledDocument().getEndPosition()
        .getOffset() - 1);
      writer.reset();
    } catch (BadLocationException ex) {
      ex.printStackTrace();
    }
  }
});
contextMenuBrowser.add(menuItem3);
contextMenuBrowser.setLightWeightPopupEnabled(false);
// Nesting a MouseListener
simlog.addMouseListener(new MouseAdapter() {
  public void mousePressed(final MouseEvent ev1) {
    // Check if it is a right click
    if (SwingUtilities.isRightMouseButton(ev1)) {
      // If it is a right click then show the menu
      contextMenuSim.show(simlog, ev1.getX(), ev1.getY());
      contextMenuSim.repaint();
    } else {
      contextMenuSim.setVisible(false);
    }
  }
});
browserlog.addMouseListener(new MouseAdapter() {
  public void mousePressed(final MouseEvent ev2) {
    // Check if it is a right click
    if (SwingUtilities.isRightMouseButton(ev2)) {
      // If it is a right click then show the menu
      contextMenuBrowser.show(browserlog, ev2.getX(), ev2.getY());
      contextMenuBrowser.repaint();
    } else {
      contextMenuBrowser.setVisible(false);
```

```
        }
      }
    });
    return tab;
  }

  /**
   *
   * @param parentFrame
   *            the parentFrame (for dialogs f.e)
   */
  public final void setKielFrame(final IKielFrame parentFrame) {
    kielFrame = parentFrame;
  }

  /**
   * is not yet called.
   */
  public final void activate() {
    // System.err.println("activate called");
  }

  /**
   *
   * @return IKielFrame that is showing the browser.
   */
  public static IKielFrame getKielFrame() {
    return kielFrame;
  }

  /**
   * @return BrowserModel
   */
  public static BrowserModel getModel() {
    return model;
  }

  /**
   *
   * @param statechart
   *            the new actual statechart
   * @param view
   *            the orginal view for this chart
   */
  public final void setCurrentStateChart(final StateChart statechart,
                                         final View view) {
    if (statechart != null) {
      kielFrame.getJFrame().setCursor(new Cursor(Cursor.WAIT_CURSOR));
      if (newFile) {
        model.setCurrentFile(null);
      } else {
        model.setCurrentFile(kielFrame.getCurrentFile());
      }
      BrowserProperties.reload();
      model.getBrowserLogFile().log(
        LogFile.INFO,
        "KielFrame tries to set new statechart "
        + statechart.getRootNode() + " View:" + view);

      // Configuration[] configs = configMngr.getConfigs(statechart);
      // model.setConfigurations(configs);
      ModelHelper.startTrace();
//          if (statechart.equals(model.getStateChart())) {
//            model.updateChart(statechart, "kielframe");
//          } else {
      model.setStateChart(statechart);
      layouter = kielFrame.getKielLayouterByName("OriginalLayout");
      try {
        layouter.setStatechart(statechart);
        if (view != null) {
          layouter.setStaticView(view);
        } else {
          model.getBrowserLogFile().log(LogFile.INFO,
                                        "Cannot set view on browser switch (view
    == null)");
        }
      } catch (Exception ex) {
        model.getBrowserLogFile().log(LogFile.ERROR,
                                      "Cannot set statechart for layouter: "
                                      + ex);
      }

      model.setLayouter(layouter, "OriginalLayout");
      model.showStaticView();
//          }
      JTabbedPane pane = kielFrame.getCurrentTabbedPane();
      if (pane != null) {
        if (statechart.getInputVariables().size() == 0) {
          pane.setEnabledAt(IN_VARS, false);
        } else {
          pane.setEnabledAt(IN_VARS, true);
        }
        if (statechart.getOutputVariables().size() == 0) {
          pane.setEnabledAt(OUT_VARS, false);
        } else {
          pane.setEnabledAt(OUT_VARS, true);
        }
        if (statechart.getOutputEvents().size() == 0) {
          pane.setEnabledAt(OUTPUT, false);
        } else {
          pane.setEnabledAt(OUTPUT, true);
        }
        if (ModelHelper.inputEventsSupport()) {
          pane.setEnabledAt(IN_EVENTS, true);
          pane.setEnabledAt(INPUT, false);
          allSignalsModel.removeInputSignals();
        } else {
          pane.setEnabledAt(INPUT, true);
        }
      }
      if (statechart.getModelSource().equals("Matlab/Stateflow")) {
        simMode.setEnabled(true);
      } else {
        simMode.setEnabled(false);
      }
      kielFrame.getJFrame().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
    interpreter.resetEdit();
    keyEdit.doClick();
  }
```

```
  /**
   * @return the actual chart (from the model)
   */
  public final StateChart getCurrentStateChart() {
    return model.getStateChart();
  }

  /**
   * @return the actual view (from the model)
   */
  public final View getCurrentView() {
    return model.getView();
  }

  /**
   *
   * @return the browser icon (until now null)
   */
  public final ImageIcon getImageIcon() {
    return ResourceLoader.createImageIcon("kiel/browser/images/middle.gif");
  }

  /**
   *
   * @return a status line component (until now null)
   */
  public final JComponent getStatusLineComponent() {
    return status.getComponent();
  }

  /**
   * @param parm1
   *            the object that send the message
   * @param parm2
   *            the message itself (as String)
   */
  public final void update(final Observable parm1, final Object parm2) {
    if (parm2.equals(BrowserModel.TABLE_UPDATE)) {
      allSignalsModel.resetEvents();
      allSignalsModel.addExpressions(model.getInputSignalTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getOutputSignalTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getLocalEventsTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getVariablesTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getInputVariablesTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      allSignalsModel.addExpressions(model.getOutputVariablesTable()
                                     .getExpInformations(),
                                     SignalTableModel.UNKNOWN_TABLE);
      // allSignalsModel.sortListAgain();
    } else if (parm2.equals(BrowserModel.MODE_CHANGE)) {
      switch (model.getMode()) {
      case BrowserModel.EDIT:

        // switching to edit
        editMode.setEnabled(true);
        break;
      case BrowserModel.SIMULATION:
        editMode.setEnabled(false);
        JTabbedPane pane = kielFrame.getCurrentTabbedPane();
        if (pane != null && pane.getTabCount() > 0) {
          pane.setSelectedIndex(0);
        }
        break;
      }
    } else if (parm2.equals(BrowserModel.TABLE_REFRESH)) {
      allSignalsModel.fireTableDataChanged();
    } else if (parm2.equals(BrowserModel.KIT_DOC_CHANGED)) {
      kielFrame.setCheckerReader(new DocumentReader(model
        .getKitDocument()));
    } else if (parm2.equals(BrowserModel.FOCUS_CHANGED)) {
      StatechartFocus changedFocus = model.getFocus();
      if ((changedFocus.hasNode1Changed() && changedFocus.getNodes()[0] != null)
          || (changedFocus.hasNode2Changed()
              && changedFocus.getNodes()[1] != null)) {
        JTabbedPane pane = kielFrame.getCurrentTabbedPane();
        if (pane != null && pane.getTabCount() > STATE_PREF) {
          pane.setSelectedIndex(STATE_PREF);
        }
      } else if (changedFocus.hasEdgeChanged()) {
        JTabbedPane pane = kielFrame.getCurrentTabbedPane();
        if (pane != null && pane.getTabCount() > TRANS_PREF) {
          pane.setSelectedIndex(TRANS_PREF);
        }
      }
    } else if (parm2.equals(BrowserModel.COMPUTATION_STARTED)) {
      kielFrame.getJFrame().setCursor(new Cursor(Cursor.WAIT_CURSOR));
    } else if (parm2.equals(BrowserModel.COMPUTATION_COMPLETED)) {
      kielFrame.getJFrame().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    } else if (parm2.equals(BrowserModel.WARNING)) {
      kielFrame.handleException((String) BrowserModel.WARNING
                                .getParameter(), false, null);
    } else if (parm2.equals(BrowserModel.EXCEPTION)) {
      kielFrame.handleException(((Exception) BrowserModel.EXCEPTION
                                 .getParameter()).toString(), false,
                                (Exception) BrowserModel.EXCEPTION.getParameter()
      );
    }
  }

  /**
   * @return String that should be shown in KielFrame
   */
  public final String getKielFrameTitle() {
    return BrowserProperties.getBrowserTitle();
  }

  /**
   *
   * @param e
   *            ActionEvent tab context menu performes action here
```

```
     */
    public final void actionPerformed(final ActionEvent e) {
      if (e.getActionCommand().equalsIgnoreCase("simlog")) {
        // Simlog called save log
        saveLogFile(simlog);
      } else {
        saveLogFile(browserlog);
      }
    }

    /**
     * @param pane
     *            JTextPane to save
     */
    private void saveLogFile(final JTextPane pane) {
      JFileChooser saver = new JFileChooser();
      if (saver.showSaveDialog(pane) == JFileChooser.APPROVE_OPTION) {
        try {
          String log = pane.getDocument().getText(0,
                                                  pane.getDocument().getLength());
          try {
            FileWriter saveLogWriter = new FileWriter(saver
              .getSelectedFile());
            saveLogWriter.write(log);
            saveLogWriter.close();
          } catch (IOException ex1) {
            ex1.printStackTrace();
            return;
          }
        } catch (BadLocationException ex) {
          ex.printStackTrace();
          return;
        }
      }
    }

    /**
     * @param o GraphicalObject
     * @param c Color
     */
    public final void highlightObject(final GraphicalObject o, final Color c) {
      if (o instanceof Edge
          && model.getStateChart().getAllObjects().contains(o)) {
        model.markObjekt(o, "fill:none;stroke:"
                            + BrowserCanvas.encodeColorForCSS(c) + ";");
      } else if (o instanceof Node
                 && model.getStateChart().getAllObjects().contains(o)) {
        model.markObjekt(o, "fill:" + BrowserCanvas.encodeColorForCSS(c)
                            + ";stroke:black;");
      }
    }

    /**
     * @param o GraphicalObject
     * @param css String
     */
    public final void highlightObject(final GraphicalObject o,
                                      final String css) {
      if (model.getStateChart().getAllObjects().contains(o)) {
        model.markObjekt(o, css);
      }
    }

    /**
     * @param o GraphicalObject
     */
    public final void removeHighlight(final GraphicalObject o) {
      if (model.getStateChart().getAllObjects().contains(o)) {
        model.removeMark(o);
      }
    }

    /**
     * Remove all marks.
     */
    public final void clearMarks() {
      model.clearAllMarks();
    }

    /**
     * Implements a popupmenu.
     *
     */
    public class EsterelTabPopupMenu extends JPopupMenu {

      public void processMouseEvent(MouseEvent event) {
        if (event.isPopupTrigger()) {
          this.show(
            event.getComponent(),
            event.getX(),
            event.getY()
            );
        }
        super.processMouseEvent(event);
      }

      /**
       * loads a statechart from the esterel tab.
       * @throws Exception any exception.
       */
      public void loadStateChartFromEsterelTab() throws Exception {
        File esterelTab = File.createTempFile("esterelTab", ".strl");
        saveEsterelTextFromEsterelTab(esterelTab);
        if (esterelTab.length() > 0) {
          kielFrame.setFileHistory(false);
          kielFrame.open(esterelTab);
          kielFrame.setFileHistory(true);
        }
        esterelTab.deleteOnExit();
      }

      /**
       * writes the content of an esterel tab to a file.
       *
       * @param aFile
       *            the file.
       * @throws Exception
       *             any exception.
       */
      public void saveEsterelTextFromEsterelTab(File aFile) throws Exception {
```

```
    JTabbedPane tabs = kitEdit.getTabs();
    String esterelprog = "";
    esterelprog = ((JTextPane) ((JPanel) ((JScrollPane) tabs
                                          .getComponent(tabs.indexOfTab(
      theEsterelLanguage.getName())))
                               .getViewport().getView()).getComponent(0)).
                    getText();
    if (esterelprog.compareTo("") != 0
        && !esterelprog.startsWith("no chart loaded")) {
      BufferedWriter out = new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream(aFile)));
      out.write(esterelprog, 0, esterelprog.length());
      out.close();
    }
  }

  public EsterelTabPopupMenu() {
    JMenuItem mi;

    mi = new JMenuItem("Transform to SSM");
    mi.addActionListener(
      new AbstractAction("Transform to SSM") {
      public void actionPerformed(final ActionEvent e) {
        try {
          loadStateChartFromEsterelTab();
        } catch (Exception ex) {
          model.getBrowserLogFile().log(
            LogFile.ERROR,
            "Exception during loading of esterel from tab: " + ex
            + " ");
          ex.printStackTrace(new PrintWriter(model
                                             .getBrowserLogFile().getWriter()))
 ;
          JTabbedPane pane = kielFrame.getCurrentTabbedPane();
          if (pane != null && pane.getTabCount() >= BROWSERLOG) {
            pane.setSelectedIndex(BROWSERLOG);
          }
        }
      }
    });
    add(mi);
    mi = new JMenuItem("Save Esterel to File");
    mi.addActionListener(new AbstractAction("Save Esterel to File") {
      public void actionPerformed(final ActionEvent e) {
        try {
          JFileChooser chooser =
            new JFileChooser(kielFrame.getFileInterfaceSaveDir());
          chooser.setDialogType(JFileChooser.SAVE_DIALOG);
          chooser.setFileFilter(new FileFilter() {
            public boolean accept(File f) {
              return f.isDirectory() ||
                f.getName().toLowerCase().endsWith(".strl");
            }

            public String getDescription() {
              return ".strl - esterel program file";
            }
          });
          int returnVal = chooser.showSaveDialog(null);
          if (returnVal == JFileChooser.APPROVE_OPTION) {
            saveEsterelTextFromEsterelTab(chooser.getSelectedFile());
          }
        } catch (Exception ex) {
          model.getBrowserLogFile().log(
            LogFile.ERROR,
            "Exception during saveing of esterel  tab: " + ex
            + " ");
          ex.printStackTrace(new PrintWriter(model
                                             .getBrowserLogFile().getWriter()))
 ;
          JTabbedPane pane = kielFrame.getCurrentTabbedPane();
          if (pane != null && pane.getTabCount() >= BROWSERLOG) {
            pane.setSelectedIndex(BROWSERLOG);
          }
        }
      }
    });
    add(mi);

    addSeparator();
    JCheckBoxMenuItem timetriggert =
      new JCheckBoxMenuItem(new AbstractAction("Time Triggered") {
      private boolean isTimeTriggert = false;

      public void actionPerformed(final ActionEvent e) {
        isTimeTriggert = !isTimeTriggert;
      }
    });
    timetriggert.setState(false);
    add(timetriggert);

  }
}


class PopupListener extends MouseAdapter {
  public void mousePressed(MouseEvent e) {
    maybeShowPopup(e);
  }

  public void mouseReleased(MouseEvent e) {
    maybeShowPopup(e);
  }

  private void maybeShowPopup(MouseEvent e) {
    if (e.isPopupTrigger()) {
      popup.show(e.getComponent(), e.getX(), e.getY());
    }
  }
}
```

F.2.2. BrowserException

```
package kiel.browser;



/**
 * <p>Description: Browser Exception is thrown whenever there is an uncoverable
 * error in the program.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.10 £ last modified £Date: 2006/04/30 12:14:13 £
 */
public class BrowserException extends Exception {

  /** Creates a new instance of BrowserException with empty message. */
  public BrowserException() {
    super();
  }

  /**
   * Creates a new instance of BrowserException with given message.
   *
   * @param message cause of exception
   */
  public BrowserException(final String message) {
    super(message);
  }
}
```

F.2.3. BrowserProperties

```
// £Id: BrowserProperties.java,v 1.36 2007/05/04 12:46:33 khe Exp £
package kiel.browser;

import java.awt.Color;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.text.CharacterIterator;
import java.text.StringCharacterIterator;
import java.util.Observable;
import java.util.Observer;

import javax.swing.KeyStroke;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import kiel.browser.model.BrowserModel;

import kiel.preferences.TypeConverters;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;

import kiel.browser.xmlconf.Configuration;
import org.w3c.dom.Document;

/**
 * <p>
 * Description: .
 * </p>
 *
 * <p>
 * Copyright: Copyright (c) 2005
 * </p>
 *
 * <p>
 * Company: Uni Kiel
 * </p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 *   modified by <a href="mailto:khe@informatik.uni-kiel.de">Karsten Heymann</a>
 * @version £Revision: 1.36 £ last modified £Date: 2007/05/04 12:46:33 £
 *
 */
public final class BrowserProperties {

    /**
     * Store references to xml-related information.
     */
    private static XMLPreferences xmlprefs;

    /**
     * jaxb context.
     */
    private  static JAXBContext jc;

    /**
     * The <code>kiel.util.LogFile</code> object for browser properties.
     */
    private static LogFile logger;

    /**
     * The Configuration object, unmarshalled from XML.
     */
    private static Configuration conf;

    static {
        xmlprefs = XMLPreferences.createInstance("browser",
                BrowserProperties.class);

        logger = xmlprefs.getLogger();

        try {
            jc = JAXBContext.newInstance("kiel.browser.xmlconf",
                    BrowserProperties.class.getClassLoader());
        } catch (JAXBException je) {
            je.printStackTrace();
        }

        if (!reload()) {
            logger.log(LogFile.ERROR, "User Preferences for module browser "
                    + "could not be initialized. Aborting!");
            System.exit(1);
        }

        Observer browserObserver = new Observer() {
            public void update(final Observable o, final Object arg) {
                if (arg.equals(XMLPreferences.PREFERENCES_REMARSHAL)) {
                    XMLPreferences.log(LogFile.DEBUG,
                            "Reloading Browser Properties");
                    BrowserProperties.reload();
                }
            }
        };
        xmlprefs.addObserver(browserObserver);
    }

    /**
     * reload preferences.
     * @return success
     */
    public static boolean reload() {
        try {
            XMLPreferences.log(LogFile.INFO, "Reloading Browser");
            XMLPreferences.log(LogFile.INFO, "Schema: "
                    + xmlprefs.getXMLSchemaURL());
            XMLPreferences.log(LogFile.INFO, "Schema: "
                    + xmlprefs.getXMLPreferencesURL());
            Unmarshaller u = jc.createUnmarshaller();
            Document doc = xmlprefs.getPreferences();
            //XMLPreferences.log(LogFile.INFO, "XML Doc: " + doc);
            conf = (Configuration) u.unmarshal(doc);
```

```
            return true;
        } catch (JAXBException je) {
            logger.log(LogFile.ERROR, "Error parsing "
                    + xmlprefs.getXMLPreferencesURL() + ": "
                    + je.getLinkedException().getMessage());
            return false;
        }
    }

    /**
     * Model needed for sending Warnings.
     */
    private static BrowserModel model = null;

    /**
     * Not for use.
     */
    private BrowserProperties() {
    }

    /**
     * @return boolean true if want to do Animations
     */
    public static boolean doAnimations() {
        return conf.isBrowserAnimation();
    }

    /**
     * @return value of animation start (in seconds)
     *
     */
    public static int getAnimationDuration() {
        final int minDur = 100;
        int dur = conf.getAnimationDuration();
        if (dur < minDur) {
            dur = minDur;
        }
        return dur;
    }

    /**
     * @return time that thread should sleep between microsteps
     *
     */
    public static int getMicroStepSleepTime() {
        final int minSleepTime = 20;
        int stime = conf.getMicroStepSleepTime();
        if (stime < minSleepTime) {
            stime = minSleepTime;
        }
        return stime;
    }

    /**
     * @return String with name for browser tools
     */
    public static String getBrowserTitle() {
        return conf.getBrowserTitle();
    }

    /**
     * @return Color to mark nodes in
     */
    public static Color getNodeMarkColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTreeMarkNodeColor());
    }

    /**
     * @return Color to mark nodes in
     */
    public static Color getNodeMarkBrightColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTreeMarkNodeBrightColor());
    }

    /**
     * @return Color to mark nodes in
     */
    public static Color getEdgeMarkColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTreeMarkEdgeColor());
    }

    /**
     * @return Color to mark unparsed Labels in estudio sim mode.
     */
    public static Color getStringLabelColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserCanvasStringLabelColor());
    }

    /**
     * @return Color to show
     */
    public static Color getInputEventColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTablesInputEventColor());
    }

    /**
     * @return Color to show
     */
    public static Color getOutputEventColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTablesOutputEventColor());
    }

    /**
     * @return Color to show
     */
    public static Color getLocalEventColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTablesLocalEventColor());
    }

    /**
     * @return Color to show
     */
```

```
    public static Color getVariablesColor() {
        return TypeConverters.parseStringToColor(
                conf.getBrowserTablesVariablesColor());
    }

    /**
     * @return String
     */
    public static String getExecuteTransition() {
        return conf.getMicroStepExecuteTransition();
    }

    /**
     * @return String
     */
    public static String getTestTransition() {
        return conf.getMicroStepTestTransition();
    }

    /**
     * @return String
     */
    public static String getOnInside() {
        return conf.getMicroStepOnInside();
    }

    /**
     * @return String
     */
    public static String getOnExit() {
        return conf.getMicroStepOnExit();
    }

    /**
     * @return String
     */
    public static String getOnEntry() {
        return conf.getMicroStepOnEntry();
    }

    /**
     * @return String
     */
    public static String getStateSuspended() {
        return conf.getMicroStepStateSuspended();
    }

    /**
     * @return String
     */
    public static String getStateActivated() {
        return conf.getMicroStepStateActivated();
    }

    /**
     * @return String
     */
    public static String getStateDeactivated() {
        return conf.getMicroStepStateDeactivated();
    }

    /**
     * @return String
     */
    public static String getActualConfig() {
        return conf.getMicroStepActualConfigState();
    }

    /**
     * This flag allows to overwrite the Browser.AlwaysStepThroug setting.
     */
    private static boolean stepThroughOverWriter = true;

    /**
     * Sets the Browser.AlwaysStepThroug setting to b.
     *
     * @param b
     *            boolean
     */
    public static void setStepThrough(final boolean b) {
        stepThroughOverWriter = b;
    }

    /**
     * @return boolean if always doing wait steps
     */
    public static boolean doStepThrough() {
        return conf.isBrowserAlwaysStepThrough() && stepThroughOverWriter;
    }

    /**
     * @return boolean true if the interface should be shown
     */
    public static boolean doShowInterface() {
        return conf.isBrowserShowInterface();
    }

    /**
     * @return boolean true if always a priority should be shown.
     */
    public static boolean doAlwaysShowPriorities() {
        return conf.isBrowserAlwaysShowPrios();
    }

    /**
     * @return boolean true if the statechart tooltips should be shown
     */
    public static boolean doShowTooltip() {
        return conf.isBrowserShowTooltip();
    }

    /**
     * @return boolean true if the interface should be shown
     */
    public static boolean animateConfigsInMicroSteps() {
        return conf.isBrowserAnimateConfigInMicroSteps();
    }

    /**
     * @return int with logLevel
```

```
     */
    public static int getLogLevel() {
        return conf.getBrowserLogLevel();
    }

    /**
     * @return String
     */
    public static String getLogLevelCompareOp() {
        return conf.getBrowserLogCompare();
    }

    /**
     * @return int Font size for internal editor.
     */
    public static int getKitEditorFontSize() {
        return conf.getKitEditorFontSize();
    }

    /**
     * @return int threshold for parser updates.
     */
    public static int getUpdateThreshold() {
        return conf.getParserThreadThreshold();
    }

    /**
     * @return char
     */
    public static char getTraceFastForwardMnemonic() {
        return conf.getBrowserMnemonicTraceFastForward().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTraceRewindMnemonic() {
        return conf.getBrowserMnemonicTraceRewind().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTraceStopMnemonic() {
        return conf.getBrowserMnemonicTraceStop().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTracePlayMnemonic() {
        return conf.getBrowserMnemonicTracePlay().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTraceStepForwardMnemonic() {
        return conf.getBrowserMnemonicTraceStepForward().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTraceStepBackMnemonic() {
        return conf.getBrowserMnemonicTraceStepBack().charAt(0);
    }

    /**
     * @return char
     */
    public static char getStateMenuMnemonic() {
        return conf.getBrowserMnemonicState().charAt(0);
    }

    /**
     * @return char
     */
    public static char getTransitionMenuMnemonic() {
        return conf.getBrowserMnemonicTransition().charAt(0);
    }

    /**
     * @return char
     */
    public static char getViewMenuMnemonic() {
        return conf.getBrowserMnemonicView().charAt(0);
    }

    /**
     * @return char
     */
    public static char getStatechartMenuMnemonic() {
        return conf.getBrowserMnemonicStatechart().charAt(0);
    }

    /**
     * @return char
     */
    public static char getEditModeMenuMnemonic() {
        return conf.getBrowserMnemonicEditMode().charAt(0);
    }

    /**
     * @return char
     */
    public static char getMacroStepMnemonic() {
        return conf.getBrowserMnemonicMacroStep().charAt(0);
    }

    /**
     * @return char
     */
    public static char getMicroStepMnemonic() {
        return conf.getBrowserMnemonicMicroStep().charAt(0);
    }

    /**
     * @return char
     */
```

```
    public static char getResetMnemonic() {
        return conf.getBrowserMnemonicReset().charAt(0);
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getCountElementsAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeCountElements());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getAlwaysFitScreenAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeAlwaysFitScreen());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getFitScreenAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeFitScreen());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getResetZoomAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeResetZoom());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getNewStatechartAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeNewStatechart());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getSaveKitAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeSaveKit());
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getLoadTraceAccelerator() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeLoadTrace());
    }

    /**
     *
     * @return KeyStroke
     */
    public static KeyStroke getChangeNodeLabel() {
        return TypeConverters.getKeyStrokeFromString(
                conf.getBrowserKeyStrokeChangeNodeLabel());
    }

    /**
     * @param m
     *            BrowserModel to send warnings to
     */
    public static void setBrowserModel(final BrowserModel m) {
        model = m;
        LogFile l = model.getBrowserLogFile();
        // l.setLogLevel(LogFile.DEBUG);
        xmlprefs.setLogger(l);
    }

    /**
     * @return String name of layouter benchmark file.
     */
    public static String getLayouterBenchName() {
        return conf.getBrowserLayouterBenchFile();
    }

    /**
     * @return String name of drawing benchmark file.
     */
    public static String getDrawingBenchName() {
        return conf.getBrowserDrawingBenchFile();
    }

    /**
     * @return String name of simulator benchmark file.
     */
    public static String getSimBenchName() {
        return conf.getBrowserSimulatorBenchFile();
    }

    /**
     * @return boolean if should create benchmark file.
     */
    public static boolean benchmarkLayouter() {
        return conf.isBrowserDoLayouterBench();
    }
    /**
     * @return boolean if should create benchmark file.
     */
    public static boolean benchmarkDrawing() {
        return conf.isBrowserDoDrawingBench();
    }
    /**
     * @return boolean if should create benchmark file.
     */
    public static boolean benchmarkSim() {
        return conf.isBrowserDoSimulatorBench();
    }
    /**
```

```
     * @return boolean if should create simout file.
     */
    public static boolean writeSimOutput() {
        return conf.isBrowserDoSimulatorOutput();
    }

    /**
     * @return value of the browserHighlightActiveStates property.
     */
    public static boolean isHighlightActiveStates() {
        return conf.isBrowserHighlightActiveStates();
    }

    /**
     * Sets the value of the browserHighlightActiveStates property.
     * @param value boolean
     */
    public static void setHighlightActiveStates(final boolean value) {
        conf.setBrowserHighlightActiveStates(value);
    }

    /**
     * @return value of the ColorizeNewConfigurations property.
     */
    public static boolean isColorizeNewConfigurations() {
        return conf.isBrowserColorizeNewConfigurations();
    }

    /**
     * Sets the value of the ColorizeNewConfigurations property.
     * @param value boolean
     */
    public static void setColorizeNewConfigurations(final boolean value) {
        conf.setBrowserColorizeNewConfigurations(value);
    }


    /**
     * Compute function name of the getter function used to fetch the
     * KeyStroke from the XML configuration that corresponds to the
     * action.
     * @param action String describing the action.
     * @return String representing a getter function name.
     */
    private static String getKeyStrokeFunctionName(final String action) {
        StringBuffer result = new StringBuffer("get");
        StringCharacterIterator iter = new StringCharacterIterator(action);
        for (char c = iter.first(); c != CharacterIterator.DONE;
        c = iter.next()) {
            if (c != '.') {
                result.append(c);
            }
        }
        return result.toString();
    }

    /**
     *
     * @param key String representing an config key
     * @return KeyStroke of that key
     */
    public static KeyStroke getKeyStroke(final String key) {
        XMLPreferences.log(LogFile.DEBUG, "Trying to get KeyStroke for action "
                + key);
        String result = "";
        Class[] parameterTypes = new Class[] {};
        String methodname = getKeyStrokeFunctionName(key);
        Method m;
        Object[] arguments = new Object[] {};
        try {
            m = conf.getClass().getMethod(methodname, parameterTypes);
            result = (String) m.invoke(conf, arguments);
        } catch (ClassCastException e) {
            XMLPreferences.log(LogFile.ERROR, "Could not cast to KeyStroke: "
                    + e.getMessage());
        } catch (SecurityException e) {
            XMLPreferences.log(LogFile.ERROR, "Not allowed to look up "
                    + "KeysStroke for " + key + ": "  + e.getMessage());
        } catch (NoSuchMethodException e) {
            XMLPreferences.log(LogFile.ERROR, "KeysStroke for Action " + key
                    + " not found: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            XMLPreferences.log(LogFile.ERROR, "Illegal argument while looking "
                    + "up KeyStroke for " + key + ":" + e.getMessage());
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            XMLPreferences.log(LogFile.ERROR, "Illegal access while looking "
                    + "up KeyStroke for " + key + ":" + e.getMessage());
        } catch (InvocationTargetException e) {
            XMLPreferences.log(LogFile.ERROR, "Illegal invocation while looking"
                    + " up KeyStroke for " + key + ":" + e.getMessage());
        }
        return KeyStroke.getKeyStroke(result.replace('+', ' '));
    }
}
```

F.3. kiel.browser.model

F.3.1. BrowserModel

```
// £Id: BrowserModel.java,v 1.109 2007/06/04 06:58:19 khe Exp £
package kiel.browser.model;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import javax.swing.text.Document;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Edge;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.eventexp.Signal;
import kiel.configMngr.Configuration;
import kiel.configMngr.MacroStep;
import kiel.configMngr.MicroStep;
import kiel.configMngr.SignalValue;
import kiel.configMngr.VariableValue;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;
import kiel.browser.BrowserException;
import kiel.browser.BrowserProperties;
import kiel.browser.view.piccolo.PiccoloToolkit;
import kiel.browser.view.rendering.Toolkit;
import kiel.simulationTrace.TraceData;
import kiel.simulationTrace.TraceStep;
import kiel.simulator.Simulator;
import kiel.simulator.SimulatorChoser;
import kiel.simulator.SimulatorException;
import kiel.util.Benchmark;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;
import kiel.util.MatlabStateflowCreator;
import kiel.util.MatlabStateflowCreatorException;
import kiel.util.languages.StatechartLanguage;
import kiel.util.main.KielLayouter;
import kiel.dataStructure.State;

/**
 * <p>Description: This is the main Browser class that contains manages all data
 * the browser needs to show.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version <p>£Revision: 1.109 £ last modified £Date: 2007/06/04 06:58:19 £</p>
 *
 * <p>
 * £Log: BrowserModel.java,v £
 * Revision 1.109  2007/06/04 06:58:19  khe
 * Make KIEL GUI work if the KIEL directory is read-only.
 *
 * Revision 1.108  2007/05/04 07:09:33  khe
 * new browser property HighlightActiveStates to replace boolean parameter
 * of the NEW_CONFIGURATION message.
 *
 * Revision 1.107  2007/05/01 07:58:47  khe
 * eliminate checkstyle warning
 *
 * Revision 1.106  2007/03/08 10:49:22  khe
 * * kielcmd can be called from any directory
 *
 * * kielcmd can read (additional) options from job file
 *
 * * kielcmd is more quiet now by default
 *
 * * kielcmd was added a verbose cmdline parameter
 *
 * * kielcmd can create ps files too (same content as eps files)
 *
 * * kielcmd now by default creates output files in current directory
 *   and not in the directory of the input file
 *
 * * kielcmd can select active states and highlight them. the highlighting
 *   can be disabled by a cmd line switch (Closes #278)
 *
 * * BrowserModel.setConfiguration got an optional extra parameter to
 *   switch on/off colorizing of created view.
 *
 * * kielcmd has a option to set any KIEL property (Closes #258)
 *
 * * XMLPreferences caches changed options for modules not yet initialized
 *   (this allows setting properties for not yet loaded modules)
 *
 * * System property kiel.basedir is honoured by FileInterface, Esterel file
 *   plugin and Layouter to normalize relative paths
 *
 * * fileInterface now checks if the statechart file to be read does exist
 *   and throws an exception if not.
 *
 * * property classes: JAXBContext.newInstance is called with explicit
 *   classloader to prevent loading problems on esterel fileInterface plugin
 *   (Closes #285)
```

```
 *
 * Revision 1.105  2007/03/05 09:16:16  khe
 * * updated Kiel property list
 * * kieltex.sty now actually changes settings
 * * kielcmd.sh fixed (added missing libraries)
 * * KielCmd: added job files, generic option setting and (nonworking)
 *   configuration setting
 *
 * Revision 1.104  2007/02/23 11:46:24  khe
 * * Log error messages only by default
 *
 * Revision 1.103  2007/02/08 13:55:19  khe
 * * changed imports from kiel.base to kiel.preferences
 * * fixed classpaths wrt kiel.base
 * * fixed build dependencies wrt kiel.base
 *
 * Revision 1.102  2007/02/06 18:23:20  khe
 * * new xml based preferences system
 * * new preferences dialog
 * * moved preferences and logfile to new module base to resolve circular build
     time
 *   dependency between util.jar and graphicalInformations.jar
 *
 * Revision 1.101  2006/05/22 17:14:32  miwi
 * *** empty log message ***
 *
 * Revision 1.100  2006/05/21 21:39:33  miwi
 * *** empty log message ***
 *
 * Revision 1.99  2006/05/21 20:24:02  miwi
 * bug fixing<br>
 *
 * Revision 1.98  2006/05/17 15:35:01  miwi
 * bug fixing 224, 219<br>
 *
 * Revision 1.97  2006/05/14 23:35:32  miwi
 * Node-Edge-Properties<br>
 *
 * Revision 1.96  2006/05/12 21:04:39  miwi
 * bug fixing<br>
 *
 * Revision 1.95  2006/04/30 12:14:13  khe
 * restored accidentially deleted files
 *
 * Revision 1.93  2006/04/18 19:28:28  miwi
 * *** empty log message ***
 *
 * Revision 1.92  2006/03/26 20:21:10  miwi
 * bugfixing...
 *
 *
 * Revision 1.88  2006/03/07 19:49:13  miwi
 * bug 186 removed<br>
 *
 * Revision 1.87  2006/02/21 14:39:26  miwi
 * enh 174 removed<br>
 *
 * Revision 1.86  2006/02/20 13:31:27  miwi
 * Highlight code changed<br>
 *
 * Revision 1.82  2006/01/31 11:31:15  miwi
 * Iterpreter added
 *
 * Revision 1.80  2006/01/18 07:28:45  miwi
 * changed flag added for matlab<br>
 *
 * Revision 1.79  2006/01/16 18:00:16  miwi
 * Optimzed imports
 *
 * Revision 1.76  2005/12/15 16:18:34  miwi
 * Time measures
 *
 * Revision 1.74  2005/12/13 14:08:42  miwi
 * New update functions
 *
 * Revision 1.72  2005/11/30 12:04:40  miwi
 * Better ParserThread handling
 *
 * Revision 1.71  2005/11/29 16:44:01  miwi
 * Better Status Line
 *
 * Revision 1.70  2005/11/28 10:41:25  miwi
 * Browser Version 2.0 RC1 import
 *
 * Revision 1.65  2005/10/05 14:25:18  apo
 * solved problem with 'currentConf' beeing null<br>
 *
 * Revision 1.64  2005/09/30 12:53:30  apo
 * added support for new variable types (int16, int8, uint32, uint16, uint8)
 * and constants<br>
 *
 * Revision 1.60  2005/09/13 08:16:42  miwi
 * New features: fit-to-pane, edge prefs etc<br>
 *
 * Revision 1.59  2005/09/10 10:50:19  miwi
 * bug fixing<br>
 *
 * Revision 1.54  2005/08/23 15:37:22  apo
 * added trace file saving feature, fixed trace bugs<br>
 *
 * Revision 1.53  2005/08/22 14:01:31  apo
 * implemented trace step button functionality except for playback<br>
 *
 * Revision 1.52  2005/08/17 12:27:01  apo
 * completed trace file loading, extended browser toolbar<br>
 *
 * Revision 1.50  2005/07/28 17:18:50  apo
 * added input event simulation support for Matlab/Stateflow-charts<br>
 *
 * Revision 1.49  2005/07/26 15:48:55  apo
 * fixed a previously introduced bug in 'nextStep'<br>
 *
 * Revision 1.48  2005/07/26 13:31:16  apo
 * added support for input and output variables.<br>
 *
 * Revision 1.47  2005/07/22 09:39:49  apo
 * expanded support for variables of type 'double' and 'float',
 * including support for future variable types<br>
 *
 * Revision 1.40  2005/06/04 15:25:26  miwi
```

```
 * Actions are shown in browser<br>
 *
 * Revision 1.39  2005/06/02 09:37:39  miwi
 * bug 83 removed<br>
 *
 * Revision 1.38  2005/05/27 10:52:20  miwi
 * On demand added<br>
 *
 * Revision 1.37  2005/05/24 17:17:48  miwi
 * New SVG Export added<br>
 *
 * Revision 1.36  2005/05/12 15:12:49  miwi
 * bug 57 removed<br>
 *
 * Revision 1.35  2005/04/13 11:03:58  miwi
 * removed . from export message<br>
 *
 * Revision 1.34  2005/04/11 17:05:54  miwi
 * Removed deprecated method calls to layouter<br>
 *
 * Revision 1.33  2005/04/04 08:50:11  miwi
 * copy defaults/better exception/better log<br>
 *
 * Revision 1.32  2005/03/24 09:21:25  miwi
 * switch to MacroStep after missing simulator exception<br>
 *
 * Revision 1.31  2005/03/23 13:05:48  miwi
 * Automatic step-wait-step on macrostep possibly<br>
 *
 * Revision 1.30  2005/03/23 12:17:37  miwi
 * more Properties better color handling<br>
 *
 * Revision 1.27  2005/03/17 07:29:57  miwi
 * Code Review<br>
 *
 * Revision 1.26  2005/03/16 09:11:59  miwi
 * Style checked<br>
 *
 * Revision 1.25  2005/03/10 11:56:05  miwi
 * new LayoutException integrated<br>
 *
 * Revision 1.24  2005/03/10 11:26:15  miwi
 * Better export system<br>
 *
 * Revision 1.23  2005/03/10 11:03:49  miwi
 * Eps output works again<br>
 *
 * Revision 1.22  2005/03/10 07:52:52  miwi
 * Output instead OutPut<br>
 *
 * Revision 1.21  2005/03/04 08:00:06  miwi
 * Charts that contain Stringlabels can not be simulated<br>
 *
 * Revision 1.20  2005/03/01 08:55:58  miwi
 * Mouse Wheel for zooming needs CTRL Key<br>
 *
 * Revision 1.19  2005/02/28 11:42:21  miwi
 * Automatic Table Resizing<br>
 *
 * Revision 1.18  2005/02/26 07:06:32  miwi
 * New feature marking in trees<br>
 *
 * Revision 1.17  2005/02/25 12:39:21  miwi
 * Smaller font if pt is removed in svg<br>
 *
 * Revision 1.16  2005/02/25 10:08:40  miwi
 * New feature: scrollbars/zooming <br>
 *
 * Revision 1.15  2005/02/23 17:15:06  miwi
 * Uninitialised signal support<br>
 *
 * Revision 1.14  2005/02/08 18:37:08  miwi
 * first support for valued signals in tables<br>
 *
 * Revision 1.13  2005/02/08 14:51:11  miwi
 * disabled buttons after simulator error<br>
 *
 * Revision 1.12  2005/02/08 12:10:09  miwi
 * changing to expressionTable<br>
 *
 * Revision 1.11  2005/02/08 11:36:25  miwi
 * missing reset in BrowserModel added<br>
 *
 * Revision 1.10  2005/02/08 11:15:30  miwi
 * corrected refresh problem in tables<br>
 *
 * Revision 1.9  2005/02/07 13:49:21  miwi
 * Signal Table for local events<br>
 *
 * Revision 1.8  2005/02/06 18:49:20  miwi
 * Better style/new message system/support for finaland/or states<br>
 *
 * Revision 1.7  2005/02/01 18:48:28  miwi
 * new signaltable handling<br>
 *
 * Revision 1.6  2005/01/31 14:15:44  miwi
 * Review changes applied<br>
 *
 * Revision 1.5  2005/01/31 10:40:07  miwi
 * Style checked release<br>
 *
 * Revision 1.4  2005/01/31 09:54:31  miwi
 * Style checked release<br>
 *
 * Revision 1.3  2005/01/26 12:12:50  miwi
 * Font handling improved/reloading properties for every statechart<br>
 *
 * Revision 1.2  2005/01/25 12:47:18  miwi
 * Better Font handling / better style<br>
 *
 * Revision 1.1  2005/01/18 16:31:58  miwi
 * New package model<br>
 *
 * Revision 1.35  2005/01/07 18:15:35  miwi
 * Happy new Year Commit :-)<br>
 *
 * Revision 1.34  2004/12/20 01:16:56  flu
 * restore last version<br>
 *
 * Revision 1.32  2004/12/08 09:02:12  miwi
```

```
     * Oberseminar Demo Version<br>
     *
     * Revision 1.31  2004/12/06 19:52:14  miwi
     * MicroStep handling<br>
     *
     * Revision 1.30  2004/12/06 19:14:38  miwi
     * New Microstep list handling<br>
     *
     * Revision 1.29  2004/12/03 16:13:09  miwi
     * New MacroStep-Microstep interface<br>
     *
     * Revision 1.28  2004/11/30 20:40:58  miwi
     * group in menu timing in estudio ini<br>
     *
     * Revision 1.27  2004/11/24 10:37:13  miwi
     * Animation works now always<br>
     *
     * Revision 1.26  2004/11/22 11:24:47  miwi
     * Animation alpha code<br>
     *
     * Revision 1.25  2004/11/18 19:26:24  miwi
     * First simulator control<br>
     *
     * Revision 1.24  2004/11/16 11:41:26  miwi
     * Configuration handling enabled again<br>
     *
     * Revision 1.23  2004/10/12 07:15:22  miwi
     * write and read plugins, var dec parsed,
     * signal decl parsed, action parser started<br>
     *
     * Revision 1.22  2004/10/05 06:37:15  miwi
     * new TreeView and lib as resource<br>
     *
     * Revision 1.20  2004/09/22 08:42:18  miwi
     * compilable with new datastructure<br>
     *
     * Revision 1.19  2004/09/13 10:51:27  miwi
     * Caching Nodes<br>
     *
     * Revision 1.18  2004/09/07 09:53:42  miwi
     * just one config<br>
     *
     * Revision 1.17  2004/09/01 17:21:11  miwi
     * graphical lib is working<br>
     *
     * Revision 1.16  2004/08/24 06:25:50  miwi
     * Prepared for eps output<br>
     *
     * Revision 1.15  2004/07/21 13:02:23  miwi
     * Exporting JPG File to current directory<br>
     *
     * Revision 1.14  2004/07/16 19:16:20  miwi
     * Import corrected for kiel.util<br>
     *
     * Revision 1.13  2004/07/16 19:02:17  miwi
     * Small Changes during refactoring fileInterface<br>
     *
     * Revision 1.12  2004/07/13 07:50:25  miwi
     * removed dependency from main <- browser<br>
     *
     * Revision 1.11  2004/07/13 07:19:42  miwi
     * working on position parsing for fileInterface<br>
     *
     * Revision 1.10  2004/07/12 17:50:18  miwi
     * Better fileInterface support<br>
     *
     * Revision 1.9  2004/07/12 13:51:25  miwi
     * programming interface for fileInterface is now just FileInterfaceModel<br>
     *
     * Revision 1.8  2004/07/12 08:30:28  miwi
     * Small changes for reading/writing Kiel files with fileInterface<br>
     *
     * Revision 1.6  2004/07/05 21:19:07  miwi
     * Integrating fileInterface in BrowserModel<br>
     *
     * Revision 1.5  2004/07/05 06:53:30  miwi
     * Start integrating FileInterface in BrowserModel<br>
     *
     * Revision 1.4  2004/07/04 18:56:06  miwi
     * new node naming in menubar<br>
     *
     * Revision 1.3  2004/07/04 12:01:52  miwi
     * Creating/Integrating new view: BrowserMenubar<br>
     *
     * Revision 1.2  2004/07/03 10:33:47  miwi
     * setting CVS Log Tags<br>
     * </p>
     *
     */

    public class BrowserModel
        extends Observable implements Observer{
      /**
        *  Edit mode.
        */
      public static final int EDIT = 0;
      /**
       *   Simulation mode.
       */
      public static final int SIMULATION = 1;
      /**
       *  Send this to observers if the simulator is reseted.
       */
      public static final ModelMessage SIMRESET = new ModelMessage();
      /**
       *  Send this to observers if simstep is ready for showing.
       */
      public static final ModelMessage SIMSTEP_COMPLETED = new ModelMessage();
      /**
       *  Send this to observers if simstep/playback is completed.
       */
      public static final ModelMessage SIMSTEP_STOP = new ModelMessage();
      /**
       *  Send this to observers to switch if simulation runs in macro mode.
       */
      public static final ModelMessage SIM_MODE_MACRO = new ModelMessage();
      /**
       *  Send this to observers to switch if simulation runs in micro mode.
       */
      public static final ModelMessage SIM_MODE_MICRO = new ModelMessage();
```

```
    /**
     *  Send this to observers if simulator is not available.
     */
    public static final ModelMessage NOSIM = new ModelMessage();
    /**
     *  Send this to observers to notify that new configs have been added.
     */
    public static final ModelMessage ALL_CONFIGS = new ModelMessage();
    /**
     * Send this to observers to reset all cached data reread properties and
     * redraw everything.
     */
    public static final ModelMessage RESET = new ModelMessage();
    /**
     *  Send this to observers to inform about next view to draw.
     */
    public static final ModelMessage NEW_VIEW = new ModelMessage();
    /**
     *  Send this to observers to inform about new configuration.
     */
    public static final ModelMessage NEW_CONFIGURATION = new ModelMessage();
    /**
     *  Send this to observers to inform about a new state.
     */
    public static final ModelMessage NEW_CHART = new ModelMessage();
    /**
     *  Send this to observers to inform about a new layouter.
     */
    public static final ModelMessage NEW_LAYOUTER = new ModelMessage();
    /**
     *  Send this to observers to show next microstep.
     */
    public static final ModelMessage NEW_MICRO = new ModelMessage();
    /**
     *  Send this to observers to show next macrostep.
     */
    public static final ModelMessage NEW_MACRO = new ModelMessage();
    /**
     *  Send this to observers if the user changes the drawing toolkit.
     */
    public static final ModelMessage NEW_TOOLKIT = new ModelMessage();
    /**
     *  Send this to observers there is an update languages list.
     */
    public static final ModelMessage NEW_LANGUAGE = new ModelMessage();
    /**
     *  Send this to observers to update Signal Tables (structur changed).
     */
    public static final ModelMessage TABLE_UPDATE = new ModelMessage();
    /**
     *  Send this to observers to update Signal Tables (data changed).
     */
    public static final ModelMessage TABLE_REFRESH = new ModelMessage();
    /**
     *  Send this to observers to mark a Object needed for checker interface.
     */
    public static final ModelMessage MARK_OBJECT = new ModelMessage();
    /**
     *  Send this to observers to remove a mark on Node.
     */
    public static final ModelMessage REMOVE_SELECTION = new ModelMessage();
    /**
     *  Send this to observers to show a hourclass.
     */
    public static final ModelMessage COMPUTATION_STARTED = new ModelMessage();
    /**
     *  Send this to observers to show a normal cursor.
     */
    public static final ModelMessage COMPUTATION_COMPLETED = new ModelMessage();
    /**
     *  Send this to observers to show a warning.
     */
    public static final ModelMessage WARNING = new ModelMessage();
    /**
     *  Send this to observers to show an exception.
     */
    public static final ModelMessage EXCEPTION = new ModelMessage();
    /**
     * Send this to obesrevrs to repaint the trace info in the toolbar.
     */
    public static final ModelMessage REPAINT_TRACEINFO = new ModelMessage();
    /**
     * Send this to observers to show new status.
     */
    public static final ModelMessage STATUS_CHANGED = new ModelMessage();
    /**
     * Send this to observers to show new status.
     */
    public static final ModelMessage KIT_STATUS_CHANGED = new ModelMessage();
    /**
     * Send this to observers to show new status.
     */
    public static final ModelMessage KIT_DOC_CHANGED = new ModelMessage();
    /**
     * Send this to observers to show new status.
     */
    public static final ModelMessage FOCUS_CHANGED = new ModelMessage();
    /**
     * Send this to obeservers if the state chart is updated
     * (not a new one just edited).
     */
    public static final ModelMessage UPDATE_CHART = new ModelMessage();
    /**
     * Send this to obeservers if the browser mode change
     */
    public static final ModelMessage MODE_CHANGE = new ModelMessage();
    /**
     * The actual statechart.
     */
    private StateChart chart;
    /**
     * The actual input trace.
     */
    private TraceData trace = new TraceData();
    /**
     * Current Configuration.
     */
    private Configuration currentConf = new Configuration(new ArrayList());
    /**
     * Current View.
```

```
     */
    private View currentView;
    /**
     * all Configurations.
     */
    private kiel.configMngr.Configuration[] allConfs;
    /**
     * Browser Logfile.
     */
    private LogFile browserLog;
    /**
     * current layouter.
     */
    private KielLayouter layouter = null;
    /**
     * current layouter name.
     */
    private String layouterName = "";
    /**
     * current simulator.
     * @see kiel.simulator.Simulator
     */
    private Simulator sim = null;
    /**
     *  flag if simulator is simulating.
     */
    private boolean isSimulating = false;
    /**
     * simulator log file.
     */
    private LogFile simLog = new LogFile(
        new PrintWriter(System.out), "Simulator");
    /**
     * Signal table model(input).
     */
    private SignalTableModel inputSignalTable = new SignalTableModel();
    /**
     * Signal table model(output).
     */
    private SignalTableModel outputSignalTable = new SignalTableModel();
    /**
     * Signal table model (local Events).
     */
    private SignalTableModel localSignals = new SignalTableModel();
    /**
     * Signal table model (local variables).
     */
    private SignalTableModel variables = new SignalTableModel();
    /**
     * Signal table model (input variables).
     */
    private SignalTableModel inputVariables = new SignalTableModel();
    /**
     * Signal table model (output variables).
     */
    private SignalTableModel outputVariables = new SignalTableModel();
    /**
     * Signal table model (input events).
     */
    private SignalTableModel inputEvents = new SignalTableModel();
    /**
     * Actual macrostep.
     */
    private MacroStep step = null;
    /**
     *  Actual microstep.
     */
    private MicroStep microStep;
    /**
     * Actual microstep number.
     */
    private int microStepNr = -1;
    /**
     *  The actual kit Document.
     */
    private Document kitDocument = null;
    /**
     * Selected objects.
     */
    private StatechartFocus focus = new StatechartFocus(null, null, null);
    /**
     * Status of model.
     */
    private String status = "Browser Status";
    /**
     * Status of kit document.
     */
    private String kitStatus = "kit not loaded";
    /**
     * Toolkit that does the drawing stuff.
     */
    private Toolkit toolkit = new PiccoloToolkit(this);
    /**
     * Benchmarks the layouter.
     */
    private Benchmark layouterBench = new Benchmark();
    /**
     * Changed chart flag for matlab grabber.
     */
    private boolean isChanged;
    /**
     * The current shown file, null indicates that the charts was newly
     * created, or switched from editor.
     */
    private File currentFile;
    /**
     * Flag for Browser Mode.
     */
    private int mode = EDIT;

    /**
     *  The constructor for data model sets default values.
     */
    public BrowserModel() {
      browserLog = new LogFile(new PrintWriter(System.out));
      browserLog.setLogLevel(BrowserProperties.getLogLevel());
      browserLog.setCompareOp(BrowserProperties.getLogLevelCompareOp());
      browserLog.enableLog();
      browserLog.setModuleName("Browser");
      XMLPreferences.getInstance("browser").addObserver(this);
```

```
    browserLog.log(LogFile.DEBUG, "Browser is started with "
                   + "log-level set to " + BrowserProperties.getLogLevel());

    inputSignalTable.addExpressions(new ArrayList(),
                                    SignalTableModel.INPUT);
    outputSignalTable.addExpressions(new ArrayList(),
                                     SignalTableModel.OUTPUT);
    localSignals.addExpressions(new ArrayList(), SignalTableModel.LOCAL);
    resetModel();
    layouterBench.setModuleName("LayouterBenchmark");
  }

  public void update(Observable o, Object arg) {
      if (arg.equals(XMLPreferences.PREFERENCES_UPDATED)) {
          browserLog.setLogLevel(BrowserProperties.getLogLevel());
          browserLog.setCompareOp(BrowserProperties.getLogLevelCompareOp());
      }
  }


  /**
   *
   * @return Simulator actual simulator
   */
  public final Simulator getSimulator() {
    return sim;
  }

  /**
   * @return the number of the actual microstep in the macrostep
   * -1 if we handle macrosteps instead if microsteps
   */
  public final int getMicroStepNumber() {
    if (step != null && microStepNr >= step.getMicroSteps().size()) {
      microStepNr = -1;
    }
    return microStepNr;
  }

  /**
   * @return MicroStep
   */
  public final MicroStep getMicroStep() {
    microStep = null;
    if (step != null
        && microStepNr < step.getMicroSteps().size()
        && microStepNr >= 0) {
      microStep = (MicroStep) step.getMicroSteps().get(microStepNr);
    }
    return microStep;
  }

  /**
   *
   * @return MicroStep null if there are no more steps.
   */
  public final MicroStep nextMicroStep() {
    microStep = null;
    microStepNr++;
    if (step != null
        && microStepNr < step.getMicroSteps().size()) {
      microStep = (MicroStep) step.getMicroSteps().get(microStepNr);
      setChanged();
      notifyObservers(NEW_MICRO);
    }
    return microStep;
  }

  /**
   * @return the current trace data object
   */
  public final TraceData getTrace() {
    return trace;
  }


  /**
   * Disables Simulation.
   * It is automatic enabled again if there is a simulationable chart loaded.
   */
  public final void disableSimulator() {
    sim = null;
    setChanged();
    notifyObservers(NOSIM);
  }

  /**
   * Must be called if a simulation starts.
   */
  public final void startSimulation() {
    if (sim != null) {
      resetSimulator(true);
    }

    sim = SimulatorChoser.getSimulator(chart.getModelSource(),
                                       chart.getModelVersion());
    if (sim == null) {
      browserLog.log(LogFile.ERROR,
                     "No adequate simulator available ("
                     + chart.getModelSource() + " Version "
                     + chart.getModelVersion() + ")");
      resetTables(true);
      setChanged();
      notifyObservers(NOSIM);
    } else {
      sim.setLogFile(simLog);
      simLog.log(LogFile.DETAIL,
                 "========== Setting statechart ==========");
      if (chart.getModelSource().equals("Matlab/Stateflow")
         && chart.getModelVersion().equals("6.1")
         && isChanged) {
        try {
          simLog.log(LogFile.DETAIL, "Transfering changed chart to matlab...");
          MatlabStateflowCreator.createChart(chart, currentView);
          simLog.log(LogFile.DETAIL, "... done");
        } catch (MatlabStateflowCreatorException ex1) {
          simLog.log(LogFile.ERROR, ex1.getMessage());
          setChanged();
          notifyObservers(NOSIM);
          sendExceptionMessage(ex1);
```

```
        return;
      } catch (IOException ex1) {
        simLog.log(LogFile.ERROR, ex1.getMessage());
        setChanged();
        notifyObservers(NOSIM);
        return;
      }
    }
    try {
      simLog.log(LogFile.DETAIL, "Setting Statechart in simulator:");
      sim.setStateChart(chart);
    } catch (SimulatorException ex) {
      sim = null;
      resetTables(true);
      setChanged();
      notifyObservers(NOSIM);
      sendExceptionMessage(ex);
    } catch (NullPointerException ex) {
      // @todo example chart that throws null
      sim = null;
      resetTables(true);
      setChanged();
      notifyObservers(NOSIM);
      sendExceptionMessage(new BrowserException(
          "Simulator crashed (null Pointer)"));
      ex.printStackTrace();
    }
    simLog.log(LogFile.DETAIL,
               "========== Quit setting statechart =========");
    if (sim != null) {
      if (ModelHelper.inputEventsSupport()) {
        Collection data = sim.getInput();
        Iterator it = data.iterator();
        ArrayList expressions = new ArrayList();

        while (it.hasNext()) {
          Event ev = (Event) it.next();
          expressions.add(new Event(ev.getName()));
        }
        inputEvents.addExpressions(expressions, SignalTableModel.INPUT);
      }
      isSimulating = true;
      ModelHelper.startSimWriter();
      setChanged();
      notifyObservers(SIM_MODE_MACRO);
    }
  }

}

/**
 * @param s sets a new statechart / searching for the right simulator
 */
public final void setStateChart(final StateChart s) {
  chart = s;
  isSimulating = false;
  sim = null;
  resetTables(true);
  focus.setFocus(null, null, null);
  prepareTablesFromStatechart();
  isChanged = false;
  if (BrowserProperties.benchmarkLayouter()) {
    layouterBench.printOutputToFile(new File(
            XMLPreferences.getSafeWriteDirectory() + File.separator
            + BrowserProperties.getLayouterBenchName()
               + chart.getRootNode() + ".txt"));
    layouterBench.printOutputToConsole(false);
  }
  setChanged();
  notifyObservers(TABLE_UPDATE);
  setChanged();
  notifyObservers(NEW_CHART);
  setMode(EDIT);
}

/**
 * Get all event/constants/variables data from the statechart and put it
 * into the tables.
 */
private void prepareTablesFromStatechart() {
  inputSignalTable.addExpressions(chart.getInputEvents(),
                                  SignalTableModel.INPUT);
  outputSignalTable.addExpressions(chart.getOutputEvents(),
                                   SignalTableModel.OUTPUT);
  if (chart.getAllVariables() != null
      && !chart.getAllVariables().isEmpty()) {
    variables.addExpressions(chart.getLocalVariables(),
                             SignalTableModel.VARIABLE);
  }

  if (chart.getConstants() != null
      && !chart.getConstants().isEmpty()) {
    variables.addExpressions(chart.getConstants(),
                             SignalTableModel.CONSTANT);
  }

  if (chart.getInputVariables() != null
      && !chart.getInputVariables().isEmpty()) {
    inputVariables.addExpressions(chart.getInputVariables(),
                                  SignalTableModel.INPUT_VARIABLE);
  }

  if (chart.getOutputVariables() != null
      && !chart.getOutputVariables().isEmpty()) {
    outputVariables.addExpressions(chart.getOutputVariables(),
                                   SignalTableModel.OUTPUT_VARIABLE);
  }
  Collection localVars = getLocalVariables((CompositeState) chart.
                                           getRootNode());
  if (!localVars.isEmpty()) {
    variables.addExpressions(localVars, SignalTableModel.VARIABLE);
  }
  Collection localEvents = getLocalEvents((CompositeState) chart.
                                          getRootNode());
  localEvents.addAll(chart.getLocalEvents());
  if (!localEvents.isEmpty()) {
    localSignals.addExpressions(localEvents, SignalTableModel.LOCAL);
  }
}
```

```
  /**
   * List of added objects.
   */
  private Collection objectsAdded;
  /**
   * List of removed objects.
   */
  private Collection objectsRemoved;
  /**
   * List of changed objects.
   */
  private Collection objectsChanged;


  /**
   * Updates the actual statechart.
   * @param s updated statechart
   * @param sourceIdentifier identifer that
   *                         created the updated chart
   */
  public final void updateChart(final StateChart s,
                                final String sourceIdentifier) {
    Hashtable added = new Hashtable(s.getAllObjects().size());
    Hashtable removed = new Hashtable(chart.getAllObjects().size());
    Hashtable changed = new Hashtable();
    s.getRootNode().setIDManual(chart.getRootNode().getID());
    if (!ModelHelper.hasDifferentAttributes(s.getRootNode(),
                                            chart.getRootNode())) {
      removed.remove(chart.getRootNode().getID());
      added.remove(s.getRootNode().getID());
      ModelHelper.mapChilds(chart.getRootNode(), s.getRootNode(), added,
                            removed,
                            changed);
    } else {
      ModelHelper.addSubTree(s.getRootNode(), added);
      added.put(s.getRootNode().getID(), s.getRootNode());
      ModelHelper.addSubTree(chart.getRootNode(), removed);
      removed.put(chart.getRootNode().getID(), chart.getRootNode());
    }

//    Iterator iter = added.values().iterator();
//    while (iter.hasNext()) {
//      System.out.println("To be added: "
//                         + ((GraphicalObject) iter.next()).getID());
//    }
//    iter = removed.values().iterator();
//    while (iter.hasNext()) {
//      System.out.println("To be removed: "
//                         + ((GraphicalObject) iter.next()).getID());
//    }
//    iter = changed.values().iterator();
//    while (iter.hasNext()) {
//      System.out.println("Has changed: "
//                         + ((GraphicalObject) iter.next()).getID());
//    }
    Edge fe = focus.getEdge();
    if (fe != null && removed.contains(fe.getID())) {
      fe = null;
    }
    Node nodes[] = focus.getNodes();
//    System.out.println("Node 1: " + nodes[0]);
//    System.out.println("Node 2: " + nodes[1]);
    if (nodes[0] != null && removed.containsKey(nodes[0].getID())) {
      nodes[0] = null;
    }
    if (nodes[1] != null && removed.containsKey(nodes[1].getID())) {
      nodes[1] = null;
    }
//    System.out.println("Node 1: " + nodes[0]);
//    System.out.println("Node 2: " + nodes[1]);
    focus.setFocus(nodes[0],nodes[1],fe);
    objectsAdded = added.values();
    objectsRemoved = removed.values();
    objectsChanged = changed.values();

    chart = s;
    isSimulating = false;
    isChanged = true;
    sim = null;
    chart.setChanged();
    resetTables(true);
    prepareTablesFromStatechart();
    ModelHelper.startTrace();

    setChanged();
    UPDATE_CHART.setParameter(sourceIdentifier);
    notifyObservers(UPDATE_CHART);

    setMode(EDIT);

    setChanged();
    notifyObservers(TABLE_UPDATE);
    setChanged();
    notifyObservers(SIMRESET);
  }

  /**
   * Specifies the new, changed or to be deleted objects.
   * @param toBeRemoved Collection with graphicalObjects that should be deleted
   *                    these objects must not be deleted already
   * @param added Collection with added objects
   * @param changed Collection with objects that changed: e.g label, type etc
   * @param sourceIdentifier String of class name that performed the changes
   */
  public final void setChangedObjects(final Collection toBeRemoved,
                                      final Collection added,
                                      final Collection changed,
                                      final String sourceIdentifier) {
    objectsAdded = null;
    objectsRemoved = null;
    objectsChanged = null;
    if (toBeRemoved != null && toBeRemoved.size() > 0) {
      Iterator iter = toBeRemoved.iterator();
      GraphicalObject go;
      while (iter.hasNext()) {
        go = (GraphicalObject) iter.next();
        if (go instanceof Node) {
          ((Node) go).getParent().removeSubnode((Node) go);
        } else if (go instanceof Edge) {
```

```
            ((Edge) go).getSource().getOutgoingTransitions().remove((Edge) go);
            ((Edge) go).getTarget().getIncomingTransitions().remove((Edge) go);
          }
        }
        objectsRemoved = toBeRemoved;
        Edge fe = focus.getEdge();
1070    if (fe != null && toBeRemoved.contains(fe)) {
          fe = null;
        }
        Node nodes[] = focus.getNodes();
        if (nodes[0] != null && toBeRemoved.contains(nodes[0])) {
          nodes[0] = null;
        }
        if (nodes[1] != null && toBeRemoved.contains(nodes[1])) {
          nodes[1] = null;
        }
1080    focus.setFocus(nodes[0],nodes[1],fe);
      }
      if (added != null && added.size() > 0) {
        objectsAdded = added;
      }
      if (changed != null && changed.size() > 0) {
        objectsChanged = changed;
      }
      if ((objectsRemoved != null && objectsRemoved.size() > 0)
          || (objectsAdded != null && objectsAdded.size() > 0)
1090      || (objectsChanged != null && objectsChanged.size() > 0)) {
        if (objectsRemoved == null) {
          objectsRemoved = new ArrayList();
        }
        if (objectsAdded == null) {
          objectsAdded = new ArrayList();
        }
        if (objectsChanged == null) {
          objectsChanged = new ArrayList();
        }
1100  //      Iterator iter = objectsRemoved.iterator();
      //      int i = 1;
      //      System.out.println("OBJECTS REMOVED");
      //      while (iter.hasNext()) {
      //        GraphicalObject obj = (GraphicalObject) iter.next();
      //        System.out.println("" + i + ". " + obj + " id: " + obj.getID());
      //        i++;
      //      }
      //      iter = objectsAdded.iterator();
      //      System.out.println("OBJECTS ADDED");
1110  //      i = 1;
      //      while (iter.hasNext()) {
      //        GraphicalObject obj = (GraphicalObject) iter.next();
      //        System.out.println("" + i + ". " + obj + " id: " + obj.getID());
      //        i++;
      //      }
      //      iter = objectsChanged.iterator();
      //      System.out.println("OBJECTS CHANGED");
      //      i = 1;
      //      while (iter.hasNext()) {
1120  //        GraphicalObject obj = (GraphicalObject) iter.next();
      //        System.out.println("" + i + ". " + obj + " id: " + obj.getID());
      //        i++;
      //      }
```

```
        isChanged = true;
        chart.setChanged();
        sim = null;
        resetTables(true);
        prepareTablesFromStatechart();
1130    ModelHelper.startTrace();

        setChanged();
        UPDATE_CHART.setParameter(sourceIdentifier);
        notifyObservers(UPDATE_CHART);

        setMode(EDIT);

        setChanged();
        notifyObservers(TABLE_UPDATE);
1140    setChanged();
        notifyObservers(SIMRESET);
      }
    }

    /**
     * @return Collection
     */
    public final Collection getRemovedObjects() {
      return objectsRemoved;
1150  }

    /**
     * @return Collection
     */
    public final Collection getAddedObjects() {
      return objectsAdded;
    }

    /**
1160 * @return Collection
     */
    public final Collection getChangedObjects() {
      return objectsChanged;
    }

    /**
     *
     * @param td TraceData
     */
1170 public final void setTrace(final TraceData td) {
      trace = td;
      trace.play();
      trace.startTrace();
      BrowserProperties.reload();
      resetSimulator(false);
      showStaticView();
      setChanged();
      notifyObservers(REPAINT_TRACEINFO);
    }
1180
    /**
     * @param start CompositeState for searching
     * @return Collection of all found Variables
```

```
     */
    private Collection getLocalVariables(final CompositeState start) {
      ArrayList list = new ArrayList();
      if (start.getVariables() != null && !start.getVariables().isEmpty()) {
        list.addAll(start.getVariables());
      }
      if (start.getSubnodes() != null && !start.getSubnodes().isEmpty()) {
        Iterator iter = start.getSubnodes().iterator();
        while (iter.hasNext()) {
          Node next = (Node) iter.next();
          if (next instanceof CompositeState) {
            list.addAll(getLocalVariables((CompositeState) next));
          }
        }
      }
      return list;
    }

    /**
     * @param start CompositeState for searching
     * @return Collection of all found local Events
     */
    private Collection getLocalEvents(final State start) {
      ArrayList list = new ArrayList();
      if (start.getLocalEvents() != null && !start.getLocalEvents().isEmpty()) {
        list.addAll(start.getLocalEvents());
      }
      if (start instanceof CompositeState) {
        if ((((CompositeState) start).getSubnodes() != null
            && !((CompositeState) start).getSubnodes().isEmpty()) {
          Iterator iter = ((CompositeState) start).getSubnodes().iterator();
          while (iter.hasNext()) {
            Node next = (Node) iter.next();
            if (next instanceof State) {
              list.addAll(getLocalEvents((State) next));
            }
          }
        }
      }
      return list;
    }

    /**
     * @param c all configs for the actual chart
     */
    public final void setConfigurations(final Configuration[] c) {
      allConfs = c;
      setChanged();
      notifyObservers(ALL_CONFIGS);
    }

    // changed by khe 2007-03-07
    /**
     * @param c the new configuration (View is set automaticaly).
     */
    public final void setConfiguration(final Configuration c) {
        if (c != null && !c.equals(currentConf)) {
            currentConf = c;
            if (layouter != null) {
                setView(layouter.getConfigurationView(currentConf));
                browserLog.log(LogFile.INFO, "Setting configuration to "
                        + currentConf);
            }
        } else if (BrowserProperties.benchmarkLayouter()) {
            layouterBench.start();
            layouterBench.stop();
        }
        setChanged();
        notifyObservers(NEW_CONFIGURATION);
    }
    // end change khe

    /**
     * @param view new view to show is notified to observers
     */
    public final void setView(final View view) {
        if (layouter != null && view != null) {
          if (BrowserProperties.benchmarkLayouter()) {
            layouterBench.start();
          }
          layouter.layoutView(view);
          if (BrowserProperties.benchmarkLayouter()) {
            layouterBench.stop();
          }
          browserLog.log(LogFile.INFO, "Doing on-demand layouting view: " + view);
        }
        currentView = view;
        setChanged();
        notifyObservers(NEW_VIEW);
        //  updateSelection();
    }

    /**
     * @return the actual showing view
     */
    public final View getView() {
      return currentView;
    }

    /**
     * @return the data model for the input signals table
     */
    public final SignalTableModel getInputSignalTable() {
      return inputSignalTable;
    }

    /**
     * @return the data model for the output signals table
     */
    public final SignalTableModel getOutputSignalTable() {
      return outputSignalTable;
    }

    /**
     * @return the data model for the input variables table
     */
    public final SignalTableModel getInputVariablesTable() {
      return inputVariables;
    }
```

```
    /**
     * @return the data model for the output variables table
     */
    public final SignalTableModel getOutputVariablesTable() {
      return outputVariables;
    }

    /**
     * @return the data model for the input events table
     */
    public final SignalTableModel getInputEventsTable() {
      return inputEvents;
    }

    /**
     * @return the data model for the remaining local signals table
     */
    public final SignalTableModel getLocalEventsTable() {
      return localSignals;
    }

    /**
     * @return the data model for all variables in Model
     */
    public final SignalTableModel getVariablesTable() {
      return variables;
    }

    /**
     * @return the browser logfile
     */
    public final LogFile getBrowserLogFile() {
      return browserLog;
    }

    /**
     * @return the browser logfile
     */
    public final LogFile getSimulatorLogFile() {
      return simLog;
    }

    /**
     * @return the actual statechart
     */
    public final StateChart getStateChart() {
      return chart;
    }

    /**
     * @return all configurations if set
     */
    public final Configuration[] getAllConfigurations() {
      return allConfs;
    }

    /**
     * @return the current configuration
     */
    public final Configuration getConfiguration() {
      return currentConf;
    }

    /**
     * @return true if views should mark states/transitions
     */
/*  public final boolean doHighlighting() {
      return doHighlighting;
    }
*/

    /**
     * Sends a warning message.
     * @param s the message
     */
    public final void sendWarningMessage(final String s) {
      browserLog.log(LogFile.ERROR, s);
      setChanged();
      WARNING.setParameter(s);
      notifyObservers(WARNING);
    }

    /**
     * Sends a exception message to observers.
     * @param e Exception the exception to send
     */
    public final void sendExceptionMessage(final Exception e) {
      browserLog.log(LogFile.ERROR, e.toString());
      setChanged();
      EXCEPTION.setParameter(e);
      notifyObservers(EXCEPTION);
    }

    /**
     * sends a table refresh message.
     */
    public final void sendTableRefresh() {
      setChanged();
      notifyObservers(TABLE_REFRESH);
    }

    /**
     *  Resets this data model to default state.
     */
    private void resetModel() {
      currentConf = new Configuration(new ArrayList());
      allConfs = null;
      chart = null;
      layouter = null;
      layouterName = "";
      isChanged = false;
      currentFile = null;
      try {
        Properties.reload();
      } catch (Exception ex2) {
        System.err.println("Error reloading rendering settings");
      }
    }

    /**
```

```
   * Resets the simulator.
   * @param withoutSimReset resets BrowserModel data but not the simulator.
   */
  public final void resetSimulator(final boolean withoutSimReset) {
    setChanged();
    notifyObservers(COMPUTATION_STARTED);
1430  simLog.log(LogFile.INFO, "========== Simulator reset starting ==========");
    if (sim != null && !withoutSimReset) {
      sim.reset();
    }
    clearAllTables();
    setChanged();
    notifyObservers(TABLE_REFRESH);

    step = null;
    microStepNr = -1;
1440  isSimulating = false;
    currentConf = null;
    try {
      BrowserProperties.reload();
    } catch (Exception ex2) {
      System.err.println("Error reloading rendering settings");
    }
    setChanged();
    notifyObservers(RESET);
    simLog.log(LogFile.INFO, "========== Simulator reset complete ==========");
1450  setChanged();
    notifyObservers(COMPUTATION_COMPLETED);
  }

  /**
   *
   * @return boolean true if simulation is running
   */
  public final boolean isSimulating() {
    return isSimulating;
1460  }


  /**
   * Clears all emits and values settings.
   */
  private void clearAllTables() {
    inputSignalTable.clearExp();
    outputSignalTable.clearExp();
    localSignals.clearExp();
1470  variables.clearExp();
    inputVariables.clearExp();
    outputVariables.clearExp();
    inputEvents.clearExp();
  }

  /**
   * @param includeInput boolean true if all input signals should be reseted too
   */
  private void resetTables(final boolean includeInput) {
1480  if (includeInput) {
      inputSignalTable.resetEvents();
      inputVariables.resetEvents();
      inputEvents.resetEvents();
    }
    outputSignalTable.resetEvents();
    variables.resetEvents();
    localSignals.resetEvents();
    outputVariables.resetEvents();
  }

  /**
   *
   * @return the actual macrostep
   */
  public final MacroStep getMacroStep() {
    return step;
  }

  /**
   * Updates the models macrostep object.
   * @param macro MacroStep
   */
  public final void setMacroStep(final MacroStep macro) {
    step = macro;
    setChanged();
    notifyObservers(NEW_MACRO);
  }



  /**
   * Sets the input tables according to the given step.
   * @param tstep - the step which is used to set the input tables.
   */
  public final void setInputTables(final TraceStep tstep) {
    inputSignalTable.resetEmits();
    inputEvents.resetEmits();

    Event[] ev = tstep.getEvents();
    Signal[] sg = tstep.getSignals();
    IntegerSignal[] intsig = tstep.getValuedSignals();
    VariableValue[] varval = tstep.getVariableAssignments();
    SignalValue[] sigval = tstep.getSignalAssignments();

    for (int i = 0; i < ev.length; i++) {
      if (ModelHelper.inputEventsSupport()) {
        inputEvents.emit(ev[i]);
      } else {
        inputSignalTable.emit(ev[i]);
      }
    }

    for (int i = 0; i < sg.length; i++) {
      inputSignalTable.emit(sg[i]);
    }

    for (int i = 0; i < intsig.length; i++) {
      inputSignalTable.emit(intsig[i]);
    }

    for (int i = 0; i < sigval.length; i++) {
      Integer val = sigval[i].getValue();
      IntegerSignal s = (IntegerSignal) sigval[i].getObject();
```

```
      ExpInformation info = ModelHelper.getInformationFromTables(s.hashCode());
      if ((info != null) && (val != null)) {
        inputSignalTable.emit(s);
        info.setValue(val.intValue());
      } else {
        ExpInformation[] expressions = (ExpInformation[])
                                       (inputEvents.getExpInformations()).
                                       toArray(new ExpInformation[0]);
        for (int j = 0; i < expressions.length; j++) {
          if (s.getName().startsWith(expressions[j].getEvent().getName())) {
            inputEvents.emit(expressions[j].getEvent());
            break;
          }
        }
      }
    }

    for (int i = 0; i < varval.length; i++) {
      String val = varval[i].getStringValue();
      Variable var = (Variable) varval[i].getObject();

      if (val != null) {
        ExpInformation info = ModelHelper.getInformationFromTables(
                                       var.hashCode());
        if (info != null) {
          info.setValue(val);
        }
      }

    }
  }

  /**
   *
   * @param input Collection with inputs.
   */
  public final void createTraceStep(final Collection input) {
    trace.record();
    Iterator it = input.iterator();
    Object element;
    TraceStep tstep = new TraceStep();
    while (it.hasNext()) {
      element = it.next();
      if (element.getClass().equals(Event.class)
          || element.getClass().equals(Signal.class)) {
        tstep.add((Event) element);
      } else if (element.getClass().equals(IntegerSignal.class)) {
        SignalValue sigval = new SignalValue(
            (IntegerSignal) element,
            (Integer) it.next());
        tstep.add(sigval);
      } else if (element instanceof Variable) {
        VariableValue varval = new VariableValue(
            (Variable) element,
            it.next().toString());
        tstep.add(varval);
      }
    }
    trace.addStep(tstep);
  }

  /**
   * Sets the model layouter and notifys the observers.
   * @param lay the new layouter
   * @param name name of the layouter
   */
  public final void setLayouter(final KielLayouter lay, final String name) {
    layouter = lay;
    layouterName = name;
    setChanged();
    NEW_LAYOUTER.setParameter(lay);
    notifyObservers(NEW_LAYOUTER);
  }

  /**
   * Shows static view.
   */
  public final void showStaticView() {
    if (layouter != null) {
      //  setView(null);  enable this line to force redraw instead of animation
      setView(layouter.getStaticView());
    }
  }

  /**
   * Sets simulator mode to macro or micro step mode.
   *
   * @param doMacroSteps boolean true if simulating macrosteps
   */
  public final void setMacroStepSimulation(final boolean doMacroSteps) {
    if (doMacroSteps) {
      microStepNr = -1;
      setChanged();
      notifyObservers(SIM_MODE_MACRO);
    } else {
      // switching to microstep simulation
      microStepNr = 0;
      setChanged();
      notifyObservers(SIM_MODE_MICRO);
    }
  }

  /**
   *
   * @return boolean true is model is in SIM_MODE_MACRO
   */
  public final boolean simulatingMacroSteps() {
    return (microStepNr == -1);
  }

  /**
   *
   * @param aNode Node
   * @param anotherNode Node
   * @param aEdge Edge
   */
  public final void setFocus(final Node aNode,
                             final Node anotherNode,
                             final Edge aEdge) {
    StatechartFocus oldFocus = new StatechartFocus(focus.getNodes()[0],
```

```
                                                    focus.getNodes()[1],
                                                    focus.getEdge());
      focus.setFocus(aNode, anotherNode, aEdge);
      if (focus.hasNode1Changed()
          && oldFocus.getNodes()[0] != null) {
        setChanged();
        REMOVE_SELECTION.setParameter(oldFocus.getNodes()[0].getID());
        notifyObservers(REMOVE_SELECTION);
      }
      if (focus.hasNode2Changed()
          && oldFocus.getNodes()[1] != null) {
        setChanged();
        REMOVE_SELECTION.setParameter(oldFocus.getNodes()[1].getID());
        notifyObservers(REMOVE_SELECTION);
      }
      if (focus.hasEdgeChanged()
          && oldFocus.getEdge() != null) {
        setChanged();
        REMOVE_SELECTION.setParameter(oldFocus.getEdge().getID());
        notifyObservers(REMOVE_SELECTION);
      }
//    if (BrowserProperties.getLogLevel() <= 2) {
//      System.out.println("Focus CHANGED:");
//      System.out.println("Node 1: changed? " + getFocus().hasNode1Changed()
//                         + " " + ((focus.getNodes()[0] != null)?
//                                  focus.getNodes()[0].getID() : "null"));
//      System.out.println("Node 2: changed? " + getFocus().hasNode2Changed()
//                         + " " + ((focus.getNodes()[1] != null)?
//                                  focus.getNodes()[1].getID() : "null"));
//      System.out.println("Edge  : changed? " + getFocus().hasEdgeChanged()
//                         + " " + getFocus().getEdge());
//    }
      if (focus.hasEdgeChanged() || focus.areNodesChanged()) {
        setChanged();
        notifyObservers(FOCUS_CHANGED);
      }
    }

    /**
     * @return Edge that was selected
     */
    public final Edge getSelectedTransition() {
      return focus.getEdge();
    }

    /**
     * Call this if you do a longer computation.
     * This may show some gui stuff (waiting cursor etc).
     */
    public final void startComputation() {
      setChanged();
      notifyObservers(COMPUTATION_STARTED);
    }

    /**
     * Call this if you finished a longer computation.
     * This may show some gui stuff (waiting cursor etc).
     */
    public final void completeComputation() {
      setChanged();
      notifyObservers(COMPUTATION_COMPLETED);
    }

    /**
     * @return Node first selected node
     */
    public final Node[] getSelectedStates() {
      return focus.getNodes();
    }

    /**
     *
     * @param aKitDocument Document to show.
     */
    public final void setKitDocument(final Document aKitDocument) {
      kitDocument = aKitDocument;
      setChanged();
      notifyObservers(KIT_DOC_CHANGED);
    }

    /**
     *
     * @param aStatus String
     */
    public final void setStatus(final String aStatus) {
      status = aStatus;
      setChanged();
      notifyObservers(STATUS_CHANGED);
    }

    /**
     * @return String
     */
    public final String getStatus() {
      return status;
    }

    /**
     * @param aStatus String
     */
    public final void setKitStatus(final String aStatus) {
      kitStatus = aStatus;
      setChanged();
      notifyObservers(KIT_STATUS_CHANGED);
    }

    /**
     * @return String
     */
    public final String getKitStatus() {
      return kitStatus;
    }

    /**
     * @param aToolkit Toolkit
     */
    public final void setToolkit(final Toolkit aToolkit) {
      toolkit = aToolkit;
      setChanged();
      notifyObservers(NEW_TOOLKIT);
```

```
  }

  /**
   * @param aCurrentFile File
   */
  public final void setCurrentFile(final File aCurrentFile) {
    this.currentFile = aCurrentFile;
  }

  /**
   * @return Actual used toolkit.
   */
  public final Toolkit getToolkit() {
    return toolkit;
  }

  /**
   * @return Document of actual showing kit file.
   */
  public final Document getKitDocument() {
    return kitDocument;
  }

  /**
   *
   * @return File
   */
  public final File getCurrentFile() {
    return currentFile;
  }

  /**
   * Show the SimStep.
   */
  public final void showSimStep() {
    setChanged();
    notifyObservers(SIMSTEP_COMPLETED);
  }

  /**
   * Mark the Simstep/Playback as stopped.
   */
  public final void quitSimStep() {
    setChanged();
    notifyObservers(REPAINT_TRACEINFO);
    setChanged();
    notifyObservers(SIMSTEP_STOP);
  }

  /**
   *  List of languages.
   */
  private ArrayList languages = new ArrayList();

  /**
   *
   * @param lang StatechartLanguage
   */
  public final void addLanguage(final StatechartLanguage lang) {
    if (!languages.contains(lang)) {
      languages.add(lang);
      setChanged();
      notifyObservers(NEW_LANGUAGE);
    }
  }

  /**
   * @param go GraphicalObject
   * @param css Color as css string
   */
  public final void markObjekt(final GraphicalObject go, final String css) {
    MARK_OBJECT.setParameter("" + go.getID()
                             + "||" + css);
    setChanged();
    notifyObservers(MARK_OBJECT);
  }

  /**
   * @param go GraphicalObject
   */
  public final void removeMark(final GraphicalObject go) {
    REMOVE_SELECTION.setParameter("" + go.getID());
    setChanged();
    notifyObservers(REMOVE_SELECTION);
  }

  /**
   * Clear all marks.
   */
  public final void clearAllMarks() {
    REMOVE_SELECTION.setParameter("");
    setChanged();
    notifyObservers(REMOVE_SELECTION);
  }

  /**
   *
   * @return Collection of languages.
   */
  public final Collection getLanguages() {
    return languages;
  }

  /**
   *
   * @return StatechartFocus
   */
  public final StatechartFocus getFocus() {
    return focus;
  }

  /**
   * @param aMode int new Mode
   */
  public final void setMode(int aMode) {
    if (aMode != mode) {
      switch (aMode) {
      case EDIT:
        mode = EDIT;
        break;
```

```
      default:
        mode = SIMULATION;
      }
      setChanged();
      notifyObservers(MODE_CHANGE);
    }
  }

  /**
   * Returns actual mode.
   * @return int actual mode
   */
  public final int getMode() {
    return mode;
  }

  /**
   * @return String name of layouter.
   */
  public final String getLayouterName() {
    return layouterName;
  }

}
```

F.3.2. ExpInformation

```
// £Id: ExpInformation.java,v 1.15 2006/04/30 12:14:14 khe Exp £
package kiel.browser.model;

import java.util.Comparator;

import kiel.dataStructure.Constant;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;

/**
 * <p>Description: Event / Variable information is collected/changed here.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.15 £ last modified £Date: 2006/04/30 12:14:14 £
 *
 */
public class ExpInformation {
  /**
   * true if this event is emited.
   */
  private boolean isEmited;
  /**
   * type of event INPUT / OUTPUT etc.
   */
  private byte type;
  /**
   * the event itself.
   */
  private Event event;
  /**
   * the variable itself.
   */
  private Variable var;
  /**
   * the constant itself.
   */
  private Constant con;
  /**
   *  Is eventinfo or variable info.
   */
  private boolean isEventInfo;
  /**
   *  Value of var/event.
   */
  private int value;

  /**
   * Value of variable as string.
   */
  private String stringvalue;
  /**
   *
   */
  private boolean undefined = true;

  /**
   *
   * <p>Title: Kiel Browser</p>
   *
   * <p>Description: Class for comparing expinformation for sorting.</p>
   *
   * <p>Copyright: Copyright (c) 2005</p>
   *
   * <p>Company: Uni Kiel</p>
   *
   * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
   */
  private static final class CaseInsensitiveComparator implements Comparator {
    /**
     * Compares to Strings, using
     * <code>String.compareToIgnoreCase(String)</code>.
     *
     * @param o1 the first string
     * @param o2 the second string
     * @return &lt; 0, 0, or &gt; 0 depending on the case-insensitive
     *         comparison of the two strings.
     */
    public int compare(final Object o1, final Object o2) {
      ExpInformation e1 = (ExpInformation) o1;
      ExpInformation e2 = (ExpInformation) o2;
      return e1.toString().compareToIgnoreCase(e2.toString());
    }
  }


  /**
   * A Comparator that uses <code>String.compareToIgnoreCase(String)</code>.
   *
   * @see Collator#compare(String, String)
   */
  public static final Comparator CASE_INSENSITIVE_ORDER = new
    CaseInsensitiveComparator();

  /**
   * Sets the information.
   * @param e the event
   * @param sType the event type INPUT/OUTPUT LOCAL etc
   * @param emited is this event emited
   */
  public ExpInformation(final Event e, final byte sType,
                        final boolean emited) {
    event = e;
    type = sType;
    isEmited = emited;
    isEventInfo = true;
    autoTestInitial();
  }

  /**
   * Sets the information.
```

```
     * @param v the variable
     */
    public ExpInformation(final Variable v) {
      var = v;
      con = null;
      type = SignalTableModel.VARIABLE;
      isEmited = false;
      isEventInfo = false;
      autoTestInitial();
    }

    /**
     *
     * @param v Variable
     * @param vType byte
     */
    public ExpInformation(final Variable v, final byte vType) {
      var = v;
      con = null;
      type = vType;
      isEmited = false;
      isEventInfo = false;
      autoTestInitial();
    }

    /**
     *
     * @param c Constant
     */
    public ExpInformation(final Constant c) {
      var = null;
      con = c;
      type = SignalTableModel.CONSTANT;
      isEmited = false;
      isEventInfo = false;
      autoTestInitial();
    }

    /**
     * set the event emited.
     * @param emit should the event be emited
     */
    public final void setEmited(final boolean emit) {
      isEmited = emit;
    }

    /**
     * @return the event/var
     */
    public final Object get() {
      if (isEventInfo) {
        return event;
      }
      if (var != null) {
        return var;
      }
      return con;
    }

    /**
     *
     * @return boolean true if expinfo is event
     */
    public final boolean isEvent() {
      return isEventInfo;
    }

    /**
     * @return the event
     */
    public final Event getEvent() {
      return event;
    }

    /**
     * @return the variable
     */
    public final Variable getVar() {
      return var;
    }

    /**
     * @return the constant
     */
    public final Constant getConstant() {
      return con;
    }

    /**
     * @return the type of the event
     */
    public final byte getType() {
      return type;
    }

    /**
     *
     * @return int value if this is an valued event var
     */
    public final int getValue() {
      return value;
    }

    /**
     *
     * @return String
     */
    public final String getStringValue() {
      return stringvalue;
    }

    /**
     *
     * @return boolean true if value is undefined
     */
    public final boolean isUndefined() {
      return undefined;
    }

    /**
```

```
     * @return is the event emited
     */
    public final boolean isEmited() {
      return isEmited;
    }

    /**
     *
     * @return String name if event/var
     */
    public final String toString() {
      if (isEventInfo) {
        return event.getName();
      }
      if (var != null) {
        return var.getName();
      }
      return con.getName();
    }

    /**
     *
     * @param val int value for the variable / event
     */
    public final void setValue(final int val) {
      this.value = val;
      this.stringvalue = Integer.toString(val);
      this.undefined = false;
    }

    /**
     *
     * @param val String
     */
    public final void setValue(final String val) {
      this.stringvalue = val;
      this.undefined = false;

      try {
        this.value = (int) Double.parseDouble(val);
      } catch (NumberFormatException e) {
        this.value = 0;
      }
    }

    /**
     * resets emit.
     */
    public final void reset() {
      this.isEmited = false;
    }

    /**
     * clears values and emit.
     */
    public final void clear() {
      this.isEmited = false;
      this.undefined = true;
      autoTestInitial();
    }

    /**
     * Automatic test and set if event/variable has an initial value.
     */
    private void autoTestInitial() {
      if (isEventInfo && event instanceof IntegerSignal
          && ((IntegerSignal) event).isInitialized()) {
        int val = 0;
        boolean setVal = true;
        try {
          val = Integer.parseInt(((IntegerSignal) event).getInitialValue().
                                 toIntExpString());
        } catch (NumberFormatException ex) {
          setVal = false;
        }
        if (setVal) {
          this.setValue(val);
        }
      } else if (!isEventInfo && var != null) {
        if (var.isInitialized()) {
          this.setValue(var.toExpString());
        }
      } else if ((var == null) && (con != null)) {
        this.setValue(con.toExpString());
      }
    }
  }
```

F.3.3. ModelHelper

```
package kiel.browser.model;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Hashtable;
import java.util.Iterator;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.Edge;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.Transition;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.eventexp.Signal;
import kiel.configMngr.ChangeValue;
import kiel.configMngr.Configuration;
import kiel.configMngr.MacroStep;
import kiel.configMngr.MicroStep;
import kiel.configMngr.SignalPresent;
import kiel.configMngr.SignalStatus;
import kiel.configMngr.SignalValue;
import kiel.configMngr.StateActivated;
import kiel.configMngr.VariableValue;
import kiel.simulator.Simulator;
import kiel.simulator.SimulatorChoser;
import kiel.simulator.SimulatorException;
import kiel.util.Benchmark;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;
import kiel.dataStructure.State;
import kiel.dataStructure.action.Actions;
import kiel.graphicalInformations.Properties;
import kiel.simulationTrace.TraceStep;
import kiel.browser.Browser;
import kiel.browser.BrowserProperties;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.23 £ last modified £Date: 2007/06/04 06:58:19 £
 */
public final class ModelHelper {

  /**
   * Model runnig this utitity methods for.
   */
  private static BrowserModel model = null;
  /**
   * Simulator benchmark.
   */
  private static Benchmark simBench = new Benchmark();

  /**
   * there will be no instances of this class because all methods are static.
   */
  private ModelHelper() {
  }

  /**
   * This must be called once before using all these little helpers.
   * @param aModel BrowserModel
   */
  public static void setBrowserModel(final BrowserModel aModel) {
    model = aModel;
    if (BrowserProperties.benchmarkSim()) {
      simBench.setModuleName("Simulator");
      simBench.printOutputToConsole(false);
      simBench.printOutputToFile(new File(XMLPreferences.getSafeWriteDirectory()
        + File.separator + BrowserProperties.getSimBenchName()));
    }
  }

  /**
   * Can the simulator simulate String Labels.
   * @return true if the actual simulator simulates stringlabels.
   */
  public static boolean canSimulateStringLabel() {
    if (model != null) {
      Simulator sim = model.getSimulator();
      if (sim == null) {
        sim = SimulatorChoser.getSimulator(
          model.getStateChart().getModelSource(),
          model.getStateChart().getModelVersion());
        if (sim == null) {
          return false;
        }
      }
      try {
        return sim.getFeature(SimulatorChoser.SIMULATES_STRING_LABEL);
      } catch (SimulatorException ex) {
        return false;
      }
    }
    return false;
  }

  /**
   * @param includeInput boolean true if all input signals should be reseted too
   */
```

```
  public static void resetEmitsInTables(final boolean includeInput) {
    if (includeInput) {
      model.getInputSignalTable().resetEmits();
      model.getInputEventsTable().resetEmits();
    }
    model.getOutputSignalTable().resetEmits();
    model.getLocalEventsTable().resetEmits();
    model.getVariablesTable().resetEmits();
  }

  /**
   *
   * @param input Collection of input events, variables or trace data.
   * @throws SimulatorException if there is an error
   */
  public static void createEmitsForSimlator(final Collection input) throws
    SimulatorException {
    Simulator sim = model.getSimulator();
    if (sim != null && input != null) {
      Iterator inputIterator = input.iterator();
      Event s;
      if (sim != null) {
        while (inputIterator.hasNext()) {
          Object next = inputIterator.next();
          if (next.getClass().equals(Signal.class)
              || next.getClass().equals(Event.class)) {
            s = (Event) next;
            sim.emit(s);
          } else if (next.getClass().equals(IntegerSignal.class)) {
            s = (IntegerSignal) next;
            /* An IntegerSignal has to be followed by an Integer, otherwise
             * the input is incorrect and a classCastException will occur!!!
             * We do not check whether next is of type Integer to simplify
             * the code.
             */
            if (inputIterator.hasNext()) {
              next = inputIterator.next();
              sim.emit(s, ((Integer) next).intValue());
            }
          } else if (next instanceof Variable) {
            Variable var = (Variable) next;
            /* A Variable has to be followed by a String, otherwise
             * the input is incorrect and a classCastException will occur!!!
             * We do not check whether next is of type String to simplify
             * the code.
             */
            if (inputIterator.hasNext()) {
              next = inputIterator.next();
              sim.setVariable(var, (String) next);
            }
          } else if (next.getClass().equals(SignalValue.class)) {
            /* We are processing a trace step object */
            Signal sig = (Signal) ((SignalValue) next).getObject();
            Integer val = ((SignalValue) next).getValue();
            if (val != null) {
              sim.emit(sig, val.intValue());
            } else {
              sim.emit(sig);
            }
          } else if (next.getClass().equals(VariableValue.class)) {
            /* We are processing a trace step object */
            Variable var = (Variable) ((VariableValue) next).getObject();
            String val = ((VariableValue) next).getStringValue();
            sim.setVariable(var, val);
          }
        }
      }
    }
  }

  /**
   * Checks the SignalStatus in a collection of microsteps.
   * @param c a collection of microsteps
   * @param s a signal to be tested
   * @return true if s is present in the collection c
   */
  public static boolean checkSignalStatus(final Collection c, final Signal s) {
    Iterator iter = c.iterator();
    MicroStep m = null;
    while (iter.hasNext()) {
      m = (MicroStep) iter.next();
      if (m instanceof SignalStatus) {
        if (isSignalPresent((SignalStatus) m, s)) {
          return true;
        }
      }
    }
    return false;
  }


  /**
   * Test if a signal s is present in a status m.
   * @param m the status
   * @param s the signal
   * @return true is s is present
   */
  public static boolean isSignalPresent(final SignalStatus m, final Signal s) {
    if ((m instanceof SignalPresent) && (m.getSignal().equals(s))) {
      return true;
    }
    return false;
  }

  /**
   * Does the simulator support input events in addition to input signals?
   * Well, we will get to know if we ask him!
   * @return true if the actual simulator supports input events.
   */
  public static boolean inputEventsSupport() {
    if (model.getSimulator() != null) {
      try {
        return model.getSimulator().getFeature(
          SimulatorChoser.INPUT_EVENTS_SUPPORT_LABEL);
      } catch (SimulatorException ex) {
        return false;
      }
    }
    return false;
  }
```

```
  /**
   *
   * @param id ID of the table entry
   * @return ExpInformation of the found entry else null
   */
  public static ExpInformation getInformationFromTables(final int id) {
    ExpInformation info = null;
    if (info == null) {
      info = model.getVariablesTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getLocalEventsTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getInputSignalTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getOutputSignalTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getInputVariablesTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getOutputVariablesTable().getExpInformation(id);
    }
    if (info == null) {
      info = model.getInputEventsTable().getExpInformation(id);
    }
    return info;
  }

  /**
   * Write the sim output to an external file for evaluation.
   */
  private static FileWriter simOutput = null;

  /**
   * Start the sim writer.
   */
  public static void startSimWriter() {
    if (BrowserProperties.writeSimOutput()) {
      try {
        if (simOutput != null) {
          simOutput.close();
        }
        simOutput = new FileWriter(new File(
                XMLPreferences.getSafeWriteDirectory() + File.separator
                + model.getStateChart().getRootNode().toString()
                    .replace('␣', '_') + ".eso"));
      } catch (IOException ex) {
        ex.printStackTrace();
      }
    }
  }

  /**
   * Gets the output from the simulator and the data from the macrostep
   * and emits/changes values.
   */
  public static void updateTablesFromSimOutput() {
    StringBuffer buf = new StringBuffer();
    Iterator iter = model.getSimulator().getOutput().iterator();
    while (iter.hasNext()) {
      Signal s = (Signal) iter.next();
      if (checkSignalStatus(model.getMacroStep().getMicroSteps(), s)) {
        model.getOutputSignalTable().emit(s);
        buf.append(s.getName());
        buf.append('␣');
      }
    }
    iter = model.getSimulator().getInput().iterator();
    while (iter.hasNext()) {
      Signal s = (Signal) iter.next();
      if (checkSignalStatus(model.getMacroStep().getMicroSteps(), s)) {
        if (ModelHelper.inputEventsSupport()) {
          model.getInputEventsTable().emit(s);
        } else {
          model.getInputSignalTable().emit(s);
        }
        buf.append(s.getName());
        buf.append('␣');
      }
    }
    iter = model.getLocalEventsTable().getExpInformations().iterator();
    while (iter.hasNext()) {
      ExpInformation s = (ExpInformation) iter.next();
      if (s.getEvent() instanceof Signal) {
        if (checkSignalStatus(model.getMacroStep().getMicroSteps(),
                              (Signal) s.getEvent())) {
          model.getLocalEventsTable().emit((Signal) s.getEvent());
          buf.append(s.getEvent().getName());
          buf.append('␣');
        }
      }
    }
    iter = model.getMacroStep().getMicroSteps().iterator();
    MicroStep m;
    while (iter.hasNext()) {
      m = (MicroStep) iter.next();
      if (m instanceof ChangeValue) {
        if (((ChangeValue) m).getValue() != null) {
          String v = ((ChangeValue) m).getStringValue();
          Object o = ((ChangeValue) m).getObject();
          if (o instanceof Event) {
            ExpInformation info = getInformationFromTables(((Event) o).
              hashCode());
            buf.append(((Event) o).getName());
            if (info != null) {
              info.setValue(v);
              buf.append('(');
              buf.append(v);
              buf.append(')');
              buf.append(',');
            }
          } else {
            ExpInformation info = getInformationFromTables(((Variable) o).
              hashCode());
            buf.append(((Variable) o).getName());
            if (info != null) {
```

```
              info.setValue(v);
              buf.append('(');
              buf.append(v);
              buf.append(')');
              buf.append(',');
            }
          }
        }
      }
    }
    if (buf.length() > 0) {
      buf.deleteCharAt(buf.length() - 1);
    }
    buf.append('\n');
    buf.append(';');
    buf.append('\n');
    if (BrowserProperties.writeSimOutput()) {
      try {
        simOutput.write(buf.toString());
        simOutput.flush();
      } catch (IOException ex) {
        ex.printStackTrace();
      }
    }
  }

  /**
   * calls the getMacroStep() methods and computes the actual configuration.
   * @return Configuration actual configuration in this step.
   */
  public static Configuration getConfigurationFromMacroStep() {
    Configuration conf = null;
    if (model.getMacroStep() != null
        && model.getMacroStep().getMicroSteps() != null
        && model.getMacroStep().getMicroSteps().size() > 0) {
      for (int i = model.getMacroStep().getMicroSteps().size() - 1;
                   i >= 0; i--) {
        MicroStep m = (MicroStep) model.getMacroStep().getMicroSteps().get(i);
        if (m.getClass() == StateActivated.class) {
          conf = ((StateActivated) m).getConfiguration();
          return conf;
        }
      }
    }
    return null;
  }

  /**
   * Get inputs from signal table.
   * @return ArrayList
   */
  public static ArrayList getInputFromGui() {
    ArrayList input = new ArrayList();
    if (model.getInputSignalTable() != null) {
      if (model.getInputVariablesTable() != null) {
        input.addAll(model.getInputVariablesTable().getVars());
      }
      if (model.getInputEventsTable() != null) {
        if (input.addAll(model.getInputEventsTable().getEmits())) {
          model.getInputSignalTable().resetEmits();
        }
        input.addAll(model.getInputSignalTable().getEmits());
        model.createTraceStep(input);
      }
    }
    return input;
  }

  /**
   * Creates a macroStep by invoking the nextStep method from the simulator
   * interface.
   * @param input list with emited input signals
   * @return a macroStep from the simulator
   * @see #kiel.simulator.Simulator
   */
  public static MacroStep createMacroStep(final Collection input) {
    Simulator sim = model.getSimulator();
    MacroStep step = null;

    if (BrowserProperties.benchmarkSim()) {
      simBench.start();
    }
    try {
      model.getSimulatorLogFile().log(LogFile.DETAIL,
                                      "========== Starting Computations ==========");

      ModelHelper.createEmitsForSimlator(input);
      step = sim.nextStep();
      model.getSimulatorLogFile().log(LogFile.DETAIL,
                                      "========== Ending Computations ==========");
      model.setMacroStep(step);
    } catch (SimulatorException ex) {
      model.sendExceptionMessage(ex);
      model.getSimulatorLogFile().log(LogFile.DETAIL,
                                      "========== Ending Computations ==========");
      model.getSimulatorLogFile().log(LogFile.ERROR, ex.getMessage());
      step = null;
    }
    if (BrowserProperties.benchmarkSim()) {
      simBench.stop();
    }
    return step;
  }

  /**
   * Searching for an Object using toString.
   * @param col Collection
   * @param o Object
   * @return Object
   */
  public static Object collectionContainsObject(final Collection col,
                                                final Object o) {
    Iterator iter = col.iterator();
    while (iter.hasNext()) {
      Object go = iter.next();
      if (go.getClass() == o.getClass()
          && go.toString().equals(o.toString())) {
        return go;
      }
    }
```

```
    return null;
  }

  /**
   * showing recursion with spaces.
   */
  private static String rec = "";

  /**
   * Compares the attributes from two states.
   * @param n1 Node
   * @param n2 Node
   * @return boolean
   */
  public static boolean hasDifferentAttributes(final Node n1, final Node n2) {
    if (n1.getClass() != n2.getClass()) {
      return true;
    }
    if (n1 instanceof State && n2 instanceof State) {
      if (!((State) n1).getName().trim().equals(((State) n2).getName().trim())
        || !areEqual(((State) n1).getBindAction(), ((State) n2).getBindAction())
        || !areEqual(((State) n1).getDoActivity(), ((State) n2).getDoActivity())
        || !areEqual(((State) n1).getEntry(), ((State) n2).getEntry())
        || !areEqual(((State) n1).getExit(), ((State) n2).getExit())
        || !Properties.getEventString((State) n1).trim().equals(
            Properties.getEventString((State) n2).trim())
        || !Properties.getVariablesString((State) n1).trim().equals(
            Properties.getVariablesString((State) n2).trim())
        ) {
        return true;
      }
    }
    return false;
  }

  /**
   * Compares the attributes from two edges.
   * @param n1 Node
   * @param n2 Node
   * @return boolean
   */
  public static boolean hasDifferentAttributes(final Edge e1, final Edge e2) {
    if (e1.getClass() != e2.getClass()) {
      return true;
    }
    if (e1 instanceof Transition && e2 instanceof Transition) {
      Transition t1 = (Transition) e1;
      Transition t2 = (Transition) e2;
      if (!t1.getLabel().toString().trim().equals(
        t2.getLabel().toString().trim())
        || t1.getLabel().getClass() != t2.getLabel().getClass()
        || t1.getPriority().getValue() != t2.getPriority().getValue()) {
        return true;
      }
    }
    return false;
  }

  /**
   * Compares two actions.
   * @param a1 Actions
   * @param a2 Actions
   * @return boolean
   */
  private static boolean areEqual(final Actions a1, final Actions a2) {
    return a1.toString().trim().equals(a2.toString().trim());
  }

  /**
   * Maps the macrostates and put objects in tables.
   * @param source CompositeState
   * @param target CompositeState
   * @param added Hashtable
   * @param removed Hashtable
   * @param changed Hashtable
   */
  public static void mapChilds(final CompositeState source,
                               final CompositeState target,
                               final Hashtable added,
                               final Hashtable removed,
                               final Hashtable changed) {
    if (source.getSubnodes().size() >= 0 && target.getSubnodes().size() >= 0) {
      rec += "␣";
      Hashtable ladded = new Hashtable(target.getSubnodes().size());
      Hashtable lremoved = new Hashtable(source.getSubnodes().size());
      Iterator iter = source.getSubnodes().iterator();
      getGO(lremoved, iter);
      iter = target.getSubnodes().iterator();
      getGO(ladded, iter);

      Hashtable other;
      if (ladded.values().size() < lremoved.values().size()) {
        //  iter = target.getSubnodes().iterator();
        iter = new ArrayList(ladded.values()).iterator();
        other = lremoved;
      } else {
        iter = new ArrayList(lremoved.values()).iterator();
        other = ladded;
      }
      GraphicalObject go;
      GraphicalObject go2;
      while (iter.hasNext()) {
        go = (GraphicalObject) iter.next();
        go2 = (GraphicalObject) other.get(go.getID());
        if (go2 != null
            //  && go instanceof CompositeState
            //  && go2 instanceof CompositeState
            && go2.getClass() == go.getClass()) {
          // this object is in both lists so needes no update
          lremoved.remove(go.getID());
          ladded.remove(go.getID());
          if (other == lremoved) {
            // go is from target statechart
            // go2 is from source statechart
            if (go instanceof Transition) {
              ((Transition) go).getLabel().setIDManual(
                ((Transition) go2).getLabel().getID());
              ((Transition) go).getPriority().setIDManual(
                ((Transition) go2).getPriority().getID());
            }
```

```
            } else {
              // go2 is from target statechart
              // go is from source statechart
              if (go instanceof Transition) {
                ((Transition) go2).getLabel().setIDManual(
                  ((Transition) go).getLabel().getID());
                ((Transition) go2).getPriority().setIDManual(
                  ((Transition) go).getPriority().getID());
              }
            }
//          System.out.println(rec + "Found Object in both states: "
//                             + go.toString());
            if (go instanceof Node && go2 instanceof Node
                && hasDifferentAttributes((Node) go, (Node) go2)) {
              if (other == lremoved) {
                // go is from target statechart
                if (go instanceof CompositeState) {
                  addSubTree((CompositeState) go, changed);
                }
                changed.put(go.getID(), go);
              } else {
                // go2 is from target statechart
                if (go2 instanceof CompositeState) {
                  addSubTree((CompositeState) go2, changed);
                }
                changed.put(go2.getID(), go2);
              }
            } else if (go instanceof Edge && go2 instanceof Edge
                && hasDifferentAttributes((Edge) go, (Edge) go2)) {
              if (other == lremoved) {
                // go is from target statechart
                changed.put(go.getID(), go);
              } else {
                // go2 is from target statechart
                changed.put(go2.getID(), go2);
              }
            } else if (go instanceof CompositeState
                && go2 instanceof CompositeState) {
              mapChilds((CompositeState) go, (CompositeState) go2,
                        added, removed, changed);
              if (go instanceof ANDState && go2 instanceof ANDState) {
                int size;
                DelimiterLine[] srcDelLines;
                DelimiterLine[] trgDelLines;

                if (other == lremoved) {
                  // go is from target statechart
                  // go2 is from source statechart
                  trgDelLines = (DelimiterLine[]) ((ANDState) go).
                                getDelimiterLines().toArray(
                                  new DelimiterLine[0]);
                  srcDelLines = (DelimiterLine[]) ((ANDState) go2).
                                getDelimiterLines().toArray(
                                  new DelimiterLine[0]);
                } else {
                  trgDelLines = (DelimiterLine[]) ((ANDState) go2).
                                getDelimiterLines().toArray(
                                  new DelimiterLine[0]);
                  srcDelLines = (DelimiterLine[]) ((ANDState) go).
                                getDelimiterLines().toArray(
                                  new DelimiterLine[0]);
                }
                size = Math.min(trgDelLines.length, srcDelLines.length);
                for (int i = 0; i < size; i++) {
                  trgDelLines[i].setIDManual(srcDelLines[i].getID());
                }
              }
            }
          }
          iter = ladded.values().iterator();
          Hashtable table = new Hashtable();
          while (iter.hasNext()) {
            go = (GraphicalObject) iter.next();
//          System.out.println(rec + "Added object " + go.toString());
            if (go instanceof CompositeState) {
//            System.out.println(rec + "---> Adding whole subtree ");
              addSubTree((CompositeState) go, table);
            }
          }
          added.putAll(ladded);
          added.putAll(table);

          iter = lremoved.values().iterator();
          table.clear();
          while (iter.hasNext()) {
            go = (GraphicalObject) iter.next();
//          System.out.println(rec + "Removed object " + go.toString());
            if (go instanceof CompositeState) {
//            System.out.println(rec + "---> Removing whole subtree ");
              addSubTree((CompositeState) go, table);
            }
          }
          removed.putAll(lremoved);
          removed.putAll(table);
          rec = rec.substring(1);
        }
      }

      /**
       *
       * @param root CompositeState
       * @param table Hashtable
       */
      public static void addSubTree(final CompositeState root,
                                    final Hashtable table) {
        if (root.getSubnodes().size() > 0) {
          Iterator iter = root.getSubnodes().iterator();
          getGO(table, iter);
          iter = root.getSubnodes().iterator();
          Node n;
          while (iter.hasNext()) {
            n = (Node) iter.next();
            if (n instanceof CompositeState) {
              addSubTree((CompositeState) n, table);
            }
          }
        }
      }
```

```
    }
  }
 }
}

/**
 * Starts and creates new trace.
 */
public static void startTrace() {
  model.getTrace().record();
  model.getTrace().startTrace();
  TraceStep step = new TraceStep();
  step.addReset();
  model.getTrace().addStep(step);
 }
}

/**
 *
 * @param table Hashtable
 * @param iter Iterator
 */
private static void getGO(final Hashtable table,
                          final Iterator iter) {
  Iterator iter2;
  Node n;
  Edge e;
  while (iter.hasNext()) {
    n = (Node) iter.next();
    table.put(n.getID(), n);
    if (n.getOutgoingTransitions().size() > 0) {
      iter2 = n.getOutgoingTransitions().iterator();
      while (iter2.hasNext()) {
        e = (Edge) iter2.next();
        table.put(e.getID(), e);
```

F.3.4. ModelMessage

```
package kiel.browser.model;

/**
 * <p>Description: A simple message class that can contains a string parameter
 * and an unique id for fast comparing.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.5 £ last modified £Date: 2006/04/30 12:14:14 £
 */
public class ModelMessage {
  /**
   * max type in all messages.
   */
  private static int max = 0;
  /**
   * parameter for this message.
   */
  private Object parameter;
  /**
   * type of message.
   */
  private int type;

  /**
   * ModelMessage constructor sets unique id for this message.
   */
  public ModelMessage() {
    setType(max);
    max++;
  }


  /**
   *
   * @param t int set message type to this
   */
  private void setType(final int t) {
    this.type = t;
  }

  /**
   * @param para sets string parameter
   */
  public final void setParameter(final Object para) {
    this.parameter = para;
  }

  /**
   *
   * @return type of message
   */
  public final int getType() {
    return type;
  }

  /**
   * @return Object Parameter
   */
  public final Object getParameter() {
    return this.parameter;
  }

  /**
   * Test if to messages are the same (according type).
   *
   * @param m ModelMessage
   * @return boolean
   */
  public final boolean equals(final ModelMessage m) {
    if (m.getType() == this.type) {
      return true;
    }
    return false;
  }

  /**
  * Returns the hashcode of the Message.
  * @return Returns the hashcode of the Message
  */
 public final int hashCode() {
     return this.type;
 }

 /**
  *
  * @return String of this class
  */
 public final String toString() {
   return "" + type;
 }

}
```

F.3.5. SignalTableModel

```
package kiel.browser.model;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;

import javax.swing.event.TableModelEvent;
import javax.swing.table.AbstractTableModel;

import kiel.dataStructure.Constant;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;

/**
 * <p>Description: A model for a browser event table.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 *
 * @version <p>£Revision: 1.35 £ last modified £Date: 2006/04/30 12:14:14 £</p>
 *
 * <p>
 * £Log: SignalTableModel.java,v £
 * Revision 1.35  2006/04/30 12:14:14  khe
 * restored accidentially deleted files
 *
 * Revision 1.33  2006/02/08 07:07:49  miwi
 * *** empty log message ***
 *
 * Revision 1.32  2006/01/16 18:00:16  miwi
 * Optimzed imports
 *
 * Revision 1.31  2005/10/17 15:04:29  apo
 * added menu item 'Simulation Mode'
 *
 * Revision 1.30  2005/09/30 12:53:30  apo
 * added support for new variable types (int16, int8, uint32, uint16, uint8)
 * and constants
 *
 * Revision 1.29  2005/08/23 13:54:49  apo
 * added check for null pointer in emit(Event)
 *
 * Revision 1.28  2005/07/26 13:31:16  apo
 * added support for input and output variables.
 *
 * Revision 1.27  2005/07/22 09:39:49  apo
 * expanded support for variables of type 'double' and 'float',
 * including support for future variable types
 *
 * Revision 1.26  2005/06/30 09:18:02  miwi
 * *** empty log message ***
 *
 * Revision 1.25  2005/06/06 16:30:04  miwi
 * *** empty log message ***
 *
 * Revision 1.24  2005/06/04 15:25:26  miwi
 * Actions are shown in browser<br>
 *
 * Revision 1.23  2005/03/22 12:39:51  miwi
 * *** empty log message ***
 *
 * Revision 1.22  2005/03/16 09:11:59  miwi
 * Style checked<br>
 *
 * Revision 1.21  2005/02/28 11:42:21  miwi
 * Automatic Table Resizing<br>
 *
 * Revision 1.20  2005/02/26 07:06:32  miwi
 * New feature marking in trees<br>
 *
 * Revision 1.19  2005/02/23 17:15:06  miwi
 * Uninitialised signal support<br>
 *
 * Revision 1.18  2005/02/22 09:11:25  miwi
 * Corrected label placement<br>
 *
 * Revision 1.17  2005/02/11 17:36:44  miwi
 * signal table is complete<br>
 *
 * Revision 1.16  2005/02/11 09:30:48  miwi
 * Valued signals improvements<br>
 *
 * Revision 1.15  2005/02/08 18:37:08  miwi
 * first support for valued signals in tables<br>
 *
 * Revision 1.14  2005/02/08 14:51:11  miwi
 * disabled buttons after simulator error<br>
 *
 * Revision 1.13  2005/02/08 13:48:15  miwi
 * *** empty log message ***
 *
 * Revision 1.12  2005/02/08 12:10:09  miwi
 * changing to expressionTable<br>
 *
 * Revision 1.11  2005/02/07 13:49:21  miwi
 * Signal Table for local events<br>
 *
 * Revision 1.10  2005/02/06 18:49:20  miwi
 * Better style/new message system/support for finaland/or states<br>
 *
 * Revision 1.9  2005/02/01 19:16:04  miwi
 * better style<br>
 *
 * Revision 1.8  2005/02/01 19:11:38  miwi
 * output signal cell are not editable and disabled<br>
 *
 * Revision 1.7  2005/02/01 18:58:11  miwi
 * output signal cell are not editable<br>
 *
```

```
     * Revision 1.6  2005/02/01 18:48:29  miwi
     * new signaltable handling<br>
     *
     * Revision 1.5  2005/01/31 12:15:26  miwi
     * see reference with#<br>
     *
     * Revision 1.4  2005/01/31 10:40:07  miwi
     * Style checked release<br>
     *
     * Revision 1.3  2005/01/31 09:54:31  miwi
     * Style checked release<br>
     *
     * Revision 1.2  2005/01/25 12:47:18  miwi
     * Better Font handling / better style<br>
     *
     * </p>
     */

    public class SignalTableModel
        extends AbstractTableModel {

      /**
       * Number of events column.
       */
      private static final byte NAMES = 0;
      /**
       * Number of values column.
       */
      private static final byte VALUES = 1;
      /**
       * input event table.
       */
      public static final byte INPUT = 0;
      /**
       * output event table.
       */
      public static final byte OUTPUT = 1;
      /**
       * mixed events table.
       */
      public static final byte MIXED_TABLE = 3;
      /**
       * local events table.
       */
      public static final byte LOCAL = 4;
      /**
       * local events table.
       */
      public static final byte VARIABLE = 5;
      /**
       * input variables table.
       */
      public static final byte INPUT_VARIABLE = 6;
      /**
       * output variables table.
       */
      public static final byte OUTPUT_VARIABLE = 7;
      /**
       * unknown events table.
       */
      public static final byte UNKNOWN_TABLE = 10;
      /**
       * data is a constant.
       */
      public static final byte CONSTANT = 11;
      /**
       * Hashtable for event name <--> event object.
       */
      private Hashtable exprTable = new Hashtable();

      /**
       * Shown Names.
       */
      private ArrayList actualTable = new ArrayList();

      /**
       * backup of deleted signals.
       */
      private ArrayList backedupSignals = new ArrayList();

      /**
       * type of whole table.
       */
      private int typeOfAllExpression;

      /**
       * if !isSorted list is sorted again.
       */
      private boolean isSorted = false;
      /**
       * how many name/value columns are next to another.
       */
      private int sizeValue = 1;

      /**
       * Constructs data model.
       */
      public SignalTableModel() {
        resetEvents();
      }

      /**
       * @param col number of column
       * @return class of data stored in this column
       */
      public final Class getColumnClass(final int col) {
        if (!isValueColumn(col)) {
          return ExpInformation.class;
        }
        return String.class;
      }

      /**
       * Id for a better hashtable.
       */
      private static int id = 0;

      /**
       * @param expr all events as collection or collection of signalinformation
       * @param typeOfAllSigs type of all events
```

```
 */
public final void addExpressions(final Collection expr,
                                 final byte typeOfAllSigs) {
  if (typeOfAllExpression == UNKNOWN_TABLE) {
    if (typeOfAllSigs == INPUT || typeOfAllSigs == OUTPUT
        || typeOfAllSigs == MIXED_TABLE) {
      typeOfAllExpression = typeOfAllSigs;
    }
  } else {
    if ((typeOfAllExpression == INPUT && typeOfAllSigs == OUTPUT)
        || (typeOfAllExpression == OUTPUT
            && typeOfAllSigs == INPUT)) {
      typeOfAllExpression = MIXED_TABLE;
    }
  }
  Iterator iter = expr.iterator();
  while (iter.hasNext()) {
    Object next = iter.next();
    ExpInformation s = null;
    if (next instanceof Event) {
      s = new ExpInformation((Event) next, (byte) typeOfAllSigs, false);
      exprTable.put(new Integer(s.getEvent().hashCode()), s);
      actualTable.add(s);
    } else if (next instanceof Variable) {
      s = new ExpInformation((Variable) next, typeOfAllSigs);
      exprTable.put(new Integer(s.getVar().hashCode()), s);
      actualTable.add(s);
    } else if (next instanceof Constant) {
      s = new ExpInformation((Constant) next);
      exprTable.put(new Integer(s.getConstant().hashCode()), s);
      actualTable.add(s);
    } else if (next instanceof ExpInformation) {
      s = (ExpInformation) next;
      typeOfAllExpression = MIXED_TABLE;
      /*if (s.get() instanceof Variable) {
        exprTable.put(new Integer(s.getVar().hashCode()), s);
        actualTable.add(s);
      } else {
        exprTable.put(new Integer(s.getEvent().hashCode()), s);
        actualTable.add(s);
      }*/
      exprTable.put(new Integer(s.get().hashCode()), s);
      actualTable.add(s);
    } else {
      System.err.println("Found not supported " + next.getClass());
    }
    id++;
  }
  isSorted = false;
  fireTableDataChanged();
  fireTableChanged(new TableModelEvent(this, 0, exprTable.values().size() - 1,
                                       NAMES));
}

/**
 * Removes all input signals from the table.
 */
public final void removeInputSignals() {
  ArrayList newTable = new ArrayList();
  ExpInformation expInfo;
  for (int i = 0; i < actualTable.size(); i++) {
    expInfo = (ExpInformation) actualTable.get(i);
    if (expInfo.isEvent() && (expInfo.getType() == INPUT)) {
      exprTable.remove(new Integer(
              (expInfo.getEvent().hashCode())));
      backedupSignals.add(expInfo);
    } else {
      newTable.add(expInfo);
    }
  }
  actualTable = newTable;
  isSorted = false;
  fireTableDataChanged();
  fireTableChanged(new TableModelEvent(this, 0,
          exprTable.values().size() - 1,
          NAMES));
}

/**
 * Restores all previously removed input signals.
 */
public final void restoreInputSignals() {
  ExpInformation expInfo;
  for (int i = 0; i < backedupSignals.size(); i++) {
    expInfo = (ExpInformation) backedupSignals.get(i);
    exprTable.put(new Integer(expInfo.getEvent().hashCode()), expInfo);
    actualTable.add(expInfo);
  }
  backedupSignals.clear();
  isSorted = false;
  fireTableDataChanged();
  fireTableChanged(new TableModelEvent(this, 0,
          exprTable.values().size() - 1,
          NAMES));
}

/**
 * @return all SignalInformations from this TableModel
 */
public final Collection getExpInformations() {
  return exprTable.values();
}

/**
 * resets set events.
 */
public final void resetEvents() {
  typeOfAllExpression = UNKNOWN_TABLE;
  exprTable.clear();
  actualTable.clear();
  backedupSignals.clear();
  isSorted = false;
  sizeValue = 1;
  fireTableStructureChanged();
}

/**
 * @return Number of rows here equal to number of events
 * @see #getColumnCount
 */
```

```
  public final int getRowCount() {
    if (exprTable.values().size() == 0) {
      return 0;
    } else if (sizeValue == 1) {
      return exprTable.values().size();
    }
    return (exprTable.values().size() / sizeValue) + 1;
  }

  /**
   * @param col int column number to be tested
   * @return boolean true if col is value column
   */
  public final boolean isValueColumn(final int col) {
    if (col % 2 == 0) {
      return false;
    }
    return true;
  }

  /**
   * @param message log message to Browserlog
   */
  public final void log(final String message) {
    System.err.println(message);
  }

  /**
   * Name of columns.
   */
  public static final String[] COLUMN_NAMES = {"Names", "Values"};

  /**
   * @param column number of column to know name for
   * @return name of column
   */
  public final String getColumnName(final int column) {
    if (isValueColumn(column)) {
      return COLUMN_NAMES[1];
    }
    return COLUMN_NAMES[0];
  }

  /**
   * @return 2 - first column are event names second are values
   */
  public final int getColumnCount() {
    return 2 * sizeValue;
  }

  /**
   * @param row is cell editable at this row
   * @param col is cell editable at this col
   * @return if one can edit this cell
   */
  public final boolean isCellEditable(final int row, final int col) {
    if (!isValueColumn(col)) {
      if (typeOfAllExpression == OUTPUT
          || typeOfAllExpression == VARIABLE
          || typeOfAllExpression == LOCAL) {
        return false;
      } else if (typeOfAllExpression == MIXED_TABLE && !isValueColumn(col)) {
        ExpInformation info = getSortedObject(getRowCount() * (col / 2) + row);
        if (info == null) {
          return false;
        }
        if (info.getType() == OUTPUT || info.getType() == VARIABLE
            || info.getType() == LOCAL) {
          return false;
        }
        if (info.getType() == INPUT_VARIABLE) {
          return true;
        }
      }
    } else {
      ExpInformation info = getSortedObject(getRowCount() * (col / 2) + row);
      if (info == null) {
        return false;
      }
      if (info.getType() == INPUT_VARIABLE) {
        return true;
      }
      if (info.getType() == OUTPUT || info.getType() == VARIABLE
          || info.getType() == LOCAL) {
        return false;
      } else {
        if (info.get() instanceof IntegerSignal) {
          return true;
        }
        return false;
      }
    }
    return true;
  }

  /**
   * @param val value to be set
   * @param row at this row
   * @param col at this column
   */
  public final void setValueAt(final Object val, final int row, final int col) {
    if (isValueColumn(col)) {
      ExpInformation s = getSortedObject(getRowCount() * (col / 2) + row);
      try {
        s.setValue(val.toString());
        s.setEmited(true);
        fireTableDataChanged();
      } catch (NumberFormatException ex) {
        return;
      }
      fireTableCellUpdated(row, col);
    } else {
      ExpInformation s = getSortedObject(getRowCount() * (col / 2) + row);
      s.setEmited(!s.isEmited());
      fireTableCellUpdated(row, col);
    }
  }

  /**
   * @param s Event to be tested
```

```
     * @return true is event is emited
     */
    public final boolean isEmited(final Event s) {
      ExpInformation info = (ExpInformation) exprTable.get(s.getName());
      return info.isEmited();
    }

    /**
     * @param s Event to be tested
     * @return true is event is emited
     */
    public final boolean isInput(final Event s) {
      switch (typeOfAllExpression) {
        case MIXED_TABLE:
          ExpInformation info = (ExpInformation) exprTable.get(s.getName());
          if (info.getType() == OUTPUT) {
            return false;
          }
        case INPUT:
          return true;
        case OUTPUT:
        default:
          return false;
      }
    }

    /**
     * @param s set event s to be emited
     */
    public final void emit(final Event s) {
      ExpInformation info = (ExpInformation) exprTable.get(
        new Integer(s.hashCode()));
      if (info != null) {
        info.setEmited(true);
        fireTableDataChanged();
      }
    }

    /**
     * Set all event emits to false.
     */
    public final void resetEmits() {
      Iterator iter = exprTable.values().iterator();
      while (iter.hasNext()) {
        ExpInformation info = (ExpInformation) iter.next();
        info.reset();
      }
      fireTableDataChanged();
    }

    /**
     * Set all event emits to false.
     */
    public final void clearExp() {
      Iterator iter = exprTable.values().iterator();
      while (iter.hasNext()) {
        ExpInformation info = (ExpInformation) iter.next();
        info.clear();
      }
      fireTableDataChanged();
    }

    /**
     * @return a Collection of all emited events
     */
    public final Collection getEmits() {
      ArrayList sigs = new ArrayList();
      Iterator iter = exprTable.values().iterator();
      while (iter.hasNext()) {
        ExpInformation info = (ExpInformation) iter.next();
        if (info.isEmited()) {
          sigs.add(info.getEvent());
          if (info.get() instanceof IntegerSignal) {
            if (!info.isUndefined()) {
              sigs.add(new Integer(info.getValue()));
            } else {
              System.err.println("Trying to emit an valued signal "
                                 + "that is undefined");
            }
          }
        }
      }
      return sigs;
    }

    /**
     * @return a Collection of all variables
     */
    public final Collection getVars() {
      ArrayList vars = new ArrayList();
      Iterator iter = exprTable.values().iterator();
      while (iter.hasNext()) {
        ExpInformation info = (ExpInformation) iter.next();
        if (info.get() instanceof Variable) {
           vars.add(info.get());
           vars.add(info.getStringValue());
        }
      }
      return vars;
    }

    /**
     * @param code int code of the event variable to find in this TableModel
     * @return ExpInformation or null if not found
     */
    public final ExpInformation getExpInformation(final int code) {
      return (ExpInformation) exprTable.get(new Integer(code));
    }

    /**
     *  force the table to sort names again.
     */
    public final void sortListAgain() {
      isSorted = false;
    }

    /**
     * @param row to show from sorted list
     * @return Object element at row position sorted
     */
```

```
  private ExpInformation getSortedObject(final int row) {
    if (row >= actualTable.size()) {
      return null;
    }
    if (!isSorted) {
      Collections.sort(actualTable, ExpInformation.CASE_INSENSITIVE_ORDER);
      isSorted = true;
    }
    return ((ExpInformation) exprTable.get(
      new Integer(((ExpInformation) actualTable.get(row)).get().hashCode())));
  }

  /**
   * @param row table row
   * @param col table column
   * @return Object at the row/col position
   */
  public final Object getValueAt(final int row, final int col) {
//    System.err.println("Trying to get Value at " + row + "/" + col);
//    System.err.println("Table is " + getRowCount() + "/" + getColumnCount());
    if (getRowCount() * (col / 2) + row < exprTable.values().size()) {
      if (!isValueColumn(col)) {
        return getSortedObject(getRowCount() * (col / 2) + row);
      } else { // col == VALUES
        ExpInformation info = getSortedObject(getRowCount() * (col / 2) + row);
        if (info.isEvent()) {
          if (info.getEvent() instanceof IntegerSignal) {
            if (!info.isUndefined()) {
              return "" + info.getValue();
            } else {
              return "␣–␣";
            }
          } else {
            return "–␣not␣a␣valued␣event␣–";
          }
        } else { // is variable
          if (info.getVar() != null) {
            if (!info.isUndefined()) {
              return "" + info.getStringValue();
            } else {
              return "␣–␣";
            }
          } else if (info.getConstant() != null) {
            return info.getStringValue();
          } else {
            return "–␣not␣a␣valued␣variable␣–";
          }
        }
      }
    }
    return null;
  }

  /**
   * resize the table to i "tables" next to another.
   *
   * @param i int
   */
  public final void resize(final int i) {
    sizeValue = i;
    fireTableStructureChanged();
    fireTableChanged(new TableModelEvent(this, 0, exprTable.values().size() - 1,
                                         NAMES));
  }

}
```

F.3.6. ParserThread

```
package kiel.browser.model;

import javax.swing.text.Document;

import kiel.dataStructure.StateChart;
import kiel.browser.Browser;
import kiel.browser.BrowserProperties;
import kiel.preferences.LogFile;
import kiel.util.languages.UpdateThread;
import kiel.util.main.KielLayouter;
import kiel.simulationTrace.TraceStep;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Thread that waits for new statecharts and updates
 *                 components.
 * </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.13 £ last modified £Date: 2007/02/08 13:55:19 £
 */
public class ParserThread extends Thread implements UpdateThread {
  /**
   * New statechart.
   */
  private StateChart chart;
  /**
   * Old statechart.
   */
  private StateChart oldchart;
  /**
   * Document with statechart text.
   */
  private Document doc;
  /**
   * Last time when key was pressed.
   */
  private Long lastKeyboardAction = new Long(System.currentTimeMillis());

  /**
   * Starts the thread.
   */
  public ParserThread() {
    start();
    chart = new StateChart();
    oldchart = chart;
  }

  /**
   *  Thread that waits for activation from actionPerformed method.
   *  And does the animation.
   */
  public final void run() {
    long minDur = BrowserProperties.getUpdateThreshold();
    long dur = minDur;
    while (true) {
      try {
        sleep(minDur);
      } catch (InterruptedException ex) {
        System.err.println("Wait returned with an exception " + ex);
      }
      synchronized (lastKeyboardAction) {
        dur = System.currentTimeMillis() - lastKeyboardAction.longValue();
      }
      if (dur > 2 * minDur) {
        forceUpdate();
      }
    }
  }

  /**
   * Layouts and shows the actual statechart.
   */
  public void forceUpdate() {
    synchronized (chart) {
      if (!chart.equals(oldchart)) {
        oldchart = chart;
        long start = System.currentTimeMillis();
        long start2 = System.currentTimeMillis();
        Browser.getModel().updateChart(chart, this.getClass().getName());
        Browser.getModel().setKitDocument(doc);
        System.err.println("Browser set time: "
                           + (System.currentTimeMillis() - start2) + " ms");
        start2 = System.currentTimeMillis();
        KielLayouter layouter = Browser.getKielFrame().
                                getKielLayouterByName(
                                  Browser.getKielFrame().
                                  getDefaultLayouterName());
        try {
          layouter.setStatechart(chart);
          layouter.layoutView(layouter.getStaticView());
          Browser.getModel().setLayouter(layouter,
                                         Browser.getKielFrame().
                                         getDefaultLayouterName());
          System.err.println("Layouter time: "
                             + (System.currentTimeMillis() - start2)
                             + " ms");
          start2 = System.currentTimeMillis();
          Browser.getModel().showStaticView();
          System.err.println("Piccolo Drawing time: "
                             + (System.currentTimeMillis() - start2)
                             + " ms");
          System.err.println("Complete Update time: "
                             + (System.currentTimeMillis() - start) + " ms");

        } catch (Exception ex) {
          Browser.getModel().getBrowserLogFile().log(LogFile.ERROR,
```

```
              "Cannot set statechart for layouter: "
              + ex);
            ex.printStackTrace();
          }
        }
      }
    }

  /**
   * @param c StateChart
   * @param aDoc Document
   */
  public final void setStatechart(final StateChart c, final Document aDoc) {
    synchronized (chart) {
      chart = c;
      doc = aDoc;
    }
//    System.out.println("QUIT Setting Chart " + chart.getRootNode().getID());
  }

  /**
   *
   */
  public final void notifyKeyboardAction() {
    synchronized (lastKeyboardAction) {
      lastKeyboardAction = new Long(System.currentTimeMillis());
    }
  }

}
```

F.4. kiel.browser.model.languages

F.4.1. Handler

```
package kiel.browser.model.languages;

import kiel.browser.model.BrowserModel;
import kiel.util.languages.StatechartLanguage;
import kiel.util.main.LanguageHandler;

/**
 * <p>Title: Kiel Browser,</p>
 *
 * <p>Description: Handler for registering new languages.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public class Handler implements LanguageHandler {
  /**
   * Model for data.
   */
  private BrowserModel model;
  /**
   * This is a <code>singleton</code>.
   */
  private static Handler instance;

  /**
   *
   * @param aModel BrowserModel
   */
  public Handler(final BrowserModel aModel) {
    model = aModel;
    instance = this;
  }

  /**
   * This is a <code>singleton</code>.
   *
   * @return Handler instance
   */
  public static Handler getInstance() {
    return instance;
  }

  /**
   * @param lang StatechartLanguage
   */
  public final void registerStateLanguages(final StatechartLanguage lang) {
    model.addLanguage(lang);
  }
}
```

F.4.2. KitLanguage

```
package kiel.browser.model.languages;

import java.awt.Color;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PushbackReader;
import java.util.ArrayList;

import javax.swing.JTextPane;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.DocumentFilter;
import javax.swing.text.Element;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;

import kiel.dataStructure.StateChart;
import kiel.browser.Browser;
import kiel.browser.view.MyDocumentFilter;
import kiel.util.kit.lexer.Lexer;
import kiel.util.kit.lexer.LexerException;
import kiel.util.kit.node.EOF;
import kiel.util.kit.node.TAdd;
import kiel.util.kit.node.TBind;
import kiel.util.kit.node.TBlank;
import kiel.util.kit.node.TBoolean;
import kiel.util.kit.node.TChoice;
import kiel.util.kit.node.TCombine;
import kiel.util.kit.node.TComment;
import kiel.util.kit.node.TDeephistory;
import kiel.util.kit.node.TDim;
import kiel.util.kit.node.TDo;
import kiel.util.kit.node.TDouble;
import kiel.util.kit.node.TDynamicchoice;
import kiel.util.kit.node.TEdge;
import kiel.util.kit.node.TEntry;
import kiel.util.kit.node.TExit;
import kiel.util.kit.node.TFinal;
import kiel.util.kit.node.TFloat;
import kiel.util.kit.node.TFork;
import kiel.util.kit.node.THistory;
import kiel.util.kit.node.TIdentifier;
import kiel.util.kit.node.TInitial;
import kiel.util.kit.node.TInput;
import kiel.util.kit.node.TInt;
import kiel.util.kit.node.TInteger;
import kiel.util.kit.node.TInternal;
import kiel.util.kit.node.TJoin;
import kiel.util.kit.node.TJunction;
import kiel.util.kit.node.TLBrace;
import kiel.util.kit.node.TLBracket;
import kiel.util.kit.node.TLReg;
import kiel.util.kit.node.TLabel;
import kiel.util.kit.node.TLocalevent;
import kiel.util.kit.node.TLocalvariable;
import kiel.util.kit.node.TModel;
import kiel.util.kit.node.TMult;
import kiel.util.kit.node.TNormaltermination;
import kiel.util.kit.node.TNumber;
import kiel.util.kit.node.TOutput;
import kiel.util.kit.node.TParallel;
import kiel.util.kit.node.TPriority;
import kiel.util.kit.node.TRBrace;
import kiel.util.kit.node.TRBracket;
import kiel.util.kit.node.TRReg;
import kiel.util.kit.node.TStatechart;
import kiel.util.kit.node.TString;
import kiel.util.kit.node.TStrongabortion;
import kiel.util.kit.node.TSuspend;
import kiel.util.kit.node.TSync;
import kiel.util.kit.node.TType;
import kiel.util.kit.node.TVar;
import kiel.util.kit.node.TVersion;
import kiel.util.kit.node.TWeakabortion;
import kiel.util.kit.node.TWith;
import kiel.util.kit.node.Token;
import kiel.util.languages.ILineNumber;
import kiel.util.languages.StatechartLanguage;
import kiel.util.languages.UpdateThread;
import javax.swing.event.CaretListener;
import javax.swing.event.CaretEvent;
import kiel.util.kit.node.TConditional;
import kiel.util.kit.node.TSuspension;
import javax.swing.JScrollPane;
import java.awt.Point;
import javax.swing.JComponent;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: A language class for the text-editor.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.8 £ last modified £Date: 2006/05/21 20:24:02 £
 */
public class KitLanguage implements StatechartLanguage {
  /**
   * Line number interface.
   */
  private ILineNumber line;
  /**
   * Text pane to write to.
   */
  private JTextPane pane;
  /**
   * Actual Statechart.
```

```
 */
 private StateChart chart;
 /**
  * Current File.
  */
 private File currentFile;
 /**
  * Thread that takes care of updating the browser components.
  */
 private UpdateThread ut;
 /**
  * Filter for syntax-highlighting.
  */
 private DocumentFilter filter;
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet KEYWORD = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet STD = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet STRING = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet NUMBER = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet COMMENT = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet STRUCTURE = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet IDS = new SimpleAttributeSet();
 /**
  * Style Constant.
  */
 public static final SimpleAttributeSet LEXERROR = new SimpleAttributeSet();
 /**
  * Buffer size for parser.
  */
 private static final int BUFFER = 1024;
 /**
  * Editing position.
  */
 private static int editPos;
 /**
  * Flag is true if actualy set a new chart.
  */
 private static boolean setChart = false;

 /**
  * Create Style Contants.
  */
 public KitLanguage() {
   StyleConstants.setBold(KEYWORD, true);
   StyleConstants.setForeground(KEYWORD, Color.blue);

   StyleConstants.setForeground(STRING, Color.blue.brighter().brighter());

   StyleConstants.setForeground(NUMBER, Color.orange);

   StyleConstants.setForeground(IDS, Color.magenta.darker());

   StyleConstants.setForeground(STD, Color.black);

   StyleConstants.setForeground(STRUCTURE, Color.black);

   StyleConstants.setForeground(COMMENT, Color.gray);
   StyleConstants.setItalic(COMMENT, true);

   StyleConstants.setForeground(LEXERROR, Color.red);
   StyleConstants.setUnderline(LEXERROR, true);
   editPos = 0;
 }

 /**
  * isReadOnly.
  *
  * @return boolean
  */
 public final boolean isReadOnly() {
   return true;
 }

 /**
  * setLineNumber.
  *
  * @param ln ILineNumber
  */
 public final void setLineNumber(final ILineNumber ln) {
   line = ln;
 }

 /**
  * setStatechart.
  * @param aChart StateChart
  * @param chartFile File
  */
 public final void setStatechart(final StateChart aChart,
                                 final File chartFile) {

   setChart = true;
   chart = aChart;
   currentFile = chartFile;
   BatchDocument bDoc = new BatchDocument();
   filter = new MyDocumentFilter(pane, line, ut);
   bDoc.setDocumentFilter(filter);
   File f;
   Lexer lexer = null;
   try {
     if (currentFile != null
         && currentFile.getName().endsWith(".kit")) {
```

```
        f = currentFile;
      } else {
        f = File.createTempFile("kitgen", ".kit");
        Browser.getKielFrame().writeStateChartFile(f, chart, null, true);
      }
      lexer = new Lexer(new PushbackReader(new FileReader(f),
                                           BUFFER));
      Browser.getModel().setKitStatus("Kit file created/loaded");
    } catch (Exception ex) {
      ex.printStackTrace();
    }
    Token t = null;
    boolean lf = false;
    do {
      try {
        t = lexer.next();
        if (t.getClass() == TIdentifier.class) {
          bDoc.appendBatchString(t.getText(), IDS);
        } else if (t.getClass() == TStatechart.class
                   || t.getClass() == TAdd.class
                   || t.getClass() == TBind.class
                   || t.getClass() == TBoolean.class
                   || t.getClass() == TChoice.class
                   || t.getClass() == TCombine.class
                   || t.getClass() == TConditional.class
                   || t.getClass() == TDeephistory.class
                   || t.getClass() == TDo.class
                   || t.getClass() == TDouble.class
                   || t.getClass() == TDynamicchoice.class
                   || t.getClass() == TEntry.class
                   || t.getClass() == TExit.class
                   || t.getClass() == TFinal.class
                   || t.getClass() == TFloat.class
                   || t.getClass() == TFork.class
                   || t.getClass() == THistory.class
                   || t.getClass() == TInitial.class
                   || t.getClass() == TInput.class
                   || t.getClass() == TInteger.class
                   || t.getClass() == TInternal.class
                   || t.getClass() == TJoin.class
                   || t.getClass() == TJunction.class
                   || t.getClass() == TLabel.class
                   || t.getClass() == TLocalevent.class
                   || t.getClass() == TLocalvariable.class
                   || t.getClass() == TModel.class
                   || t.getClass() == TMult.class
                   || t.getClass() == TNormaltermination.class
                   || t.getClass() == TOutput.class
                   || t.getClass() == TPriority.class
                   || t.getClass() == TStrongabortion.class
                   || t.getClass() == TSuspend.class
                   || t.getClass() == TSuspension.class
                   || t.getClass() == TSync.class
                   || t.getClass() == TType.class
                   || t.getClass() == TVar.class
                   || t.getClass() == TVersion.class
                   || t.getClass() == TWeakabortion.class
                   || t.getClass() == TWith.class) {
          bDoc.appendBatchString(t.getText(), KEYWORD);
        } else if (t.getClass() == TComment.class) {
          bDoc.appendBatchString(t.getText(), COMMENT);
        } else if (t.getClass() == TString.class) {
          bDoc.appendBatchString(t.getText(), STRING);
        } else if (t.getClass() == TEdge.class
                   || t.getClass() == TLBrace.class
                   || t.getClass() == TLBracket.class
                   || t.getClass() == TLReg.class
                   || t.getClass() == TParallel.class
                   || t.getClass() == TRBrace.class
                   || t.getClass() == TRBracket.class
                   || t.getClass() == TRReg.class) {
          bDoc.appendBatchString(t.getText(), STRUCTURE);
        } else if (t.getClass() == TBlank.class) {
          char[] blank = t.getText().toCharArray();
          StringBuffer buf = new StringBuffer();
          for (int i = 0; i < t.getText().length(); i++) {
            if (blank[i] == '\n') {
              if (buf.length() > 0) {
                bDoc.appendBatchString(buf.toString(), STD);
              }
              bDoc.appendBatchLineFeed(STD);
              buf = new StringBuffer();
            } else {
              buf.append(blank[i]);
            }
          }
          if (buf.length() > 0) {
            bDoc.appendBatchString(buf.toString(), STD);
          }
        } else if (t.getClass() == TNumber.class
                   || t.getClass() == TInt.class
                   || t.getClass() == TDim.class) {
          bDoc.appendBatchString(t.getText(), NUMBER);
        } else {
          bDoc.appendBatchString(t.getText(), STD);
        }
      } catch (IOException ex1) {
        ex1.printStackTrace();
      } catch (LexerException ex1) {
        ex1.printStackTrace();
      }
    } while (t.getClass() != EOF.class);
    bDoc.appendBatchLineFeed(STD);
    try {
      bDoc.processBatchUpdates(0);
    } catch (BadLocationException ex2) {
      ex2.printStackTrace();
    }
//      if (accepted(bDoc)) {
//        System.out.println("Kit file is accepted");
//      }
    pane.setDocument(bDoc);
    Browser.getModel().setKitDocument(bDoc);
    setChart = false;
    JScrollPane scroll = (JScrollPane) pane.getParent().getParent().getParent();
    ((JComponent) scroll.getParent()).revalidate();
  }

  /**
   * setTextPane.
```

```
     *
     * @param p JTextPane
     */
    public final void setTextPane(final JTextPane p) {
      pane = p;
      pane.addCaretListener(new CaretListener() {
        public void caretUpdate(CaretEvent e) {
          if (!setChart) {
            editPos = e.getDot();
          }
        }
      });
    }

    /**
     * setUpdateThread.
     *
     * @param aThread UpdateThread
     */
    public final void setUpdateThread(final UpdateThread aThread) {
      ut = aThread;
    }

    /**
     *
     * @return String
     */
    public final String getTooltip() {
      return "Statechart description language inspired by dot";
    }

    /**
     * Name of language.
     * @return String
     */
    public final String getName() {
      return "Kit";
    }

    public void restoreCursor() {
      try {
        pane.setCaretPosition(editPos);
      } catch (IllegalArgumentException ex) {
        // do not restore cursor because of bad position
        editPos = pane.getCaretPosition();
      }
    }

    public ILineNumber getLineNumber() {
      return line;
    }
  }


  /**
   *
   * <p>Title: Kiel Browser.</p>
   *
   * <p>Description: Fast way to create a Swing Document.</p>
   *
   * <p>Copyright: Copyright (c) 2005</p>
   *
   * <p>Company: Uni Kiel</p>
   *
   * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
   * @version £Revision: 1.8 £ last modified £Date: 2006/05/21 20:24:02 £
   */
  class BatchDocument extends DefaultStyledDocument {
    /**
     * EOL tag that we re-use when creating ElementSpecs.
     */
    private static final char[] EOL_ARRAY = {'\n'};

    /**
     * Batched ElementSpecs.
     */
    private ArrayList batch = null;

    /**
     * .
     */
    public BatchDocument() {
      batch = new ArrayList();
    }

    /**
     * Adds a String (assumed to not contain linefeeds) for
     * later batch insertion.
     * @param str string to append
     * @param a attributes for str
     */
    public void appendBatchString(final String str, final AttributeSet a) {
      // We could synchronize this if multiple threads
      // would be in here. Since we're trying to boost speed,
      // we'll leave it off for now.

      // Make a copy of the attributes, since we will hang onto
      // them indefinitely and the caller might change them
      // before they are processed.
//      a = a.copyAttributes();
      char[] chars = str.toCharArray();
      batch.add(new ElementSpec(a, ElementSpec.ContentType, chars, 0,
                                str.length()));
    }

    /**
     * Adds a linefeed for later batch processing.
     * @param a sets line feed with this attribute
     */
    public void appendBatchLineFeed(final AttributeSet a) {
      // See sync notes above. In the interest of speed, this
      // isn't synchronized.

      // Add a spec with the linefeed characters
      batch.add(new ElementSpec(a, ElementSpec.ContentType, EOL_ARRAY, 0, 1));

      // Then add attributes for element start/end tags. Ideally
      // we'd get the attributes for the current position, but we
      // don't know what those are yet if we have unprocessed
      // batch inserts. Alternatives would be to get the last
```

```
      // paragraph element (instead of the first), or to process
      // any batch changes when a linefeed is inserted.
      Element paragraph = getParagraphElement(0);
      AttributeSet pattr = paragraph.getAttributes();
      batch.add(new ElementSpec(null, ElementSpec.EndTagType));
      batch.add(new ElementSpec(pattr, ElementSpec.StartTagType));
    }

    /**
     *
     * @param offs int
     * @throws BadLocationException if there was an error inserting.
     */
    public void processBatchUpdates(final int offs) throws BadLocationException {
      // As with insertBatchString, this could be synchronized if
      // there was a chance multiple threads would be in here.
      ElementSpec[] inserts = new ElementSpec[batch.size()];
      batch.toArray(inserts);

      // Process all of the inserts in bulk
      super.insert(offs, inserts);
    }
  }
```

F.5. kiel.browser.view

F.5.1. BrowserCanvas

```
// £Id: BrowserCanvas.java,v 1.79 2006/05/22 19:21:52 miwi Exp £
package kiel.browser.view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Observable;

import javax.swing.JComponent;
import javax.swing.JPanel;

import kiel.dataStructure.GraphicalObject;
import kiel.graphicalInformations.CurvetoPath;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.MovetoPath;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.PathElement;
import kiel.graphicalInformations.Point;
import kiel.graphicalInformations.QuadraticCurvetoPath;
import kiel.browser.BrowserException;
import kiel.browser.model.BrowserModel;
import kiel.browser.view.rendering.Toolkit;
import kiel.browser.view.svg.SVGNode;
import kiel.browser.view.svg.SVGToolkit;
import org.csiro.svg.parser.XmlWriter;
import org.jibble.epsgraphics.EpsGraphics2D;
import org.csiro.svg.viewer.Canvas;
import javax.swing.JFrame;
import java.awt.Rectangle;
import java.awt.event.ComponentListener;
import java.awt.event.ComponentEvent;

/**
 * <p>Description: Main drawing and animation class. Creates a svg file that is
 * shown in an svg canvas.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version <p>£Revision: 1.79 £ last modified £Date: 2006/05/22 19:21:52 £</p>
 *
 */

public class BrowserCanvas extends BrowserGUI {
  /**
   * JPEG Quality.
   */
  private static final float QUALITY = (float) 0.9;

  /**
   * MoveTo PathElement Type.
   */
  private static final int MOVETO = 0;
  /**
   * LineTo PathElement Type.
   */
  private static final int LINETO = 1;
  /**
   * QuadraticCurveTo PathElement Type.
   */
  private static final int QUADRATICTO = 3;
  /**
   * CurveTo PathElement Type.
   */
  private static final int CURVETO = 4;
  /**
   * Error Type.
   */
  private static final int ERROR = -1;

  /**
   * The panel for the canvas.
   */
  private JPanel p = null;

  /**
   * The marker that does the state/transition marking.
   */
  private StateChartMarker marker;
  /**
   * Toolkit that does the drawing and animation stuff.
   */
  private Toolkit toolkit;
  /**
   * Converter from kiel ds to toolkit.
   */
  private Kiel2ToolkitConverter converter;
  /**
   * Animator from one view to another.
   */
  private View2ViewAnimator animator;

  /**
   * @param m Connects Canvas to Model
   */
  public BrowserCanvas(final BrowserModel m) {
    super(m);
    /**
     * DO not use SVGToolkit this toolkit is deprecated many interfaces
     * are not correctly implemented. From this release on use the Piccolo
     * Toolkit.
```

```
     */
    toolkit = m.getToolkit();

    marker = new StateChartMarker(m, toolkit);
    getModel().addObserver(marker);

    converter = new Kiel2ToolkitConverter(m, toolkit);
    getModel().addObserver(converter);

    animator = new View2ViewAnimator(m, toolkit);
    getModel().addObserver(animator);

    p = new JPanel(new BorderLayout());
    p.add(toolkit.getComponent());
  }



  /**
   * @return Swing Component with drawing
   */
  public final JComponent getComponent() {
    return p;
  }

  /**
   *
   * @return JComponent Canvas object from used toolkit.
   */
  public final JComponent getTkCanvas() {
    return toolkit.getCanvas();
  }

  /**
   * @param f save drawing in file f as SVG File
   */
  public final void exportToSVG(final File f) {
    Toolkit svgToolkit = new SVGToolkit();
    GraphicalObjectsLibrary.getInstance(getModel().getToolkit()).
      setNewToolkit(svgToolkit);
    Canvas canvas = (Canvas) svgToolkit.getComponent();
    JFrame frame = new JFrame();
    frame.setBounds(10,10,120,120);
    frame.getContentPane().add(canvas);
    frame.show();
    int i = 0;
//    long start = System.currentTimeMillis();
    converter.draw(svgToolkit);
    canvas.draw();
//    System.out.println("Measuring: " + (System.currentTimeMillis()-start) + "ms
     ");
    GraphicalObjectsLibrary.getInstance(getModel().getToolkit()).
      setNewToolkit(getModel().getToolkit());
    if (((SVGNode) svgToolkit.getRoot()) != null) {
      try {
        XmlWriter writer = new XmlWriter(new FileOutputStream(f));
        writer.print(((SVGNode) svgToolkit.getRoot()).getSVGElement());
      } catch (FileNotFoundException ex) {
        System.err.println("File Exception: " + ex);
      }
    }
  }

  /**
   * @param f save drawing in file f as JPEG File
   */
  public void exportToJpeg(final File f) {
  }

  /**
   * @param f save drawing as eps file f
   */
  public final void exportToEps(final File f) {
    File file = f;
    if (getModel().getStateChart() != null
        && getModel().getStateChart().getRootNode() != null) {
      EpsGraphics2D g = new EpsGraphics2D();
      g.setAccurateTextMode(false);
      getModel().getToolkit().printCanvas(g);

      FileOutputStream out = null;
      try {
        out = new FileOutputStream(file);
      } catch (FileNotFoundException ex) {
        getModel().sendExceptionMessage(ex);
        return;
      }
      try {
        out.write(g.toString().getBytes());
        out.close();
      } catch (IOException ex1) {
        getModel().sendExceptionMessage(ex1);
        return;
      }
    }
  }

  /**
   * Main method (handles all messages for class).
   * @param parm1 the sender from the message
   * @param parm2 the message (as String Object)
   */
  public final void update(final Observable parm1, final Object parm2) {
    // this class redirects the messages to the Kiel2Toolkit and
    // the View2ViewAnimation class.
  }



  /**
   * A border of white space in canvas before drawing.
   */
  public static final int BORDER = 5;



  /**
   * @param e pathelement to get type of
   * @return type of pathelement
   */
  private static int getTypeId(final PathElement e) {
    if (e instanceof MovetoPath) {
```

```
      return MOVETO;
    } else if (e instanceof LinetoPath) {
      return LINETO;
    } else if (e instanceof QuadraticCurvetoPath) {
      return QUADRATICTO;
    } else if (e instanceof CurvetoPath) {
      return CURVETO;
    }
    return ERROR;
  }

  /**
   * @param path ArrayList to test
   * @throws BrowserException ex
   */
  private static void testPath(final ArrayList path)
    throws BrowserException {
    if (path.size() <= 1) {
      throw new BrowserException(
        "Path must contain at least two elements");
    }
    if (path.get(0) != null && path.get(0).getClass() != MovetoPath.class) {
      throw new BrowserException(
        "First PathElement must be MoveToPath");
    }
    for (int i = 1; i < path.size(); i++) {
      if (path.get(i) == null) {
        throw new BrowserException(
          "" + (i + 1) + ". PathElement is null");
      }
      if (path.get(i).getClass() == MovetoPath.class) {
        throw new BrowserException(
          "Only the first element can be a MoveToPath, "
          + "for dummy elements please add LineToPath!");
      }
    }
  }

  /**
   * @param from list to be converted
   * @param to   list to be converted
   * @return a list with same pathelements
   */
  public static void correctPath(final ArrayList from,
                                 final ArrayList to)
    throws BrowserException {
    testPath(from);
    testPath(to);

    int size1 = from.size();
    int size2 = to.size();
    ArrayList maxList = null;
    ArrayList minList = null;

    if (size1 > size2) {
      maxList = from;
      minList = to;
    } else {
      maxList = to;
      minList = from;
    }
    for (int i = 0; i < maxList.size(); i++) {
      if (i < minList.size() - 1) {
//      copy and convert to highest type
//        getModel().getLogFile().log(1, "Convert to highest type " + i);
        int typ1 = getTypeId((PathElement) maxList.get(i));
        int typ2 = getTypeId((PathElement) minList.get(i));
        if (typ1 != typ2) {
          if (typ1 > typ2) {
            convertToType(maxList, minList, i, typ1, typ2);
          } else {
            convertToType(minList, maxList, i, typ2, typ1);
          }
        }
      } else if (i < maxList.size() - 1) {
        int typ3 = getTypeId((PathElement) maxList.get(i));
//        getModel().getLogFile().log(1, "Filling with dummy values " + i
//        + " Typ is " + typ3);
        if (minList.size() > 1) {
          minList.add(i, createDummy(typ3, (PathElement) minList.get(i - 1)));
        } else {
          minList.add(i, createDummy(typ3, new MovetoPath(new Point(0, 0))));
        }
      } else {
        if (maxList.size() == minList.size()) {
//          getModel().getLogFile().log(1, "Convert Last elements " + i
//            + "Sizes Max" + maxList.size() + " Min" + minList.size());
          int typ1 = getTypeId((PathElement) maxList.get(maxList.size() - 1));
          int typ2 = getTypeId((PathElement) minList.get(minList.size() - 1));
          if (typ1 != typ2) {
            if (typ1 > typ2) {
              convertToType(maxList, minList, maxList.size() - 1, typ1, typ2);
            } else {
              convertToType(minList, maxList, minList.size() - 1, typ2, typ1);
            }
          }
        } else {
          int typ3 = getTypeId((PathElement) maxList.get(i));
//          getModel().getLogFile().log(1,
//          "Filling with dummy values " + i + " Typ is " + typ3);
          if (minList.size() > 1) {
            minList.add(i, createDummy(typ3, (PathElement) minList.get(i - 1)));
          } else {
            minList.add(i, createDummy(typ3, new MovetoPath(new Point(0, 0))));
          }
        }
      }
    }
  }

  /**
   * Makes type of from[i] and to[i] the same (converting fromTyp to toType).
   * @param from list with pathelements
   * @param to list withpathelemts
   * @param i number to be converted
   * @param fromTyp from type
   * @param toType to type
   */
  private static void convertToType(final ArrayList from, final ArrayList to,
                                    final int i, final int fromTyp,
```

```
                                    final int toType) {
    PathElement path = null;
    switch (fromTyp) {
    case LINETO:
      switch (toType) {
      case CURVETO:
        path = new CurvetoPath(((PathElement) from.get(i)).getTargetPoint(),
                               ((PathElement) from.get(i - 1)).
                               getTargetPoint(),
                               ((PathElement) from.get(i)).getTargetPoint());
        break;
      case QUADRATICTO:
        path = new QuadraticCurvetoPath(((PathElement) from.get(i - 1)).
                                        getTargetPoint(),
                                        ((PathElement) from.get(i)).
                                        getTargetPoint());
        break;
      default:
        //
      }
      from.add(i, path);
      from.remove(i + 1);
      break;
    case CURVETO:
      switch (toType) {
      case LINETO:
        path = new CurvetoPath(((PathElement) to.get(i)).getTargetPoint(),
                               ((PathElement) to.get(i - 1)).getTargetPoint(),
                               ((PathElement) to.get(i)).getTargetPoint());
        to.add(i, path);
        to.remove(i + 1);
        break;
      case QUADRATICTO:
        path = new CurvetoPath(((PathElement) to.get(i)).getControlPoint1(),
                               ((PathElement) to.get(i)).getControlPoint1(),
                               ((PathElement) to.get(i)).getTargetPoint());
        to.add(i, path);
        to.remove(i + 1);
        break;
      default:
        //
      }
      break;
    case QUADRATICTO:
      switch (toType) {
      case LINETO:
        path = new QuadraticCurvetoPath(((PathElement) to.get(i - 1)).
                                        getTargetPoint(),
                                        ((PathElement) to.get(i)).
                                        getTargetPoint());
        to.add(i, path);
        to.remove(i + 1);
        break;
      case CURVETO:
        path =
          new CurvetoPath(((PathElement) from.get(i)).getControlPoint1(),
                          ((PathElement) from.get(i)).getControlPoint1(),
                          ((PathElement) from.get(i)).getTargetPoint());
        from.add(i, path);
        from.remove(i + 1);
        break;
      default:
        //
      }
      break;
    default:
      //
    }
  }

  /**
   * creates types of path elements without "moving".
   * @param typ type of path elements to be created
   * @param pe point that be is used for creation
   * @return a not "moving" point with type typ
   */
  private static PathElement createDummy(final int typ, final PathElement pe) {
    PathElement path = null;
    switch (typ) {
    case MOVETO:
      path = new MovetoPath(pe.getTargetPoint().getX(),
                            pe.getTargetPoint().getY());
      break;
    case LINETO:
      path = new LinetoPath(pe.getTargetPoint().getX(),
                            pe.getTargetPoint().getY());
      break;
    case QUADRATICTO:
      path = new QuadraticCurvetoPath(pe.getTargetPoint().getX(),
                                      pe.getTargetPoint().getY(),
                                      pe.getTargetPoint().getX(),
                                      pe.getTargetPoint().getY());
      break;
    case CURVETO:
      path = new CurvetoPath(pe.getTargetPoint().getX(),
                             pe.getTargetPoint().getY(),
                             pe.getTargetPoint().getX(),
                             pe.getTargetPoint().getY(),
                             pe.getTargetPoint().getX(),
                             pe.getTargetPoint().getY());
      break;
    default:
      //
    }
    return path;
  }

  /**
   * @param col Color to convert
   * @return String as hex
   */
  public static String encodeColorForCSS(final Color col) {
    String r;
    String g;
    String b;
    final int hexbase = 16;
    if (col.getRed() < hexbase) {
      r = "0" + Integer.toHexString(col.getRed());
    } else {
      r = Integer.toHexString(col.getRed());
```

```
    }
    if (col.getGreen() < hexbase) {
      g = "0" + Integer.toHexString(col.getGreen());
    } else {
      g = Integer.toHexString(col.getGreen());
    }
    if (col.getBlue() < hexbase) {
      b = "0" + Integer.toHexString(col.getBlue());
    } else {
      b = Integer.toHexString(col.getBlue());
    }
    return "#" + r + g + b;
  }
  /**
   * Id of group (whole drawin node or transition object).
   * @param go GraphicalObject
   * @return String
   */
  public static String getGroupId(GraphicalObject go) {
    return "Group#" + go.getID();
  }
}
```

F.5.2. BrowserGUI

```
// £Id: BrowserGUI.java,v 1.6 2005/05/12 15:12:49 miwi Exp £
package kiel.browser.view;

import java.util.Observer;

import javax.swing.JComponent;

import kiel.browser.model.BrowserModel;

/**
 * <p>Description: Abstract Class for all GUI components.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.6 £ last modified £Date: 2005/05/12 15:12:49 £
 *
 */
public abstract class BrowserGUI
    implements Observer {
  /**
   * Data model for all GUI classes.
   */
  private BrowserModel model;

  /**
   * @param m connect this GUI class with model m
   */
  public BrowserGUI(final BrowserModel m) {
    model = m;
    model.addObserver(this);
  }


  /**
   * @return Swing component for this class
   */
  public abstract JComponent getComponent();

  /**
   * @return the connected model
   */
  public final BrowserModel getModel() {
    return model;
  }

}
```

F.5.3. BrowserMenuBar

```
// £Id: BrowserMenuBar.java,v 1.28 2007/02/08 13:55:19 khe Exp £
package kiel.browser.view;

import java.util.Observable;

import javax.swing.ButtonGroup;
import javax.swing.JComponent;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

import kiel.dataStructure.Node;
import kiel.browser.controller.MenuController;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelMessage;
import kiel.preferences.LogFile;

/**
 * <p>Description: Gui stuff for browser menus.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.28 £ last modified £Date: 2007/02/08 13:55:19 £
 *
 * £Log: BrowserMenuBar.java,v £
 * Revision 1.28  2007/02/08 13:55:19  khe
 * * changed imports from kiel.base to kiel.preferences
 * * fixed classpaths wrt kiel.base
 * * fixed build dependencies wrt kiel.base
 *
 * Revision 1.27  2007/02/06 18:23:21  khe
 * * new xml based preferences system
 * * new preferences dialog
 * * moved preferences and logfile to new module base to resolve circular build time
 *   dependency between util.jar and graphicalInformations.jar
 *
 * Revision 1.26  2006/01/16 18:00:16  miwi
 * Optimzed imports
 *
 * Revision 1.25  2005/11/28 10:41:26  miwi
 * Browser Version 2.0 RC1 import
 *
 * Revision 1.24  2005/07/07 10:09:38  miwi
 * *** empty log message ***
 *
 * Revision 1.23  2005/06/30 09:18:02  miwi
 * *** empty log message ***
 *
 * Revision 1.22  2005/06/02 09:37:42  miwi
 * bug 83 removed<br>
 *
 * Revision 1.21  2005/05/27 10:52:20  miwi
 * On demand added<br>
 *
 * Revision 1.20  2005/05/24 17:17:49  miwi
 * New SVG Export added<br>
 *
 * Revision 1.19  2005/05/11 09:44:05  miwi
 * removed reference to getName()
 *
 * Revision 1.18  2005/04/11 17:05:58  miwi
 * Removed deprecated method calls to layouter<br>
 *
 * Revision 1.17  2005/04/05 17:34:46  miwi
 * getName() may be null <br>
 *
 * Revision 1.16  2005/04/04 08:50:11  miwi
 * copy defaults/better exception/better log<br>
 *
 * Revision 1.15  2005/03/31 09:27:05  miwi
 * *** empty log message ***
 *
 * Revision 1.14  2005/03/16 09:12:13  miwi
 * Style checked<br>
 *
 * Revision 1.11  2005/03/10 11:03:55  miwi
 * Eps output works again
 *
 * Revision 1.10  2005/03/01 08:47:14  tkl
 * changed: kiel.layouter.Handler.getHandeler() -> instance()
 *
 * Revision 1.9  2005/02/26 07:06:33  miwi
 * New feature marking in trees<br>
 *
 * Revision 1.8  2005/02/23 17:15:07  miwi
 * Uninitialised signal support<br>
 *
 * Revision 1.7  2005/02/22 09:56:28  miwi
 * removed bug in layouter call<br>
 *
 * Revision 1.1  2005/01/18 16:31:58  miwi
 * New package model<br>
 *
 * Revision 1.10  2004/12/06 19:14:38  miwi
 * New Microstep list handling<br>
 *
 * Revision 1.9  2004/11/30 20:40:58  miwi
 * group in menu timing in estudio ini<br>
 *
 * Revision 1.8  2004/11/18 19:26:24  miwi
 * First simulator control<br>
 *
 * Revision 1.7  2004/11/16 11:41:26  miwi
 * Configuration handling enabled again<br>
 *
 * Revision 1.6  2004/08/24 06:25:50  miwi
 * Prepared for eps output<br>
 *
 * Revision 1.2  2004/07/04 18:56:06  miwi
 * new node naming in menubar<br>
```

```
 *
 * Revision 1.1  2004/07/04 12:02:45  miwi
 * Creating/Integrating new view: BrowserMenubar <br>
 *
 */

public class BrowserMenuBar
    extends BrowserGUI {

  /**
   * Action command for exporting jpg.
   */
  public static final String EXPORTJPG = "jpg";

  /**
   * Action command for exporting eps.
   */
  public static final String EXPORTEPS = "eps";

  /**
   * Action command for exporting svg.
   */
  public static final String EXPORTSVG = "svg";

  /**
   * Action command for switching configurations.
   */
  public static final String SWITCH_CONF = "conf";

  /**
   *  holds the whole menubar.
   */
  private JMenuBar jMenuBar1 = new JMenuBar();
  /**
   * Layouter menu.
   */
  private JMenu jMenu1 = new JMenu();
  /**
   * Configurations menu.
   */
  private JMenu jMenu3 = new JMenu();
  /**
   *  "No configurations yet" item.
   */
  private JMenuItem jMenuItem1 = new JMenuItem();
  /**
   *  "Export as jpg" item.
   */
  private JMenuItem jMenuItem2 = new JMenuItem();
  /**
   *  "Export as eps" item.
   */
  private JMenuItem jMenuItem3 = new JMenuItem();
  /**
   *  "Export as svg" item.
   */
  private JMenuItem jMenuItem4 = new JMenuItem();
  /**
   *  Model log file.
   */
  private LogFile l = getModel().getBrowserLogFile();

  /**
   * Controller for this class.
   */
  private MenuController contr;

  /**
   * @param m connects this view to the model m
   */
  public BrowserMenuBar(final BrowserModel m) {
    super(m);
    contr = new MenuController(m);
    jbInit();
  }

  /**
   *  inits the swing components.
   */
  private void jbInit() {
    jMenu3.setText("Configurations");
    jMenu3.setMnemonic('C');

    jMenuItem1.setText("No Configurations yet");
    jMenuItem1.setEnabled(false);
    jMenu3.add(jMenuItem1);

    jMenu1.setText("Layouter");
    jMenu1.setMnemonic('L');
    ButtonGroup group = new ButtonGroup();

    jMenuItem2.setText("Export as jpg");
    jMenuItem2.addActionListener(contr);
    jMenuItem2.setActionCommand(EXPORTJPG);

    jMenuItem3.setText("Export as eps");
    jMenuItem3.addActionListener(contr);
    jMenuItem3.setActionCommand(EXPORTEPS);

    jMenuItem4.setText("Export as svg");
    jMenuItem4.addActionListener(contr);
    jMenuItem4.setActionCommand(EXPORTSVG);

    /*    Iterator iter = Handler.instance().getLayouterNames().iterator();
        while (iter.hasNext()) {
          String name = (String) iter.next();
          JCheckBoxMenuItem mi = new JCheckBoxMenuItem(name);
          mi.addActionListener(this);
          mi.setActionCommand(name);
          group.add(mi);
          jMenu1.add(mi);
          layouters.add(mi);
        }
     */
  }

  /**
   * @return whole menu
   */
  public final JComponent getComponent() {
```

```
    return jMenuBar1;
  }

  /**
   * @param nodes composes all names from nodes
   * @return a composed string from all nodes
   */
  private String composeNodeText(final Node[] nodes) {
    StringBuffer result = new StringBuffer();
    for (int i = 0; i < nodes.length; i++) {
      kiel.dataStructure.Node node = nodes[i];
      if (node instanceof kiel.dataStructure.PseudoState) {
        result.append("Pseudostate(" + node.getParent().toString() + ")");
      } else {
        result.append(node.toString());
      }
      if ((i + 1) < nodes.length) {
        result.append("␣");
      }
    }

    return result.toString();
  }

  /**
   * @return Menu with all configurations
   */
  public final JMenu getConfigurationsMenu() {
    return jMenu3;
  }

  /**
   * @return Menu with all layouters
   */
  public final JMenu getLayoutersMenu() {
    return jMenu1;
  }

  /**
   * @return Export as jpg Item
   */
  public final JMenuItem getExportJpgMenuItem() {
    return jMenuItem2;
  }

  /**
   * @return Export as eps Item
   */
  public final JMenuItem getExportEpsMenuItem() {
    return jMenuItem3;
  }

  /**
   * @return Export as svg Item
   */
  public final JMenuItem getExportSvgMenuItem() {
    return jMenuItem4;
  }

  /**
   * @param parm1 Observable that sends the message
   * @param parm2 the message (all messages are Strings)
   */
  public final void update(final Observable parm1, final Object parm2) {
    if (parm2.getClass() == ModelMessage.class) {
      ModelMessage message = (ModelMessage) parm2;
      if (message.equals(BrowserModel.ALL_CONFIGS)) {
        kiel.configMngr.Configuration conf = null;
        jMenu3.removeAll();
        for (int i = 0; i < getModel().getAllConfigurations().length; i++) {
          conf = getModel().getAllConfigurations()[i];
          JMenuItem jMenuConf = new JMenuItem();
          jMenuConf.setText(composeNodeText(conf.getNodes()));
          jMenuConf.addActionListener(contr);
          jMenuConf.setActionCommand(SWITCH_CONF + conf.toString());
          jMenu3.add(jMenuConf);
        }
      }
      /*else if (message.equals(getModel().NEW_LAYOUTER)) {
        Layouter layouter = (Layouter) message.getParameter();
        for (int i = 0; i < layouters.size(); i++) {
          if (Handler.instance().getLayouterNamed(
              ((JCheckBoxMenuItem)
                layouters.get(i)).getActionCommand()).equals(layouter)) {
            ((JCheckBoxMenuItem) layouters.get(i)).setSelected(true);
          }
        }
            }
       */
    }
  }

}
```

F.5.4. BrowserToolbar

```
// £Id: BrowserToolbar.java,v 1.28 2007/02/08 13:55:19 khe Exp £
package kiel.browser.view;

import java.awt.event.ActionEvent;
import java.util.Observable;

import javax.swing.AbstractAction;
import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JTextField;
import javax.swing.JToolBar;

import kiel.configMngr.Configuration;
import kiel.browser.BrowserProperties;
import kiel.browser.controller.MacroStepAction;
import kiel.browser.controller.MicroStepAction;
import kiel.browser.controller.TracePlayback;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.simulationTrace.Reset;
import kiel.simulationTrace.TraceStep;
import kiel.preferences.LogFile;
import java.util.Iterator;

/**
 * <p>Description: Swing Toolbar for simulator control.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>

 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.28 £ last modified £Date: 2007/02/08 13:55:19 £
 */
public class BrowserToolbar extends BrowserGUI {
  /**
   * action command for reset.
   */
  public static final String RESET = "reset";
  /**
   * action command for microstep.
   */
  public static final String MICROSTEP = "micro";
  /**
   * action command for macrostep.
   */
  public static final String MACROSTEP = "macro";
  /**
   * action command for macrostep continue.
   */
  public static final String CONTINUE = "cont";
  /**
   * action command for trace rewind.
   */
  public static final String TRACEREWIND = "rewind";
  /**
   * action command for trace forward.
   */
  public static final String TRACEFORWARD = "forward";
  /**
   * action command for trace playback.
   */
  public static final String TRACEPLAY = "play";
  /**
   * action command for trace stop.
   */
  public static final String TRACESTOP = "stop";
  /**
   * action command for trace step.
   */
  public static final String TRACESTEP = "step";
  /**
   * action command for trace step back.
   */
  public static final String TRACEBACK = "step_back";
  /**
   * the whole toolbar.
   */
  private JToolBar toolbar;
  /**
   * button for macro steps.
   */
  private JButton macroStep;
  /**
   * button for micro steps.
   */
  private JButton microStep;
  /**
   * button for reseting.
   */
  private JButton reset;
  /**
   * button for trace rewind.
   */
  private JButton rewind;
  /**
   * button for trace step back.
   */
  private JButton stepBack;
  /**
   * button for trace do step.
   */
  private JButton step;
  /**
   * button for trace forward.
   */
  private JButton forward;
  /**
   * button for trace playback.
   */
  private JButton play;
  /**
   * button for trace stop playback.
```

```
 */
private JButton stop;
/**
 * text component to display playback session step.
 */
private JTextField playbackSessionStep;
/**
 * icon for macro step button.
 */
private Icon macro = ResourceLoader.createImageIcon(
  "kiel/browser/images/run.gif");
/**
 * icon for micro step button.
 */
private Icon micro = ResourceLoader.createImageIcon(
  "kiel/browser/images/walk.gif");
/**
 * icon for reset button.
 */
private Icon resetIcon = ResourceLoader.createImageIcon(
  "kiel/browser/images/refresh16.gif");
/**
 * Action Class for macro step action.
 */
private MacroStepAction macroStepper = null;
/**
 * Action Class for micro step action.
 */
private MicroStepAction microStepper = null;

/**
 * Constructor inits the swing methods.
 * @param m connects this view to the model m
 */
public BrowserToolbar(final BrowserModel m) {
  super(m);
  toolbar = new JToolBar();
  macroStep = new JButton("MacroStep");
  macroStep.setIcon(macro);
  macroStep.setOpaque(false);
  macroStep.setEnabled(false);
  macroStep.setMnemonic(BrowserProperties.getMacroStepMnemonic());
  macroStepper = new MacroStepAction(m);
  macroStep.addActionListener(macroStepper);
  macroStep.addActionListener(new AbstractAction() {
    public void actionPerformed(final ActionEvent e) {
      microStep.setEnabled(false);
      macroStep.setEnabled(false);
      reset.setEnabled(false);
      //        microStep.repaint();
      //        macroStep.repaint();
      //        reset.repaint();
    }
  });

  microStep = new JButton("MicroStep");
  microStep.setIcon(micro);
  microStep.setOpaque(false);
  microStep.setActionCommand(BrowserToolbar.MICROSTEP);
  microStep.setEnabled(false);
  microStepper = new MicroStepAction(m);
  microStep.addActionListener(microStepper);
  microStep.addActionListener(new AbstractAction() {
    public void actionPerformed(final ActionEvent e) {
      //        microStep.setEnabled(false);
      macroStep.setText("continue...");
      macroStep.setToolTipText("continue to next next macrostep");
      macroStep.setActionCommand(CONTINUE);
      //        microStep.repaint();
      //        macroStep.repaint();
    }
  });
  microStep.setMnemonic(BrowserProperties.getMicroStepMnemonic());

  reset = new JButton(new AbstractAction("Reset") {
    public void actionPerformed(final ActionEvent e) {
      if (getModel().getTrace() != null) {
        getModel().getTrace().record();
        TraceStep step = new TraceStep();
        step.addReset();
        getModel().getTrace().addStep(step);
      }
      BrowserProperties.reload();
      getModel().resetSimulator(false);
      getModel().setMacroStepSimulation(true);
      getModel().showStaticView();
      getModel().setMode(BrowserModel.EDIT);
      getModel().setStatus("Resetting done.");
    }
  });
  reset.setIcon(resetIcon);
  reset.setActionCommand(BrowserToolbar.RESET);
  reset.setOpaque(false);
  reset.setMnemonic(BrowserProperties.getResetMnemonic());
  final int height = 12;
  playbackSessionStep = new JTextField("trace step: 0/0", height);
  playbackSessionStep.setEditable(false);

  step = new JButton("step forward");
  step.setActionCommand(BrowserToolbar.TRACESTEP);
  step.setOpaque(false);
  step.setMnemonic(BrowserProperties.getTraceStepForwardMnemonic());
  step.addActionListener(new AbstractAction() {
    public void actionPerformed(final ActionEvent e) {
      if (getModel().getTrace().hasNextStep()) {
        TraceStep step = getModel().getTrace().getNextStep();
        if (step.containsReset()) {
          getModel().resetSimulator(false);
          if (step.getStepDataAsCollection().size() > 1) {
            getModel().getBrowserLogFile().log(LogFile.ERROR,
                                               "Reset tracestep contains data"
);
            Iterator iter = step.getStepDataAsCollection().iterator();
            while (iter.hasNext()) {
              getModel().getBrowserLogFile().log(LogFile.ERROR,
                "" + iter.next());
            }
          }
          getModel().showStaticView();
        } else {
```

```
        getModel().setInputTables(step);
        macroStepper.doMacroStep(step.getStepDataAsCollection());
      }
    }
  }
});

stepBack = new JButton("step backward");
stepBack.setActionCommand(BrowserToolbar.TRACEBACK);
stepBack.setOpaque(false);
stepBack.setMnemonic(BrowserProperties.getTraceStepBackMnemonic());
stepBack.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    kiel.simulationTrace.TraceStep prevStep =
      getModel().getTrace().getPreviousStep();
    getModel().getTrace().play();
    getModel().getTrace().startTrace();
    getModel().resetSimulator(false);

    if (prevStep != null) {
      Configuration conf = null;
      while (getModel().getTrace().getNextStep() != prevStep) {
        TraceStep step = getModel().getTrace().getCurrentStep();
        if (step.containsReset()) {
          getModel().resetSimulator(false);
          if (step.getStepDataAsCollection().size() > 1) {
            getModel().getBrowserLogFile().log(LogFile.ERROR,
              "Reset tracestep contains data");
            Iterator iter = step.getStepDataAsCollection().iterator();
            while (iter.hasNext()) {
              getModel().getBrowserLogFile().log(LogFile.ERROR,
                "" + iter.next());
            }
          }
        } else {
          getModel().setInputTables(step);
          if (ModelHelper.createMacroStep(step.getStepDataAsCollection())
              != null) {
            Configuration confStep =
              ModelHelper.getConfigurationFromMacroStep();
            if (confStep != null) {
              conf = confStep;
            }
          }
        }
      }
      getModel().setConfiguration(conf);
      macroStepper.doMacroStep(getModel().getTrace().getCurrentStep()
                               .getStepDataAsCollection());
      if (getModel().getTrace().getCurrentStep() == null) {
        getModel().getTrace().seek( -1);
      }
    }
  }
});

rewind = new JButton("rewind");
rewind.setActionCommand(BrowserToolbar.TRACEREWIND);
rewind.setOpaque(false);
rewind.setMnemonic(BrowserProperties.getTraceRewindMnemonic());
rewind.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    getModel().getTrace().play();
    getModel().getTrace().startTrace();
    getModel().resetSimulator(false);
    getModel().showStaticView();
  }
});

TracePlayback playBack = new TracePlayback(getModel());

stop = new JButton("stop");
stop.setActionCommand(BrowserToolbar.TRACESTOP);
stop.setOpaque(false);
stop.setMnemonic(BrowserProperties.getTraceStopMnemonic());
stop.addActionListener(playBack);
stop.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    microStep.setEnabled(true);
    macroStep.setEnabled(true);
    reset.setEnabled(true);
    step.setEnabled(true);
    stepBack.setEnabled(true);
    rewind.setEnabled(true);
    play.setEnabled(true);
    forward.setEnabled(true);
  }
});

play = new JButton("play");
play.setActionCommand(BrowserToolbar.TRACEPLAY);
play.setOpaque(false);
play.setMnemonic(BrowserProperties.getTracePlayMnemonic());
play.addActionListener(playBack);
play.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    microStep.setEnabled(false);
    macroStep.setEnabled(false);
    reset.setEnabled(false);
    step.setEnabled(false);
    stepBack.setEnabled(false);
    rewind.setEnabled(false);
    play.setEnabled(false);
    forward.setEnabled(false);
  }
});

forward = new JButton("fast-forward");
forward.setActionCommand(BrowserToolbar.TRACEFORWARD);
forward.setOpaque(false);
forward.setMnemonic(BrowserProperties.getTraceFastForwardMnemonic());
forward.addActionListener(new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    Configuration conf = null;
    while (getModel().getTrace().hasNextStep()) {
      TraceStep step = getModel().getTrace().getNextStep();
      if (step.containsReset()) {
        getModel().resetSimulator(false);
        if (step.getStepDataAsCollection().size() > 1) {
          getModel().getBrowserLogFile().log(LogFile.ERROR,
```

```
                "Reset trace step contains data");
            Iterator iter = step.getStepDataAsCollection().iterator();
            while (iter.hasNext()) {
                getModel().getBrowserLogFile().log(LogFile.ERROR,
                    "" + iter.next());
            }
        }
    } else {
        getModel().setInputTables(step);
        if (ModelHelper.createMacroStep(step.getStepDataAsCollection())
                != null) {
            Configuration confStep =
                ModelHelper.getConfigurationFromMacroStep();
            if (confStep != null) {
                conf = confStep;
            }
        }
    }
    getModel().setConfiguration(conf);
    macroStepper.doMacroStep(getModel().getTrace().getCurrentStep()
                    .getStepDataAsCollection());
}
});

toolbar.add(macroStep);
toolbar.add(microStep);
toolbar.add(reset);
toolbar.addSeparator();
toolbar.add(playbackSessionStep);
toolbar.add(rewind);
toolbar.add(stop);
toolbar.add(stepBack);
toolbar.add(step);
toolbar.add(play);
toolbar.add(forward);
}

/**
 * @return the toolbar component
 */
public final JComponent getComponent() {
  return toolbar;
}

/**
 * @param parm1 Observerable that send the message
 * @param parm2 the message as String
 */
public final void update(final Observable parm1, final Object parm2) {
  if (parm2.equals(BrowserModel.REPAINT_TRACEINFO)
      || parm2.equals(BrowserModel.RESET)
      || parm2.equals(BrowserModel.SIMSTEP_COMPLETED)) {
    playbackSessionStep.setText("trace step: "
                + getModel().getTrace().getPositionInfo());
    playbackSessionStep.repaint();
  } else if (parm2.equals(BrowserModel.SIM_MODE_MACRO)
      && getModel().getSimulator() != null) {
    macroStep.setText("MacroStep");
    macroStep.setActionCommand(MACROSTEP);
    macroStep.setToolTipText(null);
  } else if (parm2.equals(BrowserModel.SIMSTEP_STOP)
      && getModel().getSimulator() != null) {
    macroStep.setEnabled(true);
    microStep.setEnabled(true);
    reset.setEnabled(true);
    step.setEnabled(true);
    stepBack.setEnabled(true);
    rewind.setEnabled(true);
    play.setEnabled(true);
    forward.setEnabled(true);
    toolbar.repaint();
  } else if (parm2.equals(BrowserModel.NOSIM)) {
    macroStep.setEnabled(false);
    microStep.setEnabled(false);
  } else if (parm2.equals(BrowserModel.UPDATE_CHART)) {
    macroStep.setEnabled(true);
    microStep.setEnabled(true);
    playbackSessionStep.setText("trace step: "
                + getModel().getTrace().getPositionInfo());
    playbackSessionStep.repaint();
  } else if (parm2.equals(BrowserModel.NEW_CHART)) {
    macroStep.setEnabled(true);
    microStep.setEnabled(true);
    playbackSessionStep.setText("trace step: "
                + getModel().getTrace().getPositionInfo());
    playbackSessionStep.repaint();
  }
}
}
```

F.5.5. BrowserTree

```
// £Id: BrowserTree.java,v 1.30 2007/02/08 13:55:20 khe Exp £
package kiel.browser.view;

import java.awt.Color;
import java.awt.Component;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Observable;

import javax.swing.Icon;
import javax.swing.JComponent;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.ToolTipManager;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.PseudoState;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.browser.BrowserException;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.preferences.LogFile;

/**
 * <p>Description: This class contains the gui stuff for showing a statechart
 * in the browser in a tree view.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.30 £ last modified £Date: 2007/02/08 13:55:20 £
 *
 */
public class BrowserTree
    extends BrowserGUI {
  /**
   * Our browser logfile.
   */
  private LogFile l;

  /**
   * in this scrollpane is our tree.
   */
  private JScrollPane scrollpane;

  /**
   *  this is the statechart tree.
   */
  private JTree tree;

  /**
   * top tree element.
   */
  private DefaultMutableTreeNode top;

  /**
   * tree model.
   */
  private DefaultTreeModel treeModel;

  /**
   * @param m the Model to connect to
   */
  public BrowserTree(final BrowserModel m) {
    super(m);
    l = m.getBrowserLogFile();
    top = new DefaultMutableTreeNode("No Chart");
    treeModel = new DefaultTreeModel(top);

    tree = new JTree(treeModel);
    tree.addMouseListener(new MouseAdapter() {
      public void mousePressed(final MouseEvent e) {
        // @todo ispopuptrigger
        int selRow = tree.getRowForLocation(e.getX(), e.getY());
        TreePath selPath = tree.getPathForLocation(e.getX(), e.getY());
        if (selRow != -1) {
          if (e.getClickCount() == 1) {
            mySingleClick(selRow, selPath);
          } else if (e.getClickCount() == 2) {
            myDoubleClick(selRow, selPath);
          }
        }
      }
    });

    tree.setCellRenderer(new MyRenderer(getModel()));
    tree.putClientProperty("JTree.lineStyle", "Angled");
    ToolTipManager.sharedInstance().registerComponent(tree);

    scrollpane = new JScrollPane(tree);
  }
```

```
  /**
   * what happens on a single click.
   * @param row row that was single clicked
   * @param path path to element that was single clicked
   */
  public final void mySingleClick(final int row, final TreePath path) {
//    l.log(LogFile.DEBUG, "Tree was one clicked: " + row
//          + " " + path.toString());
  }

  /**
   * what happens on a double click.
   * @param row row that was double clicked
   * @param path path to element that was double clicked
   */
  public final void myDoubleClick(final int row, final TreePath path) {
//    l.log(LogFile.DEBUG, "Tree was double clicked: " + row
//          + " " + path.toString());
  }

  /**
   * @return a JComponent that represents the gui
   */
  public final JComponent getComponent() {
    return scrollpane;
  }

  /**
   * @param parm1 the object that send the message
   * @param parm2 the message itself (as String)
   */
  public final void update(final Observable parm1, final Object parm2) {
    if (parm2.equals(BrowserModel.TABLE_UPDATE)
      || parm2.equals(BrowserModel.UPDATE_CHART)) {
      try {
        top = createNodes(getModel().getStateChart().getRootNode(), null);
      } catch (BrowserException ex) {
        getModel().sendExceptionMessage(ex);
        return;
      }
      treeModel = new DefaultTreeModel(top);
      tree.setModel(treeModel);
      tree.repaint();
    }
  }

  /**
   * @param st CompositeState
   * @return DefaultMutableTreeNode
   */
  private DefaultMutableTreeNode createSigsVars(final CompositeState st) {
    DefaultMutableTreeNode subNode = new DefaultMutableTreeNode(
      new String("Declarations"));
    boolean found = false;
    if (st.getVariables().size() > 0) {
      found = true;
      for (int i = 0; i < st.getVariables().size(); i++) {
        subNode.add(new DefaultMutableTreeNode(st.getVariables().get(i)));
      }
    }
    if (st.getLocalEvents().size() > 0) {
      found = true;
      for (int i = 0; i < st.getLocalEvents().size(); i++) {
        subNode.add(new DefaultMutableTreeNode(st.getLocalEvents().get(i)));
      }
    }
    if (found) {
      return subNode;
    }
    return null;
  }

  /**
   * @param root      Root node for this tree
   * @param parent create tree with top as root node
   * @return recursive created tree
   * @throws BrowserException if there is something wrong with the datastructure
   */
  private DefaultMutableTreeNode createNodes(final Node root,
                                             final DefaultMutableTreeNode parent
                                             ) throws BrowserException {

    DefaultMutableTreeNode subNode = new DefaultMutableTreeNode(root);
    if ((root instanceof ORState)
        || (root instanceof Region)
        || (root instanceof ANDState)) {
      ArrayList subs = null;
      if (root instanceof ANDState) {
        subs = (ArrayList) ((ANDState) root).getSubnodes();
        DefaultMutableTreeNode decl = createSigsVars((ANDState) root);
        if (decl != null) {
          subNode.add(decl);
        }
      } else if (root instanceof Region) {
        subs = (ArrayList) ((Region) root).getSubnodes();
      } else {
        subs = (ArrayList) ((ORState) root).getSubnodes();
        DefaultMutableTreeNode decl = createSigsVars((ORState) root);
        if (decl != null) {
          subNode.add(decl);
        }
      }
      // just to be sure
      if (subs != null) {
        kiel.dataStructure.Node dummy;
        for (int i = 0; i < subs.size(); i++) {
          dummy = (kiel.dataStructure.Node) subs.get(i);
          if (dummy.getParent() != root) {
            l.log(LogFile.DEBUG, "Wrong datastructure parent "
                  + "is incorrect: child: "
                  + dummy
                  + "; should have parent : " + root
                  + " but has actual parent: "
                  + dummy.getParent());
            throw new BrowserException("Wrong datastructure parent "
                  + "is incorrect: child: "
                  + dummy
                  + "; should have parent : " + root
                  + " but has actual parent: "
                  + dummy.getParent());
```

```
            }
            DefaultMutableTreeNode child = createNodes(dummy, subNode);
            if (child != null) {
              subNode.add(child);
            }
          }
        }
      }
      return subNode;
    }

  }

  /**
   *
   * <p>Description: Tree rendering class.</p>
   * <p>Copyright: Copyright (c) 2004</p>
   * <p>Company: Uni Kiel</p>
   *
   * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
   * @version £Revision: 1.30 £ last modified £Date: 2007/02/08 13:55:20 £
   *
   */
  class MyRenderer
      extends DefaultTreeCellRenderer {

    /**
     * @param n the node
     * @return the displayed text for this node
     */
    private String createNode(final Node n) {
      StringBuffer start = new StringBuffer();
      if (n != null) {
        if (n.getName() != null && !n.getName().equals("")) {
          start.append(n.getName());
          start.append("␣");
        }
        if (n.getClass() == ANDState.class) {
          start.append("(ANDState)");
        } else if (n.getClass() == ORState.class) {
          start.append("(ORState)");
        } else if (n.getClass() == Region.class) {
          start.append("(Region)");
        } else if (n.getClass() == SimpleState.class) {
        } else if (n.getClass() == InitialState.class) {
          start.append("(Initial)");
        } else if (n.getClass() == Choice.class) {
          start.append("(Choice)");
        } else if (n.getClass() == DynamicChoice.class) {
          start.append("(DynamicChoice)");
        } else if (n.getClass() == History.class) {
          start.append("(History)");
        } else if (n.getClass() == DeepHistory.class) {
          start.append("(DeepHistory)");
        } else if ((n.getClass() == FinalState.class)
                   || (n.getClass() == FinalSimpleState.class)
                   || (n.getClass() == FinalANDState.class)
                   || (n.getClass() == FinalORState.class)) {
          start.append("(FinalState)");
        } else if (n instanceof Suspend) {
          start.append("(Suspend)");
        } else if (n instanceof PseudoState) {
          start.append("(PseudoState)");
        } else {
          start.append(n.getClass().toString());
        }
      }
      return start.toString();
    }

    /**
     * sample andicon.
     */
    private Icon andIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/and.gif");
    /**
     * Icon for orstate.
     */
    private Icon orIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/or.gif");
    /**
     * Icon for simple state.
     */
    private Icon simpleIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/simple.gif");
    /**
     * Icon for inital state.
     */
    private Icon initialIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/init.gif");
    /**
     * Icon for choice state.
     */
    private Icon choiceIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/choice.gif");
    /**
     * Icon for region state.
     */
    private Icon regionIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/region.gif");
    /**
     * Icon for suspend state.
     */
    private Icon suspendIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/suspend.gif");
    /**
     * Icon for final state.
     */
    private Icon finalIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/final.gif");
    /**
     * Icon for history state.
     */
    private Icon historyIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/history.gif");
    /**
     * Icon for variable declaration.
     */
    private Icon varDeclIcon = ResourceLoader.createImageIcon(
```

```
    "kiel/browser/images/var.gif");
/**
 * Icon for local signal declaration.
 */
private Icon signalDeclIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/local.gif");
/**
 * BrowserModel to call methods.
 */
private BrowserModel model;

/**
 * set background and selection color.
 * @param m Model to call
 */
public MyRenderer(final BrowserModel m) {
  model = m;
  setBackground(Color.WHITE);
  setBackgroundSelectionColor(BrowserProperties.getNodeMarkColor());
}

/**
 * Called every time a tree cell must be rendered.
 * @param tree the tree component to be renderd
 * @param value the object representing the value
 * @param sel is this tree cell selected
 * @param expanded is this tree cell expanded
 * @param leaf is this tree cell a tree leaf
 * @param row the row
 * @param hasFocus has this cell the focus
 * @return this
 */
public Component getTreeCellRendererComponent(
    final JTree tree,
    final Object value,
    final boolean sel,
    final boolean expanded,
    final boolean leaf,
    final int row,
    final boolean hasFocus) {

  super.getTreeCellRendererComponent(
      tree, value, sel, expanded, leaf, row, hasFocus);

  Object n = ((DefaultMutableTreeNode) value).getUserObject();
  if (n != null) {
    if (n.getClass() == ANDState.class) {
      setText(createNode((ANDState) n));
      setIcon(andIcon);
      setToolTipText(((ANDState) n).getID());
    } else if (n.getClass() == ORState.class) {
      setText(createNode((ORState) n));
      setIcon(orIcon);
      setToolTipText(((ORState) n).getID());
    } else if (n.getClass() == Region.class) {
      setText(createNode((Region) n));
      setIcon(regionIcon);
      setToolTipText(((Region) n).getID());
    } else if (n.getClass() == InitialState.class) {
      setText(createNode((InitialState) n));
      setIcon(initialIcon);
      setToolTipText(((InitialState) n).getID());
    } else if (n.getClass() == Choice.class
             || n.getClass() == DynamicChoice.class) {
      setText(createNode((Node) n));
      setIcon(choiceIcon);
      setToolTipText(((Node) n).getID());
    } else if (n.getClass() == History.class
             || n.getClass() == DeepHistory.class) {
      setText(createNode((Node) n));
      setIcon(historyIcon);
      setToolTipText(((Node) n).getID());
    } else if (n.getClass() == FinalSimpleState.class) {
      setText(createNode((FinalSimpleState) n));
      setIcon(finalIcon);
      setToolTipText(((FinalSimpleState) n).getID());
    } else if (n.getClass() == FinalState.class) {
      setText(createNode((FinalState) n));
      setIcon(finalIcon);
      setToolTipText(((FinalState) n).getID());
    } else if (n.getClass() == FinalANDState.class) {
      setText(createNode((FinalANDState) n));
      setIcon(finalIcon);
      setToolTipText(((FinalANDState) n).getID());
    } else if (n.getClass() == FinalORState.class) {
      setText(createNode((FinalORState) n));
      setIcon(finalIcon);
      setToolTipText(((FinalORState) n).getID());
    } else if (n.getClass() == Suspend.class) {
      setText(createNode((Suspend) n));
      setIcon(suspendIcon);
      setToolTipText(((Suspend) n).getID());
    } else if (n.getClass() == SimpleState.class) {
      setText(createNode((SimpleState) n));
      setIcon(simpleIcon);
      setToolTipText(((SimpleState) n).getID());
    } else if (n instanceof Variable) {
      setText(((Variable) n).getName());
      setIcon(varDeclIcon);
      setToolTipText("");
    } else if (n instanceof Event) {
      setText(((Event) n).getName());
      setIcon(signalDeclIcon);
      setToolTipText("");
    } else {
      setToolTipText(""); //no tool tip
    }
    if (sel) {
      if (n instanceof Node) {
        model.getBrowserLogFile().log(LogFile.INFO,
                                      n.toString() + " was selected in Tree View");
        model.setMode(BrowserModel.EDIT);
        model.setFocus((Node) n, model.getSelectedStates()[1],
                                  model.getSelectedTransition());
      }
    }
  }
//    else if (n instanceof Node) {
//        model.removeMarkOnNode((Node) n);
//    }
```

```
    }

    return this;
  }
```

F.5.6. EdgePreferences

```
package kiel.browser.view;

import java.awt.Color;
import java.awt.Component;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Paint;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Observable;

import javax.swing.AbstractAction;
import javax.swing.DefaultListCellRenderer;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.SpinnerNumberModel;
import javax.swing.plaf.metal.MetalLookAndFeel;

import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.ConditionalTransition;
import kiel.dataStructure.Edge;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.StringLabel;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Suspension;
import kiel.dataStructure.Transition;
import kiel.dataStructure.WeakAbortion;
import kiel.browser.Browser;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;
import kiel.preferences.LogFile;
import kiel.util.main.KielLayouter;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: A edge preferences dialog. </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.15 £ last modified £Date: 2007/02/08 13:55:20 £
 */
public class EdgePreferences extends BrowserGUI {
  /**
   *
   */
  private JComboBox type;
  /**
   *
   */
  private JTextField source;
  /**
   *
   */
  private JTextField target;
  /**
   *
   */
  private JTextField label;
  /**
   *
   */
  private JTextField id;
  /**
   * For changing the prio.
   */
  private JSpinner prio;
  /**
   * Pref panel.
   */
  private JPanel panel;
  /**
   * Selected edge.
   */
  private Edge edge;

  /**
   * Name of edge types.
   */
  private String[] edgeTypes = {
                               "Edge",                // 0
                               "Transition",          // 1
                               "InitialArc",          // 2
                               "NormalTermination",   // 3
                               "StrongAbortion",      // 4
                               "WeakAbortion",        // 5
                               "Suspension",          // 6
                               "ConditionalTransition"}; // 7
  /**
   *
   * @param model BrowserModel
   */
  public EdgePreferences(final BrowserModel model) {
    super(model);
```

```
    try {
      jbInit();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  /**
   *
   * @param e Edge
   * @return int
   */
  private static int getIndex(final Edge e) {
    int index = 0;
    if (e.getClass() == Transition.class) {
      index = 1;
    } else if (e.getClass() == InitialArc.class) {
      index = 2;
    } else if (e.getClass() == NormalTermination.class) {
      index = 3;
    } else if (e.getClass() == StrongAbortion.class) {
      index = 4;
    } else if (e.getClass() == WeakAbortion.class) {
      index = 5;
    } else if (e.getClass() == Suspension.class) {
      index = 6;
    } else if (e.getClass() == ConditionalTransition.class) {
      index = 7;
    }
    return index;
  }

  /**
   *
   * @throws Exception ex
   */
  private void jbInit() throws Exception {
    final int height = 6;
    GridLayout grid1 = new GridLayout(0, 2);
    panel = new JPanel(grid1) {
      protected void paintComponent(final Graphics g) {
        boolean topToBottom = false;

        Color light = MetalLookAndFeel.getControl();
        Color dark = MetalLookAndFeel.getControlInfo();
        JComponent c = this;

        Paint oldPaint = ((Graphics2D) g).getPaint();

        if (topToBottom) {
          ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
            light, 0, c.getSize().height, dark, true));
        } else {
          ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
            dark, c.getSize().width, 0, light, true));
        }
        ((Graphics2D) g).fillRect(0, 0, c.getSize().width, c.getSize().height);
        ((Graphics2D) g).setPaint(oldPaint);
      }
    };
//    panel = new JPanel(grid1);

    JPanel left = new JPanel();
    left.setOpaque(false);
    left.setFocusCycleRoot(true);
    GridLayout grid2 = new GridLayout(height, 2);
    left.setLayout(grid2);
    left.add(new JLabel("Label:"));
    label = new JTextField();
    label.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).
      put(BrowserProperties.getKeyStroke("Browser.KeyStroke.ChangeEdgeLabel"),
          "nlabel");
    label.getActionMap().put("nlabel", new AbstractAction() {
      public void actionPerformed(final ActionEvent e) {
        label.requestFocusInWindow();
      }
    });
    label.addActionListener(new ActionListener() {
      public void actionPerformed(final ActionEvent e) {
        if (edge != null) {
          if (getModel().getStateChart().getModelSource().
              equals("Esterel_Studio")) {
            convertStringLabel(e);
          } else {
            ((Transition) edge).setLabel(new StringLabel(e.getActionCommand().
                                                         trim().replaceAll("\\\\n",
                                                                           "\n")));
          }
          updateChart();
          Browser.requestKeyEdit();
        }
      }

      private void convertStringLabel(ActionEvent e) {
        String label = e.getActionCommand().trim();
        if (!label.equals("")) {
          boolean hasTrigger = false;
          boolean hasAction = false;
          boolean hasCondition = false;
          boolean isImmediate = false;
          if (label.matches("[^\\s\\[/]+.*(\\[.*\\])?(/.*)?")) {
            hasTrigger = true;
          }
          if (label.matches(".*/.*")) {
            hasAction = true;
          }
          if (label.matches(".*\\[.*\\].*")) {
            hasCondition = true;
          }
          if (label.matches("\\s*#.*(\\[.*\\])?(/.*)?")) {
            isImmediate = true;
          }
          String[] answers = label.split("[#/\\[\\]]");
          if (isImmediate && answers.length > 1 &&
              !answers[0].trim().equals("")) {
            ((Transition)edge).setLabel(new StringLabel(label));
            return;
          }
          if (isImmediate && !hasTrigger) {
            ((Transition)edge).setLabel(new StringLabel(label));
```

```
            return;
          }
          if (answers.length >= 5) {
            ((Transition)edge).setLabel(new StringLabel(label));
            return;
          }
          CompoundLabel aComLabel = new CompoundLabel();
          CompoundLabelParser.setStateChart(getModel().getStateChart());
          CompoundLabelParser.setCompositeState(
            edge.getSource().getParent());
          ArrayList parts = new ArrayList();
          for (int i = 0; i < answers.length; i++) {
            if (!answers[i].trim().equals("")) {
              parts.add(answers[i].trim());
            }
          }
          try {
            if (hasTrigger) {
              aComLabel.setTrigger(
                CompoundLabelParser.parseDelayExpression(
                  ((String) parts.get(0)).trim(), isImmediate));
              parts.remove(0);
            }
            if (hasCondition) {
              aComLabel.setCondition(
                CompoundLabelParser.parseBooleanExpression(
                  ((String) parts.get(0)).trim()));
              parts.remove(0);
            }
            if (hasAction) {
              aComLabel.setEffect(
                CompoundLabelParser.parseActions(
                  ((String) parts.get(0)).trim()));
            }
            ((Transition) edge).setLabel(aComLabel);
          } catch (CompoundLabelException ex) {
            if (parts.size() > 0) {
              getModel().getBrowserLogFile().log(LogFile.ERROR,
                "KitFile Translation Edge Label Parsing ("
                + ((String) parts.get(0)).trim() + ") " + ex.toString());
            }
            ((Transition) edge).setLabel(new StringLabel(label));
          } catch (Exception ex) {
            if (parts.size() > 0) {
              getModel().getBrowserLogFile().log(LogFile.ERROR,
                "KitFile Translation Edge Label Parsing ("
                + ((String) parts.get(0)).trim() + ") " + ex.toString());
            }
            ((Transition) edge).setLabel(new StringLabel(label));
          }
        } else {
          ((Transition) edge).setLabel(new CompoundLabel());
        }
      }
    });

    left.add(label);
    left.add(new JLabel("ID:"));
    id = new JTextField();
    id.setEditable(false);
    id.setFocusable(false);
    id.setForeground(Color.GRAY);
    id.setBackground(MetalLookAndFeel.getControl());
    left.add(id);
    left.add(new JLabel("Type:"));
    type = new JComboBox(edgeTypes);
    type.addActionListener(new ActionListener() {
      private Transition getType(String type) {
        if (type.equals("Transition")) {
          return new Transition();
        } else if (type.equals("NormalTermination")) {
          return new NormalTermination();
        } else if (type.equals("WeakAbortion")) {
          return new WeakAbortion();
        } else if (type.equals("StrongAbortion")) {
          return new StrongAbortion();
        }
        return null;
      }
      public void actionPerformed(ActionEvent e) {
        if (edge != null && edge instanceof Transition) {
          JComboBox cb = (JComboBox) e.getSource();
          Transition t = getType((String) cb.getSelectedItem());
          if (t != null && t.getClass() != edge.getClass()
              && (edge.getClass() == Transition.class
              || edge.getClass() == WeakAbortion.class
              || edge.getClass() == StrongAbortion.class
              || edge.getClass() == NormalTermination.class)) {
            // type changed
            t.setLabel(((Transition) edge).getLabel());
            t.setSource(edge.getSource());
            t.setTarget(edge.getTarget());
            t.setPriority(((Transition) edge).getPriority());
            t.setIDManual(edge.getID());
            replaceEdge(t);
          } else if (t == null) {
            getModel().setStatus("Can only be changed to"
                                 + " Transition, Strong- or WeakAbortion and "
                                 + "NormalTermination");
          }
        }
      }
    });
    type.setForeground(Color.BLACK);
    type.setBackground(MetalLookAndFeel.getControl());
    type.setRenderer(new DefaultListCellRenderer() {
      public Component getListCellRendererComponent(final JList list,
        final Object value,
        final int index,
        final boolean isSelected,
        final boolean cellHasFocus) {
        JLabel typeLabel = new JLabel();
        typeLabel.setOpaque(true);
        if (value != null) {
          typeLabel.setText(value.toString());
          String s = value.toString();
          //  if (s.startsWith("Edge")) {
          //  } else if (s.startsWith("Transition")) {
          //  } else if (s.startsWith("InitialArc")) {
          if (s.startsWith("StrongAbort")) {
```

```
            typeLabel.setForeground(Color.BLACK);
          } else if (s.startsWith("NormalTerm")) {
            typeLabel.setForeground(Color.BLACK);
          } else if (s.startsWith("WeakAbort")) {
            typeLabel.setForeground(Color.BLACK);
          } else {
            typeLabel.setForeground(Color.GRAY);
          }
          //  } else if (s.startsWith("Suspens")) {
          //  } else if (s.startsWith("Cond")) {
          //  else {
          //      label.setIcon(simpleIcon);
          //  }
          typeLabel.setBackground(
            isSelected ? BrowserProperties.getNodeMarkColor() : Color.white);
        }
        return typeLabel;
      }
    });

    left.add(type);

    left.add(new JLabel("Source:"));
    source = new JTextField();
    source.setEditable(false);
    source.setFocusable(false);
    source.setForeground(Color.GRAY);
    source.setBackground(MetalLookAndFeel.getControl());
    left.add(source);

    left.add(new JLabel("Target:"));
    target = new JTextField();
    target.setEditable(false);
    target.setFocusable(false);
    target.setForeground(Color.GRAY);
    target.setBackground(MetalLookAndFeel.getControl());
    left.add(target);

    left.add(new JLabel("Priority:"));
    prio = new JSpinner(new SpinnerNumberModel(1, 0, Integer.MAX_VALUE, 1));
    for (int i = 0; i < prio.getComponentCount(); i++) {
      if (prio.getComponent(i) instanceof JButton) {
//        prio.getComponent(i).setForeground(Color.BLACK);
        prio.getComponent(i).setBackground(MetalLookAndFeel.getControl());
      }
    }
    prio.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        int p = ((Integer) prio.getValue()).intValue();
        if (edge != null && edge instanceof Transition
            && ((Transition) edge).getPriority().getValue() != p) {
          ((Transition) edge).setPriority(p);
          updateChart();
          Browser.requestKeyEdit();
        }
      }
    });

    left.add(prio);

    JPanel right = new JPanel();
    right.setOpaque(false);
    GridLayout grid3 = new GridLayout(height, 2);
    right.setLayout(grid3);

    panel.add(left);
    panel.add(right);
  }

  /**
   * @return JComponent
   */
  public final JComponent getComponent() {
    return panel;
  }

  /**
   * Updates the chart.
   */
  private void updateChart() {
    Hashtable changed = new Hashtable(1);
    changed.put(edge.getID(), edge);
    getModel().setChangedObjects(null,
                                 null,
                                 changed.values(),
                                 this.getClass().getName());
    KielLayouter layouter = Browser.getKielFrame().
                            getKielLayouterByName(
                            Browser.getKielFrame().
                            getDefaultLayouterName());
    try {
      layouter.setStatechart(getModel().getStateChart());
      layouter.layoutView(layouter.getStaticView());
      getModel().setLayouter(layouter, Browser.getKielFrame().
                              getDefaultLayouterName());
      getModel().showStaticView();
      getModel().setStatus("Operation done");
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  /**
   * Updates the chart.
   */
  private void replaceEdge(Transition t) {
    ArrayList removed = new ArrayList(1);
    removed.add(edge);
    ArrayList added = new ArrayList(1);
    added.add(t);
    getModel().setChangedObjects(removed,
                                 added,
                                 null,
                                 this.getClass().getName());
    KielLayouter layouter = Browser.getKielFrame().
                            getKielLayouterByName(
                            Browser.getKielFrame().
                            getDefaultLayouterName());
    try {
      layouter.setStatechart(getModel().getStateChart());
```

```
        layouter.layoutView(layouter.getStaticView());
        getModel().setLayouter(layouter, Browser.getKielFrame().
                getDefaultLayouterName());

        getModel().showStaticView();
        edge = t;
        getModel().setFocus(getModel().getFocus().getNodes()[0],
          getModel().getFocus().getNodes()[1],
          edge);
        getModel().setStatus("Operation done");
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }

    /**
     *
     * @param o Observable
     * @param arg Object
     */
    public final void update(final Observable o, final Object arg) {
      if (arg.equals(BrowserModel.FOCUS_CHANGED)
          && getModel().getFocus().hasEdgeChanged()
          && getModel().getFocus().getEdge() != null) {
        edge = getModel().getSelectedTransition();
        id.setText(edge.getID());
        type.setSelectedIndex(getIndex(edge));
        source.setText(edge.getSource().toString());
        target.setText(edge.getTarget().toString());
        if (edge instanceof Transition) {
          Transition trans = (Transition) edge;
          prio.setValue(new Integer(trans.getPriority().getValue()));
          label.setText(trans.getLabel().toString().replaceAll("\n","\\\\n"));
        } else {
          prio.setValue(new Integer(0));
          label.setText("");
        }
      }
    }
  }
```

F.5.7. GraphicalObjectsLibrary

```
// £Id: GraphicalObjectsLibrary.java,v 1.29 2007/02/08 13:55:20 khe Exp £
package kiel.browser.view;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.Properties;

import kiel.browser.view.rendering.CircleNode;
import kiel.browser.view.rendering.GroupNode;
import kiel.browser.view.rendering.LineNode;
import kiel.browser.view.rendering.PathNode;
import kiel.browser.view.rendering.RectNode;
import kiel.browser.view.rendering.TextNode;
import kiel.browser.view.rendering.Toolkit;
import kiel.preferences.LogFile;
import org.csiro.svg.dom.SVGDocumentImpl;
import org.csiro.svg.dom.SVGSVGElementImpl;
import org.csiro.svg.parser.SVGParseException;
import org.csiro.svg.parser.SVGParser;
import org.w3c.dom.Node;
import org.w3c.dom.Text;
import org.w3c.dom.svg.SVGCircleElement;
import org.w3c.dom.svg.SVGElement;
import org.w3c.dom.svg.SVGEllipseElement;
import org.w3c.dom.svg.SVGGElement;
import org.w3c.dom.svg.SVGLineElement;
import org.w3c.dom.svg.SVGPathElement;
import org.w3c.dom.svg.SVGRectElement;
import org.w3c.dom.svg.SVGTextElement;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

/**
 * <p>Description: The graphical objects library loads the templates from
 * resoure or files and caches them.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.29 £ last modified £Date: 2007/02/08 13:55:20 £
 *
 */
public class GraphicalObjectsLibrary {
  /**
   *  Identifier if using default templates.
   */
  public static final int DEFAULT_TEMPLATES = 0;
  /**
   *  Identifier if using estudio templates.
   */
  public static final int ESTUDIO_TEMPLATES = 1;
  /**
   *  Identifier if using stateflow templates.
   */
  public static final int STATEFLOW_TEMPLATES = 2;
  /**
   * Identifier if using UML templates.
   */
  public static final int UML_TEMPLATES = 3;
  /**
   *  Identifier reading estudio templates from.
   */
  private static final String ESTUDIO = "estudio.ini";
  /**
   *  Identifier reading stateflow templates from.
   */
  private static final String STATEFLOW = "stateflow.ini";
  /**
   *  Identifier reading default templates from.
   */
  private static final String DEFAULT_LAYOUT = "default.ini";
  /**
   *
   */
  private static final String UML = "uml.ini";
  /**
   *
   */
  private static final String DEFAULT_PSEUDO_WIDTH =
    "PSEUDOSTATE.DEFAULT.WIDTH";
  /**
   *
   */
  private static final String DEFAULT_PSEUDO_HEIGHT =
    "PSEUDOSTATE.DEFAULT.HEIGHT";
  /**
   *
   */
  private static final String DEFAULT_STATE_WIDTH = "STATE.DEFAULT.WIDTH";
  /**
   *
   */
  private static final String DEFAULT_STATE_HEIGHT = "STATE.DEFAULT.HEIGHT";
  /**
   * Properties identifier for the state.
   */
  public static final String ANDSTATE = "SVG.ANDState";
  /**
   * Properties identifier for the state.
   */
  public static final String ORSTATE = "SVG.ORState";
  /**
   * Properties identifier for the state.
   */
  public static final String SIMPLESTATE = "SVG.SimpleState";
  /**
   * Properties identifier for the state.
   */
```

```
    public static final String INITALSTATE = "SVG.Initial";
    /**
     * Properties identifier for the state.
     */
    public static final String SUSPENDSTATE = "SVG.Suspend";
    /**
     * Properties identifier for the state.
     */
    public static final String HISTORYSTATE = "SVG.History";
    /**
     * Properties identifier for the state.
     */
    public static final String DEEPHISTORYSTATE = "SVG.DeepHistory";
    /**
     * Properties identifier for the state.
     */
    public static final String JUNCTIONSTATE = "SVG.Junction";
    /**
     * Properties identifier for the state.
     */
    public static final String CHOICESTATE = "SVG.Choice";
    /**
     * Properties identifier for the state.
     */
    public static final String FINALSTATE = "SVG.Final";
    /**
     * Properties identifier for the state.
     */
    public static final String FINALCOMPOSITESTATE = "SVG.FinalComposite";
    /**
     * Cached andstate template.
     */
    private GroupNode andstate = null;
    /**
     * Cached orstate template.
     */
    private GroupNode orstate = null;
    /**
     * Cached simplestate template.
     */
    private GroupNode simplestate = null;
    /**
     * Cached initialstate template.
     */
    private GroupNode initialstate = null;
    /**
     * Cached suspendstate template.
     */
    private GroupNode suspendstate = null;
    /**
     * Cached historystate template.
     */
    private GroupNode histroystate = null;
    /**
     * Cached choicestate template.
     */
    private GroupNode choicestate = null;
    /**
     * Cached junctionstate template.
     */
    private GroupNode junctionstate = null;
    /**
     * Cached finalstate template.
     */
    private GroupNode finalstate = null;
    /**
     * Cached final compisite state template.
     */
    private GroupNode finalcompstate = null;
    /**
     * Cached deep history state template.
     */
    private GroupNode deephistorystate = null;

    /**
     * CSS Style for the rect shaped states.
     */
    public static final String STYLE_RECT = "CSS.RECT";
    /**
     * CSS Style for the circle shaped states.
     */
    public static final String STYLE_CIRCLE = "CSS.CIRCLE";
    /**
     * CSS Style for the transition labels.
     */
    //public static final String STYLE_LABEL = "CSS.LABEL";
    /**
     * CSS Style for the dellines.
     */
    public static final String STYLE_LINE = "CSS.LINE";
    /**
     * CSS Style for the header.
     */
    public static final String STYLE_HEADER = "CSS.HEADER";

    /**
     * the GO library has it's own LogFile (by default switched off).
     */
    private LogFile l;
    /**
     * path to ini file.
     */
    private String path;
    /**
     * this properties class holds the values.
     */
    private Properties init;
    /**
     * actual loaded ini file.
     */
    private String initfile;
    /**
       /**
      * is the library valid (all fields known???).
      */
     private boolean valid = false;
    /**
     * are we using resources or files.
     */
    private boolean usingResource = false;
```

```
  /**
   * resource path for loading.
   */
  private static final String RESOURCE = "kiel/browser/library/";
  /**
   *  Ini file for loading.
   */
  private static final String DEFAULT = System.getProperty("user.home")
                                        + File.separator + ".kiel"
                                        + File.separator;

  /**
   *  File used for reading.
   */
  private String file;

  /**
   * <code>Singleton</code>.
   */
  private static GraphicalObjectsLibrary instance = null;

  /**
   * Actual used toolkit.
   */
  private Toolkit tk = null;
  /**
   * Type of library loaded.
   */
  private int actualType;

  /**
   * Creates new library from file or resource: <br>
   * If you have both the supplied resource and your own ini file
   * ~/.kiel/estudio.ini the ini file is perferred.
   * @param aToolkit creates instance with a Toolkit.
   */
  public GraphicalObjectsLibrary(final Toolkit aToolkit) {
    tk = aToolkit;
    l = new LogFile(new PrintWriter(System.out));
    l.setLogLevel(LogFile.DEBUG);
    l.disableLog();
    l.setModuleName("GO Library");

    l.log(LogFile.DEBUG, "Testing whether there is a resource or a ini file");
    l.log(LogFile.DEBUG, "Preferring a ini File");
    setFile(ESTUDIO_TEMPLATES);
    actualType = ESTUDIO_TEMPLATES;
    init = new Properties();
    loadDefaults();
  }

  /**
   * @param type int set file from type.
   */
  private void setFile(final int type) {
    if (type == ESTUDIO_TEMPLATES) {
      file = ESTUDIO;
    } else if (type == STATEFLOW_TEMPLATES) {
      file = STATEFLOW;
    } else if (type == UML_TEMPLATES) {
      file = UML;
    } else {
      file = DEFAULT_LAYOUT;
    }
  }
  /**
   * @param type int type of templates.s
   */
  public final void setTemplateType(final int type) {
    if (actualType != type) {
      setFile(type);
      clearCache();
      System.gc();
      loadDefaults();
      actualType = type;
    }
    changeDefaultDimensions();
  }

  /**
   * @return static instance of GraphicalObjectsLibrary
   *
   * Class is implemented as <code>singleton</code>
   * @param aToolkit instance with a Toolkit.
   */
  public static final GraphicalObjectsLibrary getInstance(
    final Toolkit aToolkit) {
    if (instance == null) {
      instance = new GraphicalObjectsLibrary(aToolkit);
      return instance;
    }
    return instance;
  }

  /**
   * Set a new toolkit (must clear cache).
   * @param aTk Toolkit
   */
  public final void setNewToolkit(final Toolkit aTk) {
    tk = aTk;
    clearCache();
  }

  /**
   * @param f svg file to be loaded
   * @return the first group element in the svg file f
   */
  public final GroupNode getGroupFromFile(final String f) {
    SVGParser parser = new SVGParser();
    SVGDocumentImpl doc = null;
    try {
      if (usingResource) {
        try {
          parser.setErrorHandler(parser);
          parser.setEntityResolver(parser);
          parser.parse(new InputSource(getClass().getClassLoader().
                                       getResource(f).openStream()));
          doc = new SVGDocumentImpl(parser.getDocument());
        } catch (NullPointerException ex0) {
          l.log(LogFile.ERROR, "Error during parsing Resource "
```

```
                + f + " "
                + ex0.getMessage());
            System.err.println("Error during parsing Resource "
                + f + " "
                + ex0.getMessage());
            return null;
          } catch (IOException ex1) {
            l.log(LogFile.ERROR, "IOError during parsing Resource "
                + f + " "
                + ex1.getMessage());
            return null;
          } catch (SAXException ex2) {
            l.log(LogFile.ERROR, "SaxError during parsing Resource " + f);
            return null;
          }
        } else {
          doc = parser.parseSVG(f);
        }
      } catch (SVGParseException ex) {
        l.log(LogFile.ERROR, "Error during parsing File " + f);
        return null;
      }
      SVGSVGElementImpl root = (SVGSVGElementImpl) doc.getRootElement();
      for (int i = 0; i < root.getChildNodes().getLength(); i++) {
        Node dummy = root.getChildNodes().item(i);
        if (dummy instanceof SVGGElement) {
          return (GroupNode) convertToAbstractToolkit((SVGGElement) dummy);
        }
      }
      return null;
    }

    /**
     *
     * @param aText SVGTextElement
     * @return String
     */
    private static String getText(final SVGTextElement aText) {
      String value = null;
      for (int i = 0; i < aText.getChildNodes().getLength(); i++) {
        if (aText.getChildNodes().item(i) instanceof Text) {
          value = ((Text) aText.getChildNodes().item(i)).getData();
          break;
        }
      }
      return value;
    }

    /**
     * convertToAbstractToolkit.
     *
     * @param root Node
     * @return GroupNode
     */
    private kiel.browser.view.rendering.Node
      convertToAbstractToolkit(final SVGElement root) {
      kiel.browser.view.rendering.Node node;
      kiel.browser.view.rendering.Node child = null;
      if (root instanceof SVGGElement) {
        node = tk.createGroupElement();
      } else if (root instanceof SVGRectElement) {
        node = tk.createRectElement();
        tk.getStyleChanger().changeStyle(node, getStyle(STYLE_RECT));
        ((RectNode) node).setRx((int) Float.parseFloat(root.getAttribute("rx")));
        ((RectNode) node).setRy((int) Float.parseFloat(root.getAttribute("ry")));
//        System.out.println("Setting rx/ry to " + ((RectNode) node).getRx() + "/"
//                           + ((RectNode) node).getRy());
      } else if (root instanceof SVGEllipseElement
                 || root instanceof SVGCircleElement) {
        node = tk.createCircleElement();
        tk.getStyleChanger().changeStyle(node, getStyle(STYLE_CIRCLE));
      } else if (root instanceof SVGPathElement) {
        node = tk.createPathElement();
        tk.getStyleChanger().changeStyle(node, getStyle(STYLE_LINE));
      } else if (root instanceof SVGLineElement) {
        node = tk.createLineElement();
        tk.getStyleChanger().changeStyle(node, getStyle(STYLE_LINE));
      } else if (root instanceof SVGTextElement) {
        node = tk.createTextElement();
        String v = getText((SVGTextElement) root);
        if (v != null && v.length() > 0) {
          ((TextNode) node).setText(v);
        }
      } else {
        return null;
      }

      node.setId(root.getId());

      if (!root.getAttribute("style").equals("")) {
        tk.getStyleChanger().changeStyle(node, root.getAttribute("style"));
      }

      for (int i = 0; i < root.getChildNodes().getLength(); i++) {
        Node dummy = root.getChildNodes().item(i);
        if (dummy instanceof SVGElement) {
          child = convertToAbstractToolkit((SVGElement) dummy);
          if (child != null) {
            node.appendChild(child);
          }
        }
      }
      return node;
    }

    /**
     * loads the default values from resources included in browser.jar.
     */
    private void loadDefaults() {
      try {
        InputStream stream = new FileInputStream(new File(DEFAULT + file));
        init.load(stream);
        stream.close();
        valid = true;
        usingResource = false;
        initfile = DEFAULT + file;
        cacheTemplates();
      } catch (IOException ex) {
        try {
          l.log(LogFile.DEBUG, "Trying to load graphical objects as resource: "
```

```
            + RESOURCE + file);
        if (getClass().getClassLoader().
            getResource(RESOURCE + file) != null) {
          InputStream stream = getClass().getClassLoader().
                               getResource(RESOURCE + file).openStream();
          init.load(stream);
          stream.close();
          valid = true;
          usingResource = true;
          l.log(LogFile.DEBUG, "resource is valid ->");
          initfile = RESOURCE + file;
          cacheTemplates();
        } else {
          l.log(LogFile.ERROR, "Resource could not be found ->"
                + " GraphicalLib is not valid");
          valid = false;
          usingResource = false;
        }
      } catch (IOException ex2) {
        l.log(LogFile.ERROR, "Resource could not be found ->"
              + " GraphicalLib is not valid");
        valid = false;
        usingResource = false;
        return;
      }
    }
  }

  /**
   *  Overwrites the default sizes for states and pseudostates in an ini file.
   */
  private void changeDefaultDimensions() {
    String val = init.getProperty(DEFAULT_PSEUDO_HEIGHT);
    if (val != null) {
      try {
        int i = Integer.parseInt(val);
        kiel.graphicalInformations.Properties.setPseudoStateDefaultHeight(i);
      } catch (NumberFormatException ex) {
        l.log(LogFile.ERROR, "Error parsing default pseudo height " + ex);
      }
    }
    val = init.getProperty(DEFAULT_PSEUDO_WIDTH);
    if (val != null) {
      try {
        int i = Integer.parseInt(val);
        kiel.graphicalInformations.Properties.setPseudoStateDefaultWidth(i);
      } catch (NumberFormatException ex) {
        l.log(LogFile.ERROR, "Error parsing default pseudo width " + ex);
      }
    }
    val = init.getProperty(DEFAULT_STATE_HEIGHT);
        if (val != null) {
          try {
            int i = Integer.parseInt(val);
            kiel.graphicalInformations.Properties.setStateDefaultHeight(i);
          } catch (NumberFormatException ex) {
            l.log(LogFile.ERROR, "Error parsing default height " + ex);
          }
    }
    val = init.getProperty(DEFAULT_STATE_WIDTH);
    if (val != null) {
      try {
        int i = Integer.parseInt(val);
        kiel.graphicalInformations.Properties.setStateDefaultWidth(i);
      } catch (NumberFormatException ex) {
        l.log(LogFile.ERROR, "Error parsing default width " + ex);
      }
    }
  }

  /**
   * caches all templates.
   */
  private void cacheTemplates() {
    andstate = getTemplate(ANDSTATE);
    orstate = getTemplate(ORSTATE);
    simplestate = getTemplate(SIMPLESTATE);
    initialstate = getTemplate(INITALSTATE);
    suspendstate = getTemplate(SUSPENDSTATE);
    junctionstate = getTemplate(JUNCTIONSTATE);
    histroystate = getTemplate(HISTORYSTATE);
    deephistorystate = getTemplate(DEEPHISTORYSTATE);
    choicestate = getTemplate(CHOICESTATE);
    finalstate = getTemplate(FINALSTATE);
    finalcompstate = getTemplate(FINALCOMPOSITESTATE);
  }

  /**
   * clears the cached templates.
   */
  private void clearCache() {
    andstate = null;
    orstate = null;
    simplestate = null;
    initialstate = null;
    suspendstate = null;
    histroystate = null;
    deephistorystate = null;
    choicestate = null;
    finalstate = null;
    finalcompstate = null;
    junctionstate = null;
  }

  /**
   * @param f loads ini file for templates from file "file"
   */
  public final void setInitfile(final String f) {
    l.log(LogFile.DEBUG, "Trying to load graphical objects library: " + f);
    try {
      init.load(new FileInputStream(new File(f)));
      valid = true;
      usingResource = false;
    } catch (IOException ex) {
      l.log(LogFile.ERROR, "Error loading ini file: " + f
            + " trying defaults");
      loadDefaults();
      return;
    }
    initfile = f;
```

```
    cacheTemplates();
  }

  /**
   * The templates are only loaded once to speed up showing
   * so you call this method how often you like :-).
   * @param s type of template you want e.g.<br>
   * ANDSTATE,<br>
   * ORSTATE,<br>
   * SIMPLESTATE,<br>
   * etc.
   * @return SVG group element containing the template
   */
  public final GroupNode getTemplate(final String s) {
    if (s.equals(ANDSTATE) && andstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(andstate);
    } else if (s.equals(ORSTATE) && orstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(orstate);
    } else if (s.equals(SIMPLESTATE) && simplestate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(simplestate);
    } else if (s.equals(INITALSTATE) && initialstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(initialstate);
    } else if (s.equals(FINALSTATE) && finalstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(finalstate);
    } else if (s.equals(CHOICESTATE) && choicestate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(choicestate);
    } else if (s.equals(SUSPENDSTATE) && suspendstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(suspendstate);
    } else if (s.equals(JUNCTIONSTATE) && junctionstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(junctionstate);
    } else if (s.equals(HISTORYSTATE) && histroystate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(histroystate);
    } else if (s.equals(FINALCOMPOSITESTATE) && finalcompstate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(finalcompstate);
    } else if (s.equals(DEEPHISTORYSTATE) && deephistorystate != null) {
      l.log(1, "Cached " + s);
      return (GroupNode) createClone(deephistorystate);
    }
    l.log(1, "Getting Template for: " + s);
    String svgfile = init.getProperty(s);
    return getGroupFromFile(initfile.substring(0, initfile.lastIndexOf("/") + 1)
                            + svgfile);
  }

  /**
   * Clone the given node.
   * @param root Node
   * @return Node
   */
  private kiel.browser.view.rendering.Node
    createClone(final kiel.browser.view.rendering.Node root) {

    kiel.browser.view.rendering.Node node;
    kiel.browser.view.rendering.Node child = null;
    kiel.browser.view.rendering.Node dummy = null;
    if (root instanceof GroupNode) {
      node = tk.createGroupElement();
    } else if (root instanceof RectNode) {
      node = tk.createRectElement();
      ((RectNode) node).setRx(((RectNode) root).getRx());
      ((RectNode) node).setRy(((RectNode) root).getRy());
    } else if (root instanceof CircleNode) {
      node = tk.createCircleElement();
    } else if (root instanceof PathNode) {
      node = tk.createPathElement();
    } else if (root instanceof LineNode) {
      node = tk.createLineElement();
    } else if (root instanceof TextNode) {
      node = tk.createTextElement();
      ((TextNode) node).setText(((TextNode) root).getText());

    } else {
      return null;
    }

    node.setId(root.getId());
    tk.getStyleChanger().changeStyle(node, root.getStyle());
    Iterator iter = root.getChildNodes().iterator();
    while (iter.hasNext()) {
      dummy = (kiel.browser.view.rendering.Node) iter.next();
      child = createClone(dummy);
      if (child != null) {
        node.appendChild(child);
      }
    }
    return node;
  }

  /**
   * @return if library is valid
   */
  public final boolean isValid() {
    l.log(LogFile.DEBUG, "Valid: " + valid);
    return valid;
  }

  /**
   * @param s type of style you want e.g.<br>
   * STYLE_RECT,<br>
   * STYLE_CIRCLE,<br>
   * etc.
   * @return CSS2 style as string
   */
  public final String getStyle(final String s) {
    return init.getProperty(s);
  }

  /**
   * @param s type of color you want e.g. <br>
   * COLOR_ACTIVE<br>
```

```
 * COLOR_ONEXIT
 * @return CSS2 Color value as string
 */
public final String getColor(final String s) {
  String c = init.getProperty(s);
  if (c == null) {
    return "none";
  }
  return c;
 }
}
```

F.5.8. NodePreferences

```
package kiel.browser.view;

import java.awt.Color;
import java.awt.Component;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Paint;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Observable;

import javax.swing.AbstractAction;
import javax.swing.DefaultListCellRenderer;
import javax.swing.Icon;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.plaf.metal.MetalLookAndFeel;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.State;
import kiel.browser.Browser;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.util.main.KielLayouter;
import kiel.dataStructure.eventexp.Event;
import java.util.Iterator;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Variable;
import kiel.dataStructure.action.Action;
import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.Transition;
import kiel.util.CompoundLabelException;
import kiel.dataStructure.StringLabel;
import kiel.util.CompoundLabelParser;
import kiel.dataStructure.intexp.IntegerVariable;
import kiel.util.declaration.DeclarationParser;
import java.util.Collection;
import kiel.util.declaration.*;
import kiel.dataStructure.eventexp.Signal;
import kiel.dataStructure.action.GenerateEvent;
import kiel.dataStructure.SimpleState;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Creates a node preferences dialog.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.20 £ last modified £Date: 2007/02/06 18:23:22 £
 */
public class NodePreferences extends BrowserGUI {
  /**
   * Textfield for preference.
   */
  private JPanel panel;
  /**
   * Textfield for preference.
   */
  private JTextField name;
  /**
   * Textfield for preference.
   */
  private JTextField id;
  /**
   * Textfield for preference.
   */
  private JTextField internal;
  /**
   * Box for preference.
   */
  private JComboBox type;
  /**
   * Box for preference.
   */
  private PlusMinusComboBox out;
  /**
   * Box for preference.
   */
  private PlusMinusComboBox in;
  /**
   * Box for preference.
   */
  private PlusMinusComboBox childs;
```

```
    /**
     * Box for preference.
     */
    private PlusMinusComboBox vars;
    /**
     * Box for preference.
     */
    private PlusMinusComboBox events;
    /**
     * Box for preference.
     */
    private PlusMinusComboBox exit;
    /**
     * Box for preference.
     */
    private PlusMinusComboBox entry;
    /**
     * Box for preference.
     */
    private PlusMinusComboBox doActivity;
    /**
     * Box for preference.
     */
    private PlusMinusComboBox bind;
    /**
     * Node.
     */
    private Node node;
    /**
     * flag.
     */
    private boolean addInEdges = false;
    /**
     * flag.
     */
    private boolean addOutEdges = false;
    /**
     * flag.
     */
    private boolean addChildNodes = false;

    /**
     * Created a node preferences dialog.
     * @param model BrowserModel
     */
    public NodePreferences(final BrowserModel model) {
      super(model);
      node = null;
      try {
        jbInit();
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }

    /**
     * @throws Exception if error
     */
    private void jbInit() throws Exception {
      GridLayout grid1 = new GridLayout(0, 2);
      panel = new JPanel(grid1) {
        protected void paintComponent(final Graphics g) {
          boolean topToBottom = false;

          Color light = MetalLookAndFeel.getControl();
          Color dark = MetalLookAndFeel.getControlInfo();
          JComponent c = this;

          Paint oldPaint = ((Graphics2D) g).getPaint();

          if (topToBottom) {
            ((Graphics2D) g).setPaint(new GradientPaint(0, 0, light, 0,
              c.getSize().height, dark, true));
          } else {
            ((Graphics2D) g).setPaint(new GradientPaint(0, 0, dark,
              c.getSize().width, 0, light, true));
          }
          ((Graphics2D) g).fillRect(0, 0, c.getSize().width, c.getSize().height);
          ((Graphics2D) g).setPaint(oldPaint);
        }
      };
      panel.setFocusCycleRoot(true);
      JPanel left = new JPanel();
      left.setOpaque(false);
      final int height = 7;
      GridLayout grid2 = new GridLayout(7, 2);
      left.setLayout(grid2);
      left.add(new JLabel("Name:"));
      name = new JTextField();
      name.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).
        put(BrowserProperties.getChangeNodeLabel(), "nlabel");
      name.getActionMap().put("nlabel", new AbstractAction() {
        public void actionPerformed(final ActionEvent e) {
          name.requestFocusInWindow();
        }
      });
      name.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
          String aName = e.getActionCommand().trim();
          if (node != null) {
            node.setName(aName);
            updateChart();
            Browser.requestKeyEdit();
          }
        }
      });
      left.add(name);
      left.add(new JLabel("ID:"));
      id = new JTextField();
      id.setEditable(false);
      id.setFocusable(false);
      id.setForeground(Color.GRAY);
      id.setBackground(MetalLookAndFeel.getControl());
      left.add(id);
      left.add(new JLabel("Type:"));
      type = new JComboBox(stateTypes);
      type.setForeground(Color.BLACK);
      type.setBackground(MetalLookAndFeel.getControl());
      type.setRenderer(new DefaultListCellRenderer() {
        /**
```

```
     * sample andicon.
     */
    private Icon andIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/and.gif");
    /**
     * Icon for orstate.
     */
    private Icon orIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/or.gif");
    /**
     * Icon for simple state.
     */
    private Icon simpleIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/simple.gif");
    /**
     * Icon for inital state.
     */
    private Icon initialIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/init.gif");
    /**
     * Icon for choice state.
     */
    private Icon choiceIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/choice.gif");
    /**
     * Icon for region state.
     */
    private Icon regionIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/region.gif");
    /**
     * Icon for suspend state.
     */
    private Icon suspendIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/suspend.gif");
    /**
     * Icon for final state.
     */
    private Icon finalIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/final.gif");
    /**
     * Icon for history state.
     */
    private Icon historyIcon = ResourceLoader.createImageIcon(
      "kiel/browser/images/history.gif");
    public Component getListCellRendererComponent(
      final JList list,
      final Object value,
      final int index,
      final boolean isSelected,
      final boolean cellHasFocus) {
      JLabel label = new JLabel();
      label.setOpaque(true);
      if (value != null) {
        label.setText(value.toString());
        String s = value.toString();
        if (s.startsWith("AND")) {
          label.setIcon(andIcon);
        } else if (s.startsWith("OR")) {
          label.setIcon(orIcon);
        } else if (s.startsWith("Reg")) {
          label.setIcon(regionIcon);
        } else if (s.startsWith("Fin")) {
          label.setIcon(finalIcon);
        } else if (s.startsWith("Init")) {
          label.setIcon(initialIcon);
        } else if (s.startsWith("Choice") || s.startsWith("Dynam")) {
          label.setIcon(choiceIcon);
        } else if (s.startsWith("Hist") || s.startsWith("Deep")) {
          label.setIcon(historyIcon);
        } else {
          label.setIcon(simpleIcon);
        }
        label.setBackground(
          isSelected ? BrowserProperties.getNodeMarkColor() : Color.white);
      }
      return label;
    }
  });
  type.addActionListener(new ActionListener() {
// "Simple", "ANDState", "ORState", "Region",
// "FinalState", "FinalANDState", "FinalORState",
// "Initial", "Choice", "DynamicChoice", "History",
// "DeepHistory"
    private Node getType(String type) {
      if (type.equals("Simple")) {
        return new SimpleState();
      } else if (type.equals("ANDState")) {
        return new ANDState();
      } else if (type.equals("ORState")) {
        return new ORState();
      } else if (type.equals("FinalState")) {
        return new FinalSimpleState();
      } else if (type.equals("FinalANDState")) {
        return new FinalANDState();
      } else if (type.equals("FinalORState")) {
        return new FinalORState();
      } else if (type.equals("Initial")) {
        return new InitialState();
      } else if (type.equals("Choice")) {
        return new Choice();
      } else if (type.equals("DynamicChoice")) {
        return new DynamicChoice();
      } else if (type.equals("History")) {
        return new History();
      } else if (type.equals("DeepHistory")) {
        return new DeepHistory();
      }
      return null;
    }
    public void actionPerformed(ActionEvent e) {
      if (node != null) {
        JComboBox cb = (JComboBox) e.getSource();
        Node n = getType((String) cb.getSelectedItem());
        if (n != null && n.getClass() != node.getClass()
           && (node.getClass() != Region.class)) {
          // type changed
          Iterator iter = node.getIncomingTransitions().iterator();
          Edge e1;
          while (iter.hasNext()) {
            e1 = (Edge) iter.next();
```

```
            e1.setTarget(n);
          }
          iter = node.getOutgoingTransitions().iterator();
          while (iter.hasNext()) {
            e1 = (Edge) iter.next();
            e1.setSource(n);
          }
          if (node.getParent() != null) {
            n.setParent(node.getParent());
            node.getParent().removeSubnode(node);
            n.getParent().addSubnode(n);
          }
          if (n instanceof State && node instanceof State) {
            ((State) n).addConstants(((State) node).getConstants());
            ((State) n).addVariables(((State) node).getVariables());
            ((State) n).addLocalEvents(((State) node).getLocalEvents());
            ((State) n).setBindAction(((State) node).getBindAction());
            ((State) n).setEntry(((State) node).getEntry());
            ((State) n).setDoActivity(((State) node).getDoActivity());
            ((State) n).setExit(((State) node).getExit());
            ((State) n).setInternalTransition(
              ((State) node).getInternalTransition());
            ((State) n).setExit(((State) node).getExit());
          }
          n.setName(node.getName());
          n.setIDManual(node.getID());
          replaceNode(n);
        }
      }
    }
  });

  left.add(type);
  left.add(new JLabel("Incoming:"));
  in = new PlusMinusComboBox(new ArrayList(), false, true);
  in.addItemListener(new ItemListener() {
    public void itemStateChanged(final ItemEvent e) {
      if (e.getStateChange() == ItemEvent.SELECTED
          && !addInEdges) {
        if (in.getSelected() != null
            && in.getSelected() instanceof Edge) {
          Node[] nodes = getModel().getSelectedStates();
          getModel().setFocus(nodes[0], nodes[1], (Edge) in.getSelected());
        }
      } else if (addInEdges) {
        addInEdges = false;
      }
    }
  });
  left.add(in);
  left.add(new JLabel("Outgoing:"));
  out = new PlusMinusComboBox(new ArrayList(), false, true);
  out.addItemListener(new ItemListener() {
    public void itemStateChanged(final ItemEvent e) {
      if (e.getStateChange() == ItemEvent.SELECTED
          && !addOutEdges) {
        if (out.getSelected() != null
            && out.getSelected() instanceof Edge) {
          Node[] nodes = getModel().getSelectedStates();
          getModel().setFocus(nodes[0], nodes[1], (Edge) out.getSelected());
        }
      } else if (addOutEdges) {
        addOutEdges = false;
      }
    }
  });
  left.add(out);
  left.add(new JLabel("Internal:"));
  internal = new JTextField();
  internal.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
      if (node != null && node instanceof State) {
        if (getModel().getStateChart().getModelSource().
            equals("Esterel Studio")) {
          getModel().setStatus("Internal transitions are not supported by "
                               + "EsterelStudio.");
          internal.setText("");
        } else {
          ((State) node).getInternalTransition().setLabel(
            new StringLabel(e.getActionCommand().
                            trim().replaceAll("\\\\n",
                                              "\n")));
          getModel().setStatus("done.");
          updateChart();
        }
        Browser.requestKeyEdit();
      }
    }
  });

  left.add(internal);
  left.add(new JLabel("Children:"));
  childs = new PlusMinusComboBox(new ArrayList(), false, true);
  childs.addItemListener(new ItemListener() {
    public void itemStateChanged(final ItemEvent e) {
      if (e.getStateChange() == ItemEvent.SELECTED
          && !addChildNodes) {
        if (childs.getSelected() != null
            && childs.getSelected() instanceof Node) {
          Node[] nodes = getModel().getSelectedStates();
          getModel().setFocus((Node) childs.getSelected(), nodes[1],
                              getModel().getSelectedTransition());
        }
      } else if (addChildNodes) {
        addChildNodes = false;
      }
    }
  });
  left.add(childs);

  JPanel right = new JPanel();
  right.setOpaque(false);
  GridLayout grid3 = new GridLayout(6, 2);
  right.setLayout(grid3);
  right.add(new JLabel("Variables:"));
  vars = new PlusMinusComboBox(new ArrayList(), true, false);
  vars.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && vars.getSelected() != null
          && node instanceof State) {
```

```
        Variable v = (Variable) vars.getSelected();
        vars.removeItem(v);
        ((State) node).removeVariable(v);
        updateChart();
      }
    }
  });
  vars.addPlusButtonListener(new ActionListener() {
    private int vn = 1;
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Variable v = new IntegerVariable("v" + vn);
        vn++;
        vars.addItem(v);
        ((State) node).addVariable(v);
        updateChart();
      }
    }
  });
  vars.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        try {
          Collection coll = DeclarationParser.parseVariables(decl);
          if (old != -1) {
            ((State) node).removeVariable((Variable) cb.getItemAt(old));
            vars.removeItem(cb.getItemAt(old));
          }
          vars.addItems(coll);
          ((State) node).addVariables(coll);
          updateChart();
        } catch (DeclarationException ex) {
          getModel().sendExceptionMessage(ex);
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });
  right.add(vars);
  right.add(new JLabel("Events:"));
  events = new PlusMinusComboBox(new ArrayList(), true, false);
  events.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && events.getSelected() != null
          && node instanceof State) {
        Event ev = (Event) events.getSelected();
        events.removeItem(ev);
        ((State) node).removeLocalEvent(ev);
        updateChart();
      }
    }
  });
  events.addPlusButtonListener(new ActionListener() {
    private int en = 1;
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Event ev = new Signal("e" + en);
        en++;
        events.addItem(ev);
        ((State) node).addLocalEvent(ev);
        updateChart();
      }
    }
  });
  events.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        try {
          Collection coll = DeclarationParser.parseEvents(decl);
          if (old != -1) {
            ((State) node).removeLocalEvent((Event) cb.getItemAt(old));
            events.removeItem(cb.getItemAt(old));
          }
          events.addItems(coll);
          ((State) node).addLocalEvents(coll);
          updateChart();
        } catch (DeclarationException ex) {
          getModel().sendExceptionMessage(ex);
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });
  right.add(events);
  right.add(new JLabel("Entry Action:"));
  entry = new PlusMinusComboBox(new ArrayList(), true, false);
  entry.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && entry.getSelected() != null
          && node instanceof State) {
        Action a = (Action) entry.getSelected();
        entry.removeItem(a);
        ((State) node).getEntry().getActionsAsCollection().remove(a);
        updateChart();
      }
    }
  });
  entry.addPlusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Action a;
        if (((State) node).getLocalEvents().size() > 0) {
          a = new GenerateEvent((Event) ((State) node).
                                getLocalEvents().get(0));
        } else {
          Event ev = new Signal("aEvent");
          ((State) node).addLocalEvent(ev);
          a = new GenerateEvent(ev);
```

```
        }
        entry.addItem(a);
        ((State) node).getEntry().addAction(a);
        updateChart();
      }
    }
  });
  entry.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        if (decl.trim().length() > 0) {
          try {
            CompoundLabelParser.setStateChart(getModel().getStateChart());
            CompoundLabelParser.setCompositeState(node.getParent());
            Action a = CompoundLabelParser.parseAction(decl);
            if (old != -1) {
              ((State) node).getEntry().getActionsAsCollection().remove(
                cb.getItemAt(old));
              entry.removeItem(cb.getItemAt(old));
            }
            entry.addItem(a);
            ((State) node).getEntry().addAction(a);
            updateChart();
          } catch (CompoundLabelException ex) {
            getModel().sendExceptionMessage(ex);
          }
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });
  right.add(entry);
  right.add(new JLabel("Do Action:"));
  doActivity = new PlusMinusComboBox(new ArrayList(), true, false);
  doActivity.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && doActivity.getSelected() != null
          && node instanceof State) {
        Action a = (Action) doActivity.getSelected();
        doActivity.removeItem(a);
        ((State) node).getDoActivity().getActionsAsCollection().remove(a);
        updateChart();
      }
    }
  });
  doActivity.addPlusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Action a;
        if (((State) node).getLocalEvents().size() > 0) {
          a = new GenerateEvent((Event) ((State) node).
                                getLocalEvents().get(0));
        } else {
          Event ev = new Signal("aEvent");
          ((State) node).addLocalEvent(ev);
          a = new GenerateEvent(ev);
        }
        doActivity.addItem(a);
        ((State) node).getDoActivity().addAction(a);
        updateChart();
      }
    }
  });
  doActivity.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        if (decl.trim().length() > 0) {
          try {
            CompoundLabelParser.setStateChart(getModel().getStateChart());
            CompoundLabelParser.setCompositeState(node.getParent());
            Action a = CompoundLabelParser.parseAction(decl);
            if (old != -1) {
              ((State) node).getDoActivity().getActionsAsCollection().remove(
                cb.getItemAt(old));
              doActivity.removeItem(cb.getItemAt(old));
            }
            doActivity.addItem(a);
            ((State) node).getDoActivity().addAction(a);
            updateChart();
          } catch (CompoundLabelException ex) {
            getModel().sendExceptionMessage(ex);
          }
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });
  right.add(doActivity);
  right.add(new JLabel("Exit Action:"));
  exit = new PlusMinusComboBox(new ArrayList(), true, false);
  exit.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && exit.getSelected() != null
          && node instanceof State) {
        Action a = (Action) exit.getSelected();
        exit.removeItem(a);
        ((State) node).getExit().getActionsAsCollection().remove(a);
        updateChart();
      }
    }
  });
  exit.addPlusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Action a;
        if (((State) node).getLocalEvents().size() > 0) {
          a = new GenerateEvent((Event) ((State) node).
                                getLocalEvents().get(0));
```

```
        } else {
          Event ev = new Signal("aEvent");
          ((State) node).addLocalEvent(ev);
          a = new GenerateEvent(ev);
        }
        exit.addItem(a);
        ((State) node).getExit().addAction(a);
        updateChart();
      }
    }
  });
  exit.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        if (decl.trim().length() > 0) {
          try {
            CompoundLabelParser.setStateChart(getModel().getStateChart());
            CompoundLabelParser.setCompositeState(node.getParent());
            Action a = CompoundLabelParser.parseAction(decl);
            if (old != -1) {
              ((State) node).getExit().getActionsAsCollection().remove(
                cb.getItemAt(old));
              exit.removeItem(cb.getItemAt(old));
            }
            exit.addItem(a);
            ((State) node).getExit().addAction(a);
            updateChart();
          } catch (CompoundLabelException ex) {
            getModel().sendExceptionMessage(ex);
          }
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });

  right.add(exit);
  right.add(new JLabel("Bind Action:"));
  bind = new PlusMinusComboBox(new ArrayList(), true, false);
  bind.addMinusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && bind.getSelected() != null
          && node instanceof State) {
        Action a = (Action) bind.getSelected();
        bind.removeItem(a);
        ((State) node).getBindAction().getActionsAsCollection().remove(a);
        updateChart();
      }
    }
  });
  bind.addPlusButtonListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      if (node != null && node instanceof State) {
        Action a;
        if (((State) node).getLocalEvents().size() > 0) {
          a = new GenerateEvent((Event) ((State) node).
                                getLocalEvents().get(0));
        } else {
          Event ev = new Signal("aEvent");
          ((State) node).addLocalEvent(ev);
          a = new GenerateEvent(ev);
        }
        bind.addItem(a);
        ((State) node).getBindAction().addAction(a);
        updateChart();
      }
    }
  });
  bind.addActionListener(new ActionListener() {
    private int old = -1;
    public void actionPerformed(ActionEvent e) {
      JComboBox cb = (JComboBox) e.getSource();
      if (e.getActionCommand().equals("comboBoxEdited")
          && node != null && node instanceof State
          && cb.getSelectedItem() instanceof String) {
        String decl = (String) cb.getSelectedItem();
        if (decl.trim().length() > 0) {
          try {
            CompoundLabelParser.setStateChart(getModel().getStateChart());
            CompoundLabelParser.setCompositeState(node.getParent());
            Action a = CompoundLabelParser.parseAction(decl);
            if (old != -1) {
              ((State) node).getBindAction().getActionsAsCollection().remove(
                cb.getItemAt(old));
              bind.removeItem(cb.getItemAt(old));
            }
            bind.addItem(a);
            ((State) node).getBindAction().addAction(a);
            updateChart();
          } catch (CompoundLabelException ex) {
            getModel().sendExceptionMessage(ex);
          }
        }
      } else if (cb.getSelectedIndex() != -1) {
        old = cb.getSelectedIndex();
      }
    }
  });
  right.add(bind);

  panel.add(left);
  panel.add(right);
}

/**
 * @return JComponent
 */
public final JComponent getComponent() {
  return panel;

}

/**
 * Updates the chart.
```

```
     */
    private void updateChart() {
      Hashtable changed = new Hashtable(1);
      if (node instanceof CompositeState) {
        ModelHelper.addSubTree((CompositeState) node, changed);
      }
      changed.put(node.getID(), node);
      getModel().setChangedObjects(null,
                                   null,
                                   changed.values(),
                                   this.getClass().getName());
      KielLayouter layouter = Browser.getKielFrame().
                              getKielLayouterByName(
                                Browser.getKielFrame().
                                getDefaultLayouterName());
      try {
        layouter.setStatechart(getModel().getStateChart());
        layouter.layoutView(layouter.getStaticView());
        getModel().setLayouter(layouter, Browser.getKielFrame().
                               getDefaultLayouterName());
        getModel().showStaticView();
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }

    /**
     * Updates the chart.
     */
    private void replaceNode(Node n) {
      ArrayList removed = new ArrayList(1);
      removed.add(node);
      ArrayList added = new ArrayList(1);
      added.add(n);
      getModel().setChangedObjects(removed,
                                   added,
                                   null,
                                   this.getClass().getName());
      KielLayouter layouter = Browser.getKielFrame().
                              getKielLayouterByName(
                              Browser.getKielFrame().
                              getDefaultLayouterName());
      try {
        layouter.setStatechart(getModel().getStateChart());
        layouter.layoutView(layouter.getStaticView());
        getModel().setLayouter(layouter, Browser.getKielFrame().
                               getDefaultLayouterName());
        getModel().showStaticView();
        node = n;
        getModel().setFocus(node,
          getModel().getFocus().getNodes()[1],
          getModel().getFocus().getEdge());
        getModel().setStatus("Operation done");
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }


    /**
     * List of all Node types.
     */
    private String[] stateTypes = {
                                  "Simple", "ANDState", "ORState", "Region",
                                  "FinalState", "FinalANDState", "FinalORState",
                                  "Initial", "Choice", "DynamicChoice", "History",
                                  "DeepHistory"};

    /**
     * @param n Node
     * @return int type number of Node n
     */
    private static int getIndex(final Node n) {
      int index = 0;
      if (n.getClass() == ANDState.class) {
        index = 1;
      } else if (n.getClass() == ORState.class) {
        index = 2;
      } else if (n.getClass() == Region.class) {
        index = 3;
      } else if (n.getClass() == FinalState.class
                 || n.getClass() == FinalSimpleState.class) {
        index = 4;
      } else if (n.getClass() == FinalANDState.class) {
        index = 5;
      } else if (n.getClass() == FinalORState.class) {
        index = 6;
      } else if (n.getClass() == InitialState.class) {
        index = 7;
      } else if (n.getClass() == Choice.class) {
        index = 8;
      } else if (n.getClass() == DynamicChoice.class) {
        index = 9;
      } else if (n.getClass() == History.class) {
        index = 10;
      } else if (n.getClass() == DeepHistory.class) {
        index = 11;
      }
      return index;
    }

    /**
     * @param o Observable
     * @param arg Object
     */
    public final void update(final Observable o, final Object arg) {
      Node nn = null;
      if (arg.equals(BrowserModel.FOCUS_CHANGED)) {
        if (getModel().getFocus().hasNode1Changed()
            && getModel().getFocus().getNodes()[0] != null) {
          nn = getModel().getFocus().getNodes()[0];
        } else if (getModel().getFocus().hasNode2Changed()
                   && getModel().getFocus().getNodes()[1] != null) {
          nn = getModel().getFocus().getNodes()[1];
        }
      } else if (arg.equals(BrowserModel.UPDATE_CHART) && node != null) {
        Iterator iter = getModel().getStateChart().getAllObjects().iterator();
        while (iter.hasNext()) {
          GraphicalObject go = (GraphicalObject) iter.next();
          if (go.getID().equals(node.getID()) && go instanceof Node) {
            nn = (Node) go;
```

```
      break;
    }
  }
  if (nn == null) {
    clearPanel();
    return;
  }
} 960
if (nn != null) {
  node = nn;
  refreshNode();
}

/**
 * Refreshes this view.
 */
private void clearPanel() {
  name.setText("");
  id.setText("");
  out.clearItems();
  in.clearItems();
  childs.clearItems();
  events.clearItems();
  vars.clearItems();
  entry.clearItems();
  doActivity.clearItems();
  exit.clearItems();
  bind.clearItems();
  node = null;
}

/**
 * Refreshes this view.
 */
private void refreshNode() {
  name.setText("");
  if (node.getName() != null && !node.getName().equals("")) {
    name.setText(node.getName());
  }
  id.setText("");
  if (node.getID() != null && !node.getID().equals("")) {
    id.setText(node.getID());
  }
  type.setSelectedIndex(getIndex(node));
  out.clearItems();
  addOutEdges = true;
  if (node.getOutgoingTransitions() != null
      && node.getOutgoingTransitions().size() > 0) {
    out.addItems(node.getOutgoingTransitions());
  }
  in.clearItems();
  addInEdges = true;
  if (node.getIncomingTransitions() != null
      && node.getIncomingTransitions().size() > 0) {
    in.addItems(node.getIncomingTransitions());
  }
  childs.clearItems();
  events.clearItems();
  vars.clearItems();
  entry.clearItems();
  doActivity.clearItems();
  exit.clearItems();
  bind.clearItems();
  if (node instanceof State) {
    if (((State) node).getLocalEvents() != null
        && ((State) node).getLocalEvents().size() > 0) {
      events.addItems(((State) node).getLocalEvents());
    }
    if (((State) node).getVariables() != null
        && ((State) node).getVariables().size() > 0) {
      vars.addItems(((State) node).getVariables());
    }
    if (((State) node).getInternalTransition() != null
        && ((State) node).getInternalTransition().getLabel() != null
        && !((State) node).getInternalTransition().getLabel().toString().
            equals("")) {
      internal.setText(((State) node).getInternalTransition().getLabel().
          toString());
    }
    if (((State) node).getEntry() != null
        && !((State) node).getEntry().toString().equals("")) {
      entry.addItems(((State) node).getEntry().getActionsAsCollection());
    }
    if (((State) node).getExit() != null
        && !((State) node).getExit().toString().equals("")) {
      exit.addItems(((State) node).getExit().getActionsAsCollection());
    }
    if (((State) node).getDoActivity() != null
        && !((State) node).getDoActivity().toString().equals("")) {
      doActivity.addItems(((State) node).getDoActivity().
          getActionsAsCollection());
    }
    if (((State) node).getBindAction() != null
        && !((State) node).getBindAction().toString().equals("")) {
      bind.addItems(((State) node).getBindAction().
          getActionsAsCollection());
    }
    addChildNodes = true;
    if (node instanceof CompositeState
        && ((CompositeState) node).getSubnodes().size() > 0) {
      childs.addItems(((CompositeState) node).getSubnodes());
    }
  }
}
```

F.5.9. PlusMinusComboBox

```
package kiel.browser.view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Paint;
import java.awt.event.ActionListener;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

import javax.swing.DefaultListCellRenderer;
import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JSplitPane;
import javax.swing.plaf.metal.MetalLookAndFeel;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.browser.BrowserProperties;
import java.awt.event.ItemEvent;
import javax.swing.event.PopupMenuListener;
import javax.swing.event.PopupMenuEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import javax.swing.AbstractAction;
import java.awt.event.ActionEvent;
import javax.swing.KeyStroke;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Box with a plus and a minus button.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.9 £ last modified £Date: 2006/05/21 20:24:02 £
 */
public class PlusMinusComboBox extends JPanel {
  /**
   * List of elements.
   */
  private ArrayList objs;
  /**
   * Box.
   */
  private JComboBox combo;
  /**
   * Button.
   */
  private JButton plus;
  /**
   * Button.
   */
  private JButton minus;

  /**
   * @param list Collection
   * @param editable flag if combobox should be editable
   * @param disable flag for disabling the plus-minus buttons
   */
  PlusMinusComboBox(final Collection list,
                    final boolean editable,
                    final boolean disable) {
    objs = new ArrayList();
    objs.addAll(list);
    combo = new JComboBox(objs.toArray());
    combo.setOpaque(false);
    combo.setForeground(Color.BLACK);
    combo.setBackground(MetalLookAndFeel.getControl());
    combo.setEditable(editable);

    combo.setRenderer(new DefaultListCellRenderer() {
      /**
       * Icon for variable declaration.
       */
      private Icon varDeclIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/var.gif");
      /**
       * Icon for local signal declaration.
       */
      private Icon signalDeclIcon = ResourceLoader.createImageIcon(
        "kiel/browser/images/local.gif");

      private Icon edgeIcon = ResourceLoader.createImageIcon(
```

```
    "kiel/browser/images/trans.gif");
  /**
   * sample andicon.
   */
  private Icon andIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/and.gif");
  /**
   * Icon for orstate.
   */
  private Icon orIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/or.gif");
  /**
   * Icon for simple state.
   */
  private Icon simpleIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/simple.gif");
  /**
   * Icon for inital state.
   */
  private Icon initialIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/init.gif");
  /**
   * Icon for choice state.
   */
  private Icon choiceIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/choice.gif");
  /**
   * Icon for region state.
   */
  private Icon regionIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/region.gif");
  /**
   * Icon for suspend state.
   */
  private Icon suspendIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/suspend.gif");
  /**
   * Icon for final state.
   */
  private Icon finalIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/final.gif");
  /**
   * Icon for history state.
   */
  private Icon historyIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/history.gif");

  public Component getListCellRendererComponent(
    final JList list,
    final Object value,
    final int index,
    final boolean isSelected,
    final boolean cellHasFocus) {
    JLabel label = new JLabel();
    label.setOpaque(true);
    if (value != null) {
      if (value instanceof Event) {
        label.setText(((Event) value).toDeclaration());
        label.setIcon(signalDeclIcon);
        label.setBackground(isSelected ? Color.red : Color.white);
        label.setForeground(isSelected ? Color.white : Color.black);
      } else if (value instanceof Variable) {
        label.setText(((Variable) value).toDeclaration());
        label.setIcon(varDeclIcon);
        label.setBackground(isSelected ? Color.red : Color.white);
        label.setForeground(isSelected ? Color.white : Color.black);
      } else if (value instanceof Edge) {
        label.setText(value.toString());
        label.setIcon(edgeIcon);
        label.setBackground(
          isSelected ? BrowserProperties.getEdgeMarkColor() : Color.white);
      } else if (value instanceof Node) {
        label.setText(value.toString());
        label.setBackground(
          isSelected ? BrowserProperties.getNodeMarkColor() : Color.white);
        if (value.getClass() == ANDState.class) {
          label.setIcon(andIcon);
        } else if (value.getClass() == ORState.class) {
          label.setIcon(orIcon);
        } else if (value.getClass() == Region.class) {
          label.setIcon(regionIcon);
          label.setForeground(Color.gray);
        } else if (value.getClass() == FinalState.class
                   || value.getClass() == FinalSimpleState.class) {
          label.setIcon(finalIcon);
        } else if (value.getClass() == FinalANDState.class) {
          label.setIcon(finalIcon);
        } else if (value.getClass() == FinalORState.class) {
          label.setIcon(finalIcon);
        } else if (value.getClass() == InitialState.class) {
          label.setIcon(initialIcon);
        } else if (value.getClass() == Choice.class) {
          label.setIcon(choiceIcon);
        } else if (value.getClass() == DynamicChoice.class) {
          label.setIcon(choiceIcon);
        } else if (value.getClass() == History.class) {
          label.setIcon(historyIcon);
        } else if (value.getClass() == DeepHistory.class) {
          label.setIcon(historyIcon);
        }
      } else {
        label.setText(value.toString());
        label.setBackground(isSelected ? Color.red : Color.white);
        label.setForeground(isSelected ? Color.white : Color.black);
      }

    }
    return label;
  }
});
combo.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT).
  put(KeyStroke.getKeyStroke(KeyEvent.VK_PLUS, KeyEvent.CTRL_MASK),
      "pmcbplus");
combo.getActionMap().put("pmcbplus", new AbstractAction() {
  public void actionPerformed(final ActionEvent e) {
    plus.doClick();
  }
});
combo.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT).
  put(KeyStroke.getKeyStroke(KeyEvent.VK_MINUS, KeyEvent.CTRL_MASK),
```

```
            "pmcbminus");
    combo.getActionMap().put("pmcbminus", new AbstractAction() {
      public void actionPerformed(final ActionEvent e) {
        minus.doClick();
      }
    });

    JPanel pane1 = PaintUtils.createJPanel(null, true);
    plus = new JButton();
    plus.setOpaque(false);
    plus.setFocusable(false);
    plus.setText("+");
    plus.setEnabled(!disable);
    minus = new JButton();
    minus.setOpaque(false);
    minus.setFocusable(false);
    minus.setText("-");
    minus.setEnabled(!disable);
    pane1.add(plus);
    pane1.add(minus);
    pane1.setLayout(new GridLayout(0, 2));
    JSplitPane pane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, pane1, combo);
    pane.setBorder(null);
    pane.setDividerSize(0);
    final int dividerLoc = 90;
    pane.setDividerLocation(dividerLoc);
    pane.setEnabled(false);
    this.setLayout(new BorderLayout());
    this.add(pane, BorderLayout.CENTER);
  }

  /**
   * @param g Graphics
   */
  protected final void paintComponent(final Graphics g) {
    boolean topToBottom = true;
    Color light = MetalLookAndFeel.getControl();
    Color dark = MetalLookAndFeel.getControlInfo();
    JComponent c = this;

    Paint oldPaint = ((Graphics2D) g).getPaint();

    if (topToBottom) {
      ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                                                  light, 0, c.getSize().height,
                                                  dark, true));
    } else {
      ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                                                  dark, c.getSize().width, 0,
                                                  light, true));
    }
    ((Graphics2D) g).fillRect(0, 0, c.getSize().width, c.getSize().height);
    ((Graphics2D) g).setPaint(oldPaint);
  }

  /**
   * @param listen ItemListener
   */
  public final void addItemListener(final ItemListener listen) {
    combo.addItemListener(listen);
  }

  /**
   * @param listen ItemListener
   */
  public final void addPlusButtonListener(final ActionListener listen) {
    plus.addActionListener(listen);
  }

  /**
   *
   * @param listen ItemListener
   */
  public final void addMinusButtonListener(final ActionListener listen) {
    minus.addActionListener(listen);
  }

  /**
   * Adds list of items.
   * @param items Collection
   */
  public final void addItems(final Collection items) {
    objs.addAll(items);
    Iterator iter = items.iterator();
    while (iter.hasNext()) {
      combo.addItem(iter.next());
    }
  }

  /**
   * remove all items.
   */
  public final void clearItems() {
    objs.clear();
    combo.removeAllItems();
  }

  /**
   * @return Object selected
   */
  public final Object getSelected() {
    return combo.getSelectedItem();
  }

  /**
   *
   * @param listen ActionListener
   */
  public final void addActionListener(final ActionListener listen) {
    combo.addActionListener(listen);
  }

  /**
   * addItem.
   *
   * @param obj Object
   */
  public final void addItem(final Object obj) {
    objs.add(obj);
    combo.addItem(obj);
```

```
}
    /**
     * remove an Item.
     *
     * @param obj Object
     */
    public final void removeItem(final Object obj) {
        objs.remove(obj);
        combo.removeItem(obj);
    }
}
```

F.5.10. ResourceLoader

```
// £Id: ResourceLoader.java,v 1.6 2005/05/12 15:12:49 miwi Exp £
package kiel.browser.view;

import javax.swing.ImageIcon;

/**
 * <p>Description: A utility class for loading resources.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.6 £ last modified £Date: 2005/05/12 15:12:49 £
 */

public final class ResourceLoader {

  /**
   *  Default constructor is private.
   */
  private ResourceLoader() {
  }

  /**
   * @param path path to resource
   * @return an ImageIcon, or null if the path was invalid.
   */
  public static ImageIcon createImageIcon(final String path) {
    java.net.URL imgURL = ResourceLoader.class.getClassLoader().getResource(
        path);
    if (imgURL != null) {
      return new ImageIcon(imgURL);
    } else {
      System.err.println("Couldn't find file: " + path);
      return null;
    }
  }
}
```

F.5.11. SignalTable

```
// £Id: SignalTable.java,v 1.29 2006/02/08 07:07:49 miwi Exp £
package kiel.browser.view;

import java.awt.Color;
import java.awt.Component;
import java.awt.FontMetrics;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

import javax.swing.DefaultCellEditor;
import javax.swing.JCheckBox;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;
import javax.swing.table.TableColumnModel;

import kiel.dataStructure.Constant;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.browser.BrowserProperties;
import kiel.browser.model.ExpInformation;
import kiel.browser.model.SignalTableModel;

//import kiel.dataStructure.eventexp.IntegerSignal;
//import kiel.dataStructure.intexp.IntegerExpression;

/**
 * <p>Description: Gui stuff for the various events/variables tables in the
 * browser.</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Uni Kiel</p>
 *
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.29 £ last modified £Date: 2006/02/08 07:07:49 £
 *
 * @todo extend this class so it is an browserGui class that can react
 *       on micro/macro Steps
 *
 *
 */
public class SignalTable
    implements TableModelListener, ComponentListener {

  /**
   *  Swing table shown in this view.
   */
  private JTable table;
  /**
   *  the model needed for resizing.
   */
  private SignalTableModel model;
  /**
   *  Scrollpane for the table.
   */
  private JScrollPane pane;
  /**
   *  Offset for Icon and checkbox.
   */
  private static final int SIGNAL_CELL_OFFSET = 50;
  /**
   *  Offset for value string.
   */
  private static final int VALUE_CELL_OFFSET = 5;
  /**
   *  Offset for table header width.
   */
  private static final int HEADER_WIDTH_OFFSET = 5;

  /**
   * @param m Signal Table is feed with this model
   */
  public SignalTable(final SignalTableModel m) {
    table = new JTable(m);
    m.addTableModelListener(this);
    model = m;
    DefaultTableCellRenderer renderer = new DefaultTableCellRenderer() {
      /**
       *
       * @param table Table to render
       * @param value value to render
       * @param isSelected is value selected
       * @param hasFocus has cell the focus
       * @param row table row
       * @param column table column
       * @return rendered cell component
       */
      public Component getTableCellRendererComponent(final JTable mytable,
          final Object value,
          final boolean isSelected,
          final boolean hasFocus,
          final int row,
          final int column) {
        return super.getTableCellRendererComponent(mytable, value,
            false, false, row, column);
      }
    };
    renderer.setOpaque(true);
    table.setDefaultRenderer(ExpInformation.class,
                             new ExpressionCellRenderer());
    table.setDefaultRenderer(String.class,
                             renderer);
    table.setDefaultEditor(ExpInformation.class,
                           new DefaultCellEditor(new JCheckBox()));
    pane = new JScrollPane(table);
    pane.addComponentListener(this);
```

```
    table.addComponentListener(this);
  }

  /**
   * @return Swing component for this table
   */
  public final JComponent getComponent() {
    return pane;
  }

  /**
   *
   * @param tableModelEvent .
   */
  public final void tableChanged(final TableModelEvent tableModelEvent) {
    if (tableModelEvent.getType() == TableModelEvent.UPDATE
        && tableModelEvent.getColumn() == 0) {
      initColumnSizes();
    }
  }

  /**
   * This method picks good column sizes.
   * If all column heads are wider than the column's cells'
   * contents, then you can just use column.sizeWidthToFit().
   */
  private void initColumnSizes() {
    TableColumn column = null;
    Component comp = null;
    int headerWidth = 0;
    int cellWidth = 0;
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

    for (int i = 0; i < model.getColumnCount(); i++) {
      column = table.getColumnModel().getColumn(i);

      headerWidth = getHeaderSize(i);
      cellWidth = getMaxCellSize(i);

      column.setPreferredWidth(Math.max(headerWidth, cellWidth));
    }
  }

  /**
   * This method should be called to set the column at pColumn index to a width
   * of pWidth.
   *
   * @param pColumn int column to set width for
   * @param pWidth int width to set
   */
  public final void setColumnWidth(final int pColumn, final int pWidth) {
    //Get the column model.
    TableColumnModel colModel = table.getColumnModel();
    //Get the column at index pColumn, and set its preferred width.
    colModel.getColumn(pColumn).setPreferredWidth(pWidth);
  }

  /**
   * Sets the header and column size as per the header text.
   *
   * @param pColumn int column to compute header size for
   * @return int pixel size of header
   */
  private int getHeaderSize(final int pColumn) {
    //Get the column name of the given column.
    String value = table.getColumnName(pColumn);
    //Calculate the width required for the column.
    FontMetrics metrics = table.getGraphics().getFontMetrics();
    return metrics.stringWidth(value)
        + (2 * table.getColumnModel().getColumnMargin()) + HEADER_WIDTH_OFFSET;
  }

  /**
   * Sets the header and column size as per the header text.
   *
   * @param pColumn int column to compute size for
   * @return int maximum cell size in pixel
   */
  private int getMaxCellSize(final int pColumn) {
    FontMetrics metrics = table.getGraphics().getFontMetrics();
    int max = 0;
    int act = 0;

    for (int i = 0; i < model.getRowCount(); i++) {
      Object obj = model.getValueAt(i, pColumn);
      if (obj instanceof ExpInformation) {
        ExpInformation info = (ExpInformation) obj;
        act = metrics.stringWidth(info.toString()) + SIGNAL_CELL_OFFSET;
        if (act > max) {
          max = act;
        }
      } else if (obj instanceof String) {
        act = metrics.stringWidth((String) obj) + VALUE_CELL_OFFSET;
        if (act > max) {
          max = act;
        }
      }
    }
    return max + (2 * table.getColumnModel().getColumnMargin());
  }

  /**
   *
   * @param e ComponentEvent
   */
  public final void componentHidden(final ComponentEvent e) {
  }

  /**
   *
   * @param e ComponentEvent
   */
  public final void componentMoved(final ComponentEvent e) {
  }

  /**
   *
   * @param e ComponentEvent
   */
  public final void componentShown(final ComponentEvent e) {
```

```
    }

    /**
     * @see #autoModelResize
     * @param e ComponentEvent
     */
    public final void componentResized(final ComponentEvent e) {
      autoModelResize();
    }


    /**
     *  Resizes the model so smaller/no scrollbars are needed.
     */
    public final void autoModelResize() {
      int size = fitToHeight();
      if (size == -1) {
        size = fitToWidth();
      }
      if (size > 0 && size != model.getColumnCount() / 2) {
        model.resize(size);
      }
    }

    /**
     *
     * @return int
     */
    private int fitToWidth() {
      double size = pane.getWidth() / ((double) table.getWidth()
                          / ((double) table.getColumnCount() / 2));
      if (size > 1.0) {
        return (int) Math.floor(size);
      } else {
        return 1;
      }
    }

    /**
     *
     * @return int
     */
    private int fitToHeight() {
      int size = (int) Math.ceil(model.getExpInformations().size()
                    / ((double) pane.getHeight()
                       / ((double) table.getHeight()
                          / (double) table.getRowCount())));
      if (((double) table.getWidth() / ((double) table.getColumnCount() / 2))
            * size <= pane.getWidth())  {
        return size;
      }
      return -1;
    }
  }

  /**
   * <p>Description: Rendering class for events/variables.</p>
   * <p>Copyright: Copyright (c) 2004</p>
   * <p>Company: Uni Kiel</p>
   *
   * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
   */
  class ExpressionCellRenderer
      extends JCheckBoxMenuItem implements TableCellRenderer {

    /**
     * Calls just super constructor.
     */
    public ExpressionCellRenderer() {
      super();
    }

    /**
     *
     * @param table Table to render
     * @param value value to render
     * @param isSelected is value selected
     * @param hasFocus has cell the focus
     * @param row table row
     * @param column table column
     * @return rendered cell component
     */
    public Component getTableCellRendererComponent(final JTable table,
                                                   final Object value,
                                                   final boolean isSelected,
                                                   final boolean hasFocus,
                                                   final int row,
                                                   final int column) {
      if (value != null) {
        ExpInformation info = (ExpInformation) value;
        if (info.isEvent()) {
          Event e = info.getEvent();
          setBackground(Color.WHITE);
          setText(e.getName());
          setToolTipText("Declared as: " + e.toDeclaration());
          setSelected(info.isEmited());
          if (info.getType() == SignalTableModel.INPUT) {
            setIcon(ResourceLoader.createImageIcon(
                "kiel/browser/images/input.gif"));
            setForeground(BrowserProperties.getInputEventColor());
          } else if (info.getType() == SignalTableModel.OUTPUT) {
            setIcon(ResourceLoader.createImageIcon(
                "kiel/browser/images/output.gif"));
            setForeground(BrowserProperties.getOutputEventColor());
          } else {
            setIcon(ResourceLoader.createImageIcon(
                "kiel/browser/images/local.gif"));
            setForeground(BrowserProperties.getLocalEventColor());
          }
        } else {
          Variable v = info.getVar();
          if (v != null) {
            setForeground(BrowserProperties.getVariablesColor());
            setBackground(Color.WHITE);
            setText(v.getName());
            setToolTipText("Declared as: " + v.toDeclaration());
            setSelected(false);
            setIcon(
              ResourceLoader.createImageIcon("kiel/browser/images/var.gif"));
          } else {
```

```
            }
          }
          return this;
        }
        return null;
      }
    }

    Constant c = info.getConstant();
    setForeground(BrowserProperties.getVariablesColor());
    setBackground(Color.WHITE);
    setText(c.getName());
    setToolTipText("Declarated as: " + c.toDeclaration());
    setSelected(false);
    setIcon(
      ResourceLoader.createImageIcon("kiel/browser/images/constant.gif"));
  }
}
```

F.5.12. TextEditor

```
package kiel.browser.view;

//import kiel.fileInterface.kit.KitGenerator;
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Rectangle;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JComponent;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextPane;
import javax.swing.SwingUtilities;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;

import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ParserThread;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;
import kiel.util.languages.StatechartLanguage;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Creates the language tabs with textpanes inside.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.13 £ last modified £Date: 2007/02/23 01:51:15 £
 *
 * @todo create strategies for updateing statechart --> thinking
 */
public class TextEditor extends BrowserGUI {
  /**
   * Tabs for languages.
   */
  private JTabbedPane tabs;

  /**
   * @param m BrowserModel
   */
  public TextEditor(final BrowserModel m) {
    super(m);
    tabs = new JTabbedPane();
  }

  /**
   *
   * @return Swing component for this class
   */
  public final JComponent getComponent() {
    return tabs;
  }

  /**
   * @return null at the moment
   */
  public final Document getDocument() {
    // return pane.getStyledDocument();
    return null;
  }

  /**
   * This method is called whenever the observed object is changed.
   *
   * @param o the observable object.
   * @param arg an argument passed to the <code>notifyObservers</code> method.
   */
  public final void update(final Observable o, final Object arg) {
    if (arg.equals(BrowserModel.NEW_CHART)) {
      Iterator iter = getModel().getLanguages().iterator();
      while (iter.hasNext()) {
        StatechartLanguage lang = (StatechartLanguage) iter.next();
        lang.setStatechart(getModel().getStateChart(),
                           getModel().getCurrentFile());
        lang.getLineNumber().clearIcons();
      }
    } else if (arg.equals(BrowserModel.UPDATE_CHART)
               && !BrowserModel.UPDATE_CHART.getParameter().equals(
                     ParserThread.class.getName())) {
//      System.out.println("Updating Statechart: "
//                         + BrowserModel.UPDATE_CHART.getParameter());
      Iterator iter = getModel().getLanguages().iterator();
      while (iter.hasNext()) {
        StatechartLanguage lang = (StatechartLanguage) iter.next();
        lang.setStatechart(getModel().getStateChart(), null);
        lang.getLineNumber().clearIcons();
        lang.restoreCursor();
      }
    } else if (arg.equals(BrowserModel.NEW_LANGUAGE)) {
      tabs.removeAll();
      Iterator iter = getModel().getLanguages().iterator();
      int i = 0;
      while (iter.hasNext()) {
        StatechartLanguage lang = (StatechartLanguage) iter.next();
        JTextPane textEditor = new JTextPane();
        textEditor.setFont(new Font("DialogInput", Font.PLAIN,
                                    BrowserProperties.getKitEditorFontSize()));
        XMLPreferences.getInstance("browser").addObserver(
```

```
                new TextPaneObserver(textEditor));
        LineNumber line = new LineNumber(textEditor);
        lang.setLineNumber(line);
        lang.setTextPane(textEditor);
        tabs.insertTab(lang.getName(), null, createNewEditor(textEditor, line),
                       lang.getTooltip(), i);
        lang.setUpdateThread(new ParserThread());
        i++;
      }
    }
  }

  /**
   * Requests focus on selected textpane.
   */
  public final void requestFocus() {
    if (tabs.getSelectedComponent() != null) {
      JTextPane text = searchTextPane((JComponent) tabs.getSelectedComponent());
      text.requestFocusInWindow();
    }
  }

  /**
   * @return JTextPane selected textpane from tab.
   */
  public JTextPane getSelectedTextPane() {
       return searchTextPane((JComponent) tabs.getSelectedComponent());
  }

  /**
   *
   * @param comp JComponent
   * @return JTextPane
   */
  private JTextPane searchTextPane(final JComponent comp) {
    for (int i = 0; i < comp.getComponentCount(); i++) {
      if (comp.getComponent(i) instanceof JComponent) {
        if (comp.getComponent(i).getClass() == JTextPane.class) {
          return (JTextPane) comp.getComponent(i);
        } else {
          return searchTextPane((JComponent) comp.getComponent(i));
        }
      }
    }
    return null;
  }


  /**
   * @param pane JTextPane
   * @param lineNumber LineNumber
   * @return JComponent
   */
  private JComponent createNewEditor(final JTextPane pane,
                                     final LineNumber lineNumber) {

    JPanel noWrap = new JPanel(new BorderLayout());
    JScrollPane scroll = new JScrollPane(noWrap);

    pane.setText("no chart loaded");
    lineNumber.setFont(pane.getFont());
    scroll.setRowHeaderView(lineNumber);
    noWrap.add(pane);

    pane.addCaretListener(new CaretListener() {
      public void caretUpdate(final CaretEvent e) {
        displaySelectionInfo(e.getDot(), e.getMark());
      }

      //This method can be invoked from any thread.  It
      //invokes the setText and modelToView methods, which
      //must run in the event dispatching thread. We use
      //invokeLater to schedule the code for execution
      //in the event dispatching thread.
      protected void displaySelectionInfo(final int dot, final int mark) {
        SwingUtilities.invokeLater(new Runnable() {
          public void run() {
            if (dot == mark) { // no selection
              try {
                Rectangle caretCoords = pane.modelToView(dot);
                //Convert it to view coordinates.
                if (caretCoords != null) {
                  getModel().setKitStatus("P: " + dot
                                          + " L: "
                                          + ((caretCoords.y
                                              / lineNumber.getLineHeight())
                                             + 1));
                }
              } catch (BadLocationException ble) {
                getModel().setStatus("P: " + dot);
              }
            } else if (dot < mark) {
              getModel().setKitStatus("From: " + dot + " to " + mark);
            } else {
              getModel().setKitStatus("From: " + mark + " to " + dot);
            }
          }
        });
      }
    });
    return scroll;
  }

  /**
   * @return Returns the tabs.
   */
  public final JTabbedPane getTabs() {
    return tabs;
  }
}

class TextPaneObserver implements Observer {
    private JTextPane t;

    public TextPaneObserver(JTextPane t) {
        this.t = t;
    }

    public void update(Observable o, Object arg) {
        if (arg.equals(XMLPreferences.PREFERENCES_UPDATED)) {
```

```
            }
        }
        int newFontSize = BrowserProperties.getKitEditorFontSize();
        t.setFont(new Font("DialogInput", Font.PLAIN, newFontSize));
    }
```

F.5.13. View2ViewAnimator

```
package kiel.browser.view;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.Region;
import kiel.dataStructure.State;
import kiel.dataStructure.Transition;
import kiel.graphicalInformations.CompositeStateLayoutInformation;
import kiel.graphicalInformations.DelimiterLineLayoutInformation;
import kiel.graphicalInformations.EdgeLayoutInformation;
import kiel.graphicalInformations.LabelLayoutInformation;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelMessage;
import kiel.browser.view.rendering.CircleNode;
import kiel.browser.view.rendering.GroupNode;
import kiel.browser.view.rendering.LineNode;
import kiel.browser.view.rendering.PathNode;
import kiel.browser.view.rendering.RectNode;
import kiel.browser.view.rendering.TextNode;
import kiel.browser.view.rendering.Toolkit;
import kiel.util.EdgeWorker;
import kiel.util.NodeWorker;
import kiel.util.TreeHelper;
import kiel.util.WorkException;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.16 £ last modified £Date: 2007/02/06 18:23:22 £
 *
 */
public class View2ViewAnimator implements Observer, NodeWorker, EdgeWorker {
  /**
   *
   */
  private BrowserModel model;
  /**
   *
   */
  private Toolkit toolkit;
  /**
   *
   */
  private View actualView;
  /**
   *
   */
  private View oldView;
//  /**
//   *
//   */
//  private ArrayList animationElements;
  /**
   *
   * @param aModel BrowserModel
   * @param aToolkit a Toolkit used for drawing and animations
   */
  public View2ViewAnimator(final BrowserModel aModel,
                           final Toolkit aToolkit) {
    model = aModel;
    toolkit = aToolkit;
    actualView = aModel.getView();
    oldView = null;
//    animationElements = new ArrayList();
  }

  /**
   *
   * @param o Observable
   * @param arg Object
   */
  public final void update(final Observable o, final Object arg) {
    // Just react on messages if animaitons are active
    if (BrowserProperties.doAnimations()
        && arg.getClass() == ModelMessage.class) {
      ModelMessage message = (ModelMessage) arg;
      if (message.equals(BrowserModel.NEW_VIEW)) {
        oldView = actualView;
        actualView = model.getView();
        if (oldView != null && oldView != actualView && actualView != null) {
          // no chart change but view changes so animator has to run....
          log("Creating Animaitons : \t\t");
          animate();
          log("[done]");
          toolkit.startPlayback();
//          try {
//            log(toolkit.getDocument().getText(0,
//              toolkit.getDocument().getLength()));
//          } catch (BadLocationException ex) {
```

```
//            System.err.println("Kann Document nicht ausgeben: " + ex);
//          }
        }
      } else if (message.equals(BrowserModel.UPDATE_CHART)) {
        Iterator iter = model.getRemovedObjects().iterator();
        while (iter.hasNext()) {
          GraphicalObject obj = (GraphicalObject) iter.next();
          kiel.browser.view.rendering.Node n =
            toolkit.getElementById(BrowserCanvas.getGroupId(obj));
//          System.out.print("Trying to remove Object " + obj.toString() + " "
//                           + obj.getClass() + "\t");
          if (n != null) {
            ((GroupNode) n).animatedRemove(true);
//            if (toolkit.getElementById(BrowserCanvas.getGroupId(obj)) != null) {
//              System.out.println("[failed] could not be removed");
//            } else {
//              System.out.println("[okay]");
//            }
          }
//          else {
//            System.out.println("[failed] could not be found");
//          }
        }
        iter = model.getChangedObjects().iterator();
        while (iter.hasNext()) {
          GraphicalObject obj = (GraphicalObject) iter.next();
          kiel.browser.view.rendering.Node n =
            toolkit.getElementById(BrowserCanvas.getGroupId(obj));
//          System.out.print("Trying to remove changed Object " + obj.toString()
//                           + " " + obj.getClass() + "\t");
          if (n != null) {
            ((GroupNode) n).animatedRemove(true);
//            if (toolkit.getElementById(BrowserCanvas.getGroupId(obj)) != null) {
//              System.out.println("[failed] could not be removed");
//            } else {
//              System.out.println("[okay]");
//            }
          }
//          else {
//            System.out.println("[failed] could not be found");
//          }
        }
      } else if (message.equals(BrowserModel.NEW_CHART)) {
        // new chart -> no views waiting for new_view messages
        oldView = null;
        actualView = null;
      }
    }
  }

  /**
   * Animate from on oldview to actualView.
   */
  private void animate() {
    log("\nNormaly there should be a animation from " + oldView
        + " to " + actualView);
    try {
      TreeHelper.walkStatechart(model.getStateChart(), this, this);
    } catch (WorkException ex) {
      ex.printStackTrace();
    }
  }

  /**
   * Debug method for logging with kiel.util.LogFile.
   * @param str String
   */
  private void log(final String str) {
//    if (BrowserProperties.getLogLevel() > 0) {
//      System.out.println(str);
//    }
  }

  /**
   *
   * @param e Edge to do animation
   * @param n Node parent state
   * @throws WorkException if there is an error
   *
   */
  public void work(Edge e, kiel.dataStructure.Node n) throws WorkException {
    EdgeLayoutInformation morphSource = oldView.getLayoutInformation(e);
    EdgeLayoutInformation morphTarget = actualView.getLayoutInformation(e);
    boolean invisibleTarget = false;
    boolean missingSource = false;

    GroupNode g = (GroupNode)
                  toolkit.getElementById(BrowserCanvas.getGroupId(e));
    // if (g != null && g.getParent() != null && e.getSource().getParent()
    //      != null  && !g.getParent().getId().equals(
    //        BrowserCanvas.getGroupId(e.getSource().getParent()))) {
    //      // the object has changed in model (different parent)
    //      ((GroupNode) g).animatedRemove(true);
    //      g = null;
    //    }
    PathNode p = (PathNode) toolkit.getElementById(e.getID());
    if (g == null) {
      log("Edge " + e + " must be created and append");
      GroupNode gn = Kiel2ToolkitConverter.drawEdge(e, actualView, toolkit);
      toolkit.getElementById(BrowserCanvas.getGroupId(n.getParent()))
        .appendChild(gn);
      p = (PathNode) toolkit.getElementById(e.getID());
      morphSource = morphTarget;
      log("Seaching for Node " + n.getParent());
      if (p == null) {
        System.err.println("/!\\ Element missing AFTER drawing -> exit");
        System.exit(-1);
      }
      missingSource = true;
    }
//    if (morphSource == null) {
//      missingSource = true;
//      morphSource = morphTarget;
//    }

    log("Animating Edge: " + e + " " + e.getID());
    Node realParent = n.getParent();
    if (realParent != null && realParent.getClass() == Region.class) {
      realParent = realParent.getParent();
    }
```

```
        if (realParent != null
            && ((CompositeStateLayoutInformation) actualView.getLayoutInformation(
              realParent)).isCollapsed()) {
          invisibleTarget = true;
          log("Target is invisible!");
        }
        if (!missingSource
240         && realParent != null                                                    300
            && ((CompositeStateLayoutInformation) oldView.getLayoutInformation(
              realParent)).isCollapsed()) {
          missingSource = true;
          if (morphSource.getAllPathElements().size() == 0) {
            morphSource = morphTarget;
          }
          log("Source is missing!");
        }

250                                                                                310
        if (!invisibleTarget) {
          log("p ..........: " + p);
          log("Source .....: " + morphSource);
          log("Target .....: " + morphTarget);
          log("e.getSource.: " + e.getSource());
          log("e.getTarget.: " + e.getTarget());
          p.morphPath(morphSource.getAllPathElements(),
                      morphTarget.getAllPathElements(),
                      toolkit.getElementById(e.getSource().getID()),
260                   toolkit.getElementById(e.getTarget().getID()));              320
          p.setTooltip("" + e + ":" + actualView);
          if (e instanceof Transition) {
            if ((e.getSource().getOutgoingTransitions().size() > 1
                && ((Transition) e).getPriority().getValue() > 0)
                || (((Transition) e).getPriority().getValue() > 0
                && BrowserProperties.doAlwaysShowPriorities())) {
              // draw priority
              LabelLayoutInformation source =
                oldView.getLayoutInformation(((Transition) e).getPriority());
270           LabelLayoutInformation target =                                      330
                actualView.getLayoutInformation(((Transition) e).getPriority());
              if ((source != null) && (!source.isHidden())) {
                if ((source.getUpperLeft().getX()
                     != target.getUpperLeft().getX())
                    || (source.getUpperLeft().getY()
                        != target.getUpperLeft().getY())) {
                  kiel.browser.view.rendering.Node rn =
                    toolkit.getElementById(((Transition) e).getPriority().getID());
                  if (rn != null) {
280                 rn.animatePositionTo(target.getUpperLeft().getX(),             340
                                         target.getUpperLeft().getY());
                  } else {
                    Kiel2ToolkitConverter.drawPriorityLabel(e, actualView, toolkit,
                      (GroupNode) p.getParent());
                  }
                }
              }
            }
            if (((Transition) e).getLabel() != null) {
290           log("Animating label: " + ((Transition) e).getLabel().toString());   350
              LabelLayoutInformation source =
                oldView.getLayoutInformation(((Transition) e).getLabel());
              LabelLayoutInformation target =
                actualView.getLayoutInformation(((Transition) e).getLabel());
              if (((Transition) e).getLabel().toString().length() > 0) {
                kiel.browser.view.rendering.Node rn =
                  toolkit.getElementById(((Transition) e).getLabel().getID());
                if (rn != null) {
                  rn.animatePositionTo(target.getUpperLeft().getX(),
                                       target.getUpperLeft().getY());
                } else {
                  Kiel2ToolkitConverter.drawTransitionLabel(e, actualView, toolkit,
                    (GroupNode) p.getParent());
                  log("Label could not be found --> redrawing ("
                                + ((Transition) e).getLabel().toString() + ")");
                }
              }
            }
          }
          if (missingSource) {
            ((GroupNode) p.getParent()).animatedInsert();
          }
        } else if (invisibleTarget && !missingSource) {
          ((GroupNode) p.getParent()).animatedRemove(false);
        }

        log("Animating Edge: " + e  + " [done]");
      }

      /**
       * Called if TreeHelper enters a node.
       * @param n Node entered node
       * @throws WorkException if there is an error
       */
      public void startNode(kiel.dataStructure.Node n) throws WorkException {
        NodeLayoutInformation source = oldView.getLayoutInformation(n);
        NodeLayoutInformation target = actualView.getLayoutInformation(n);
        boolean invisibleTarget = false;
        boolean missingSource = false;
        log("Working on Node ID: " + n.getID() + " Name: "
                            + n.getName());

        kiel.browser.view.rendering.Node g =
                    toolkit.getElementById(BrowserCanvas.getGroupId(n));
//      if (g != null && g.getParent() != null && n.getParent() != null
//          && !g.getParent().getId().equals(
//          BrowserCanvas.getGroupId(n.getParent()))) {
//        // the object has changed in model (different parent)
//        ((GroupNode) g).animatedRemove(true);
//        g = null;
//      }
        if (g == null) {
          GroupNode gn = Kiel2ToolkitConverter.drawNode(n, actualView, toolkit);
          log("Node " + n + " is missing drawing and append to "
                                + n.getParent());
          if (n.getParent() != null && gn != null) {
            g = toolkit.getElementById(BrowserCanvas.getGroupId(n.getParent()));
            if (g != null) {
              g.appendChild(gn);
            } else {
              log("___NODE: " + n.getParent() + " cant be found___");
            }
```

```
      } else if (n.getParent() == null && gn != null) {
        toolkit.getRoot().appendChild(gn);
      }
      g = toolkit.getElementById(BrowserCanvas.getGroupId(n));
      source = actualView.getLayoutInformation(n);
      //      oldView = actualView;
      missingSource = true;
      if (g == null || gn == null) {
        log("/!\\ Node " + n + " missing AFTER drawing -> EXIT");
        System.exit(-1);
      }

    }
    log("Animating Node: " + n);

    Node realParent = n.getParent();
    if (realParent != null && realParent.getClass() == Region.class) {
      realParent = realParent.getParent();
    }
    if (realParent != null
        && ((CompositeStateLayoutInformation) actualView.getLayoutInformation(
          realParent)).isCollapsed()) {
      invisibleTarget = true;
      log("Target is invisible!");
    }
    if (!missingSource
        && realParent != null
        && ((CompositeStateLayoutInformation) oldView.getLayoutInformation(
          realParent)).isCollapsed()) {
      missingSource = true;
      log("Source is missing!");
    }

    if (!invisibleTarget) {
      Hashtable dellines = null;
      if (n instanceof ANDState) {
        Iterator iter = ((ANDState) n).getDelimiterLines().iterator();
        dellines = new Hashtable(((ANDState) n).getDelimiterLines().size());
        while (iter.hasNext()) {
          DelimiterLine line = (DelimiterLine) iter.next();
          dellines.put(line.getID(), line);
        }
      }
      Iterator iter = g.getChildNodes().iterator();
      while (iter.hasNext()) {
        kiel.browser.view.rendering.Node e =
          (kiel.browser.view.rendering.Node) iter.next();
        if ((e instanceof RectNode
             || e instanceof CircleNode)
            && e.getId().equals(n.getID())) {
          applyAnimationForWidthHeight(e, source, target);
          e.setTooltip("" + n + " : " + actualView);
        } else if (e.getId().equals("line")) {
          int linePos = Properties.getCompositeStateHeaderHeight();
          int width = source.getWidth();
          int aniWidth = target.getWidth();
          if (n.getClass() == FinalANDState.class
              || n.getClass() == FinalORState.class) {
            linePos += Properties.getFinalCompositeStateOffset();
            width -= Properties.getFinalCompositeStateOffset();
            aniWidth -= Properties.getFinalCompositeStateOffset();
          }
          if (width != aniWidth) {
            ((LineNode) e).animateLine(0, e.getY(), aniWidth, e.getY());
          }
        } else if (e.getId().equals("label")) {
          applyAnimationForLabel(n, (TextNode) e, source, target);
        } else if (e.getId().equals("relative")) {
          applyAnimationForRelativeNode(n, e, source, target);
        } else if (e.getId().equals("activities")) {
          applyAnimationForActivities(n, e, source, target);
        } else if (dellines != null && dellines.containsKey(e.getId())) {
          applyAnimationForDelimiter((LineNode) e,
                                     actualView.getLayoutInformation(
                                       (DelimiterLine) dellines.get(
                                         e.getId())));
          dellines.remove(e.getId());
        }
      }
      if (dellines != null && dellines.size() > 0) {
        iter = dellines.values().iterator();
        DelimiterLine line = null;
        DelimiterLineLayoutInformation dLayout = null;
        while (iter.hasNext()) {
          line = (DelimiterLine) iter.next();
          dLayout = actualView.getLayoutInformation(line);
          if (dLayout != null) {
            LineNode l = model.getToolkit().createLineElement();
            l.setXPos(dLayout.getStart().getX());
            l.setYPos(dLayout.getStart().getY());
            l.setEndXPos(dLayout.getEnd().getX());
            l.setEndYPos(dLayout.getEnd().getY());
            l.setDashed();
            l.setId(line.getID());
            model.getToolkit().getStyleChanger().changeStyle(l,
              Kiel2ToolkitConverter.getGraphicalLibrary().getStyle(
                GraphicalObjectsLibrary.STYLE_LINE));
            g.appendChild(l);
          }
        }
      }
      if ((target.getXPos() != source.getXPos())
          || (target.getYPos() != source.getYPos())) {
        g.animatePositionTo(target.getXPos(), target.getYPos());
      }
      if (missingSource) {
        ((GroupNode) g).animatedInsert();
      }
    } else if (invisibleTarget && !missingSource) {
      ((GroupNode) g).animatedRemove(false);
//    } else if (missingSource && !invisibleTarget) {
//      ((GroupNode)g).animatedInsert();
    }
//    else {
//      log("FOUND UNWANTED CASE!!!! DOING NOTHING");
//    }
//    log("Animating Node: " + n + " [done]");
  }
```

```
    /**
     * applyAnimationForRelativeNode
     *
     * @param n Node
     * @param e Node
     * @param source NodeLayoutInformation
     * @param target NodeLayoutInformation
     * @test WORKS
     */
    private void applyAnimationForRelativeNode(Node n,
                                               kiel.browser.view.rendering.Node e,
                                               NodeLayoutInformation source,
                                               NodeLayoutInformation target) {
      if (source.getHeight() != target.getHeight()
          || source.getWidth() != target.getWidth()) {
        int rW = 0;
        int rH = 0;
        if (n.getClass() == FinalState.class
            || n.getClass() == FinalSimpleState.class) {
          rW = Properties.getFinalStateOffset();
          rH = Properties.getFinalStateOffset();
        } else if (n.getClass() == FinalANDState.class
                || n.getClass() == FinalORState.class) {
          rW = Properties.getFinalCompositeStateOffset();
          rH = Properties.getFinalCompositeStateOffset();
        }
//      log("Relative width:" + rW + " rH:" + rH + " : " + target.getWidth()
//          + " " + target.getHeight());
        e.animateWidthTo(target.getWidth() - 2 * rW);
        e.animateHeightTo(target.getHeight() - 2 * rH);
        e.animatePositionTo(rW, rH);
      }
    }

    /**
     * applyAnimationForLabel
     *
     * @param e SVGElement
     * @param source NodeLayoutInformation
     * @param target NodeLayoutInformation
     * @test OK, TEST POSITIONING (BASELINE)
     */
    private void applyAnimationForLabel(kiel.dataStructure.Node n,
                                        kiel.browser.view.rendering.TextNode e,
                                        NodeLayoutInformation source,
                                        NodeLayoutInformation target) {
      if (source.getHeight() != target.getHeight()
          || source.getWidth() != target.getWidth()) {
        float height = e.getHeight();
        float width = e.getWidth();
        float x;
        float y;

        if (!((n instanceof kiel.dataStructure.ANDState)
            || (n instanceof kiel.dataStructure.ORState))) {
          x = ((float) target.getWidth() - width) / 2.0f;
          if (n instanceof State) {
            y = ((float) (target.getHeight()
                - Properties.getUpperOffset((State) n)
                - Properties.getLowerOffset((State) n))
                - height) / 2.0f;
            y += (float) Properties.getUpperOffset((State) n);
          } else {
            y = ((float) target.getHeight() - height) / 2.0f;
          }
          e.animatePositionTo((int) Math.round(x), (int) Math.round(y));
        } else {
          e.animatePositionTo((int) Math.round((target.getWidth()
                                                - width) / 2.0f),
                              Properties.getCompositeStateHeaderHeight()
                              - (int) height);
        }
      }
      if (n.getName() != null && n.getName().length() > 0) {
        e.setText(n.getName());
      }
    }

    /**
     * applyAnimationForDelimiter
     *
     * @param n Node
     * @param e SVGElement
     * @param source NodeLayoutInformation
     * @param target NodeLayoutInformation
     * @text WORKS
     */
    private void applyAnimationForDelimiter(
      kiel.browser.view.rendering.LineNode e,
      DelimiterLineLayoutInformation target) {
      log("Trying to animate to " + target.getStart() + " -> "
                              + target.getEnd());
      e.animateLine(target.getStart().getX(), target.getStart().getY(),
                    target.getEnd().getX(), target.getEnd().getY());
    }

    /**
     * applyAnimationForActivities
     *
     * @param e SVGElement
     * @param source NodeLayoutInformation
     * @param target NodeLayoutInformation
     * @test WORKS
     */
    private void applyAnimationForActivities(kiel.dataStructure.Node n,
                                             kiel.browser.view.rendering.Node e,
                                             NodeLayoutInformation source,
                                             NodeLayoutInformation target) {
      Iterator iter = e.getChildNodes().iterator();
      while (iter.hasNext()) {
        kiel.browser.view.rendering.Node el =
          (kiel.browser.view.rendering.Node) iter.next();
        if (el.getId().equals("region_limiter_right")) {
          if (source.getWidth() != target.getWidth()) {
            LineNode l = (LineNode) el;
            l.animateLine(target.getWidth(), l.getY(), target.getWidth(),
                          l.getEndY());
          }
        } else if (el.getId().equals("hori_limit")) {
          int aniWidth = target.getWidth();
```

```
                                                                    NodeLayoutInformation source,
                                                                    NodeLayoutInformation target) {
      int width = source.getWidth();
      if (n.getClass() == FinalANDState.class
          || n.getClass() == FinalORState.class) {                                                  620
        aniWidth -= Properties.getFinalCompositeStateOffset();                                      log("Actual width is " + source.getWidth());
        width -= Properties.getFinalCompositeStateOffset();                                         log("Actual height is " + source.getHeight());
      }                                                                                             log("Actual width is " + target.getWidth());
      if (width != aniWidth) {                                                                      log("Actual height is " + target.getHeight());
        LineNode l = (LineNode) el;                                                                 if (source.getWidth() != target.getWidth()) {
600     l.animateLine(l.getX(), l.getY(), aniWidth, l.getEndY());                                     e.animateWidthTo(target.getWidth());
      }                                                                                             }
    }                                                                                               if (source.getHeight() != target.getHeight()) {
  }                                                                                                   e.animateHeightTo(target.getHeight());
                                                                                                    }
                                                                                                  }                                                                  630
                                                                                                  /**
                                                                                                   * Called if TreeHelper exits a node.
                                                                                                   * @param n Node exited node
                                                                                                   * @throws WorkException if there is an error
                                                                                                   */
                                                                                                  public void endNode(Node n) throws WorkException {
                                                                                                    // nothing we are doing all stuff in the the startNode method
                                                                                                  }
                                                                                                }                                                                    640
  /**
   * applyAnimationForRect
   *
   * @param n Node
   * @param e SVGElement
   * @param source NodeLayoutInformation
   * @param target NodeLayoutInformation
   * @tst WORKS
   */
  private void applyAnimationForWidthHeight(kiel.browser.view.rendering.Node e,
```

F.5.14. BrowserStatusLine

```
package kiel.browser.view;

import java.awt.Dimension;
import java.awt.Font;
import java.util.Observable;

import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JSplitPane;

import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelMessage;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £ last modified £Date: 2006/02/08 07:07:49 £
 *
 */
public class BrowserStatusLine extends BrowserGUI {
  /**
   * Label showing the status information.
   */
  private JLabel status;
  /**
   * Label showing the editor information.
   */
  private JLabel textEditor;
  /**
   * For showing text label and browser label.
   */
  private JSplitPane pane;
  /**
   * Size of font.
   */
  private static final int FONT_SIZE = 10;
  /**
   * Split size.
   */
  private static final int SPLIT_SIZE = 420;
  /**
   * Pref size.
   */
  private static final int PREF_SIZE = 550;

  /**
   *
   * @param m BrowserModel
   */
  public BrowserStatusLine(final BrowserModel m) {
    super(m);

    status = PaintUtils.createJLabel();
    if (m.getStatus() != null) {
      status.setText(m.getStatus());
    } else {
      status.setText("");
    }
    status.setOpaque(false);
    status.setFont(new Font("Monospaced", Font.PLAIN, FONT_SIZE));

    textEditor = PaintUtils.createJLabel();
    textEditor.setText(m.getKitStatus());
    textEditor.setOpaque(false);
    textEditor.setFont(new Font("Monospaced", Font.PLAIN, FONT_SIZE));

    pane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                          status,
                          textEditor);
    pane.setPreferredSize(new Dimension(PREF_SIZE, FONT_SIZE));
    pane.setDividerLocation(SPLIT_SIZE);
    pane.setDividerSize(0);
    pane.setBorder(null);
  }

  /**
   *
   * @return Swing component for this class
   */
  public final JComponent getComponent() {
    return pane;
  }

  /**
   * This method is called whenever the observed object is changed.
   *
   * @param o the observable object.
   * @param arg an argument passed to the <code>notifyObservers</code> method.
   */
  public final void update(final Observable o, final Object arg) {
    if (arg.getClass() == ModelMessage.class) {
      ModelMessage message = (ModelMessage) arg;
      if (message.equals(BrowserModel.STATUS_CHANGED)) {
        status.setText(getModel().getStatus());
        status.setToolTipText(getModel().getStatus());
        status.repaint();
      } else if (message.equals(BrowserModel.KIT_STATUS_CHANGED)) {
        textEditor.setText(getModel().getKitStatus());
        textEditor.setToolTipText(getModel().getKitStatus());
        textEditor.repaint();
      }

    }
  }
}
```

F.5.15. PaintUtils

```
package kiel.browser.view;

import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.LayoutManager;
import java.awt.Paint;

import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JToolBar;
import javax.swing.plaf.metal.MetalLookAndFeel;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public final class PaintUtils {

  /**
   * Not for instantiation.
   *
   */
  private PaintUtils() {
  }

  /**
   *
   * @param c .
   * @param g .
   * @param topToBottom .
   */
  private static void paintComponent(final JComponent c, final Graphics g,
                                     final boolean topToBottom) {
    paintComponent(
      c,
      g,
      topToBottom,
      MetalLookAndFeel.getControl(),
      MetalLookAndFeel.getControlInfo());
  }

  /**
   *
   * @param c .
   * @param g .
   * @param topToBottom .
   * @param light .
   * @param dark .
   */
  private static void paintComponent(final JComponent c, final Graphics g,
                                     final boolean topToBottom,
                                     final Color light, final Color dark) {

    Paint oldPaint = ((Graphics2D) g).getPaint();

    if (topToBottom) {
      ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                                                  light, 0, c.getSize().height,
                                                  dark, true));
    } else {
      ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                                                  dark, c.getSize().width, 0,
                                                  light, true));
    }
    ((Graphics2D) g).fillRect(0, 0, c.getSize().width, c.getSize().height);
    ((Graphics2D) g).setPaint(oldPaint);
  }

  /**
   * .
   * @param title .
   * @param shortcut .
   * @param icon .
   * @return .
   */
  public static JMenu createJMenu(final String title, final String shortcut,
                                  final String icon) {
//    JMenu menu = createJMenu(title, shortcut);
//    menu.setOpaque(true);
//    return menu;
    return null;
  }

//  /**
//   *
//   * @param title .
//   * @param shortcut .
//   * @return .
//   */
//  public static JMenu createJMenu(final String title, final String shortcut) {
//
//    /**
//     *
//     * @author <a href="mailto:flu@informatik.uni-kiel.de">
//     * Florian Luepke </a>
//     */
//    class MyMenu extends JMenu implements Preferences.Listener {
```

```
//
//      /**
//       *
//       *
//       */
//      MyMenu() {
//        setOpaque(false);
120 //        Preferences.addListener(this);
//        preferencesChanged();
//      }
//
//      /**
//       *
//       * @see kiel.editor.resources.Preferences.Listener#
//       * preferencesChanged()
//       */
//      public void preferencesChanged() {
130 //        setText(ResourceBundle.getString(title));
//        if (shortcut != null) {
//          setMnemonic(ResourceBundle.getString(shortcut).charAt(0));
//        }
//      }
//    }
//
//
//    return new MyMenu();
//  }
140
  /**
   *
   * @param items .
   * @return .
   */
  public static JComboBox createJComboBox(final String[] items) {
    return new JComboBox(items) {
      {
        setOpaque(false);
150     for (int i = 0; i < getComponentCount(); i++) {
          if (getComponent(i) instanceof JComponent) {
            ((JComponent) getComponent(i)).setOpaque(false);
          }
        }
        ((JComponent) getRenderer()).setOpaque(false);
      }
    };
  }

160 /**
   *
   * @return .
   */
  public static JSlider createJSlider() {
    return new JSlider() {
      {
        setOpaque(false);
        for (int i = 0; i < getComponentCount(); i++) {
          if (getComponent(i) instanceof JComponent) {
170         ((JComponent) getComponent(i)).setOpaque(false);
          }
        }
      }
    };
  }

  /**
   *
   * @return .
180 */
  public static JToolBar createJToolBar() {
    return new JToolBar() {
      protected void paintComponent(final Graphics g) {
        PaintUtils.paintComponent(this, g, true);
      }
    };
  }

  /**
190 *
   * @param topToBottom .
   * @return .
   */
  public static JToolBar createJToolBar(final boolean topToBottom) {
    return new JToolBar() {
      protected void paintComponent(final Graphics g) {
        PaintUtils.paintComponent(this, g, topToBottom);
      }
    };
200 }

  /**
   *
   * @param layout .
   * @return .
   */
  public static JPanel createJPanel(final LayoutManager layout) {
    return new JPanel(layout) {
      protected void paintComponent(final Graphics g) {
210     PaintUtils.paintComponent(this, g, true);
      }
    };
  }

  /**
   *
   * @param layout .
   * @param topToBottom .
   * @return .
220 */
  public static JPanel createJPanel(final LayoutManager layout,
                                    final boolean topToBottom) {
    return new JPanel(layout) {
      protected void paintComponent(final Graphics g) {
        PaintUtils.paintComponent(this, g, topToBottom);
      }
    };
  }

230 /**
   *
   * @return .
```

```
 */
public static JLabel createJLabel() {
  return new JLabel() {
    protected void paintComponent(final Graphics g) {
      PaintUtils.paintComponent(this, g, true);
      super.paintComponent(g);
    }
  };
}

/**
 *
 * @return .
 */
public static JLabel createJLabelHorizontal() {
  return new JLabel() {
    protected void paintComponent(final Graphics g) {
      PaintUtils.paintComponent(this, g, false);
      super.paintComponent(g);
    }
  };
}

/**
 *
 * @return .
 */
public static JMenuBar createJMenuBar() {
  return new JMenuBar() {
    protected void paintComponent(final Graphics g) {
      PaintUtils.paintComponent(this, g, false);
    }
  };
}
}
```

F.5.16. StateChartMarker

```
package kiel.browser.view;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import kiel.dataStructure.Edge;
import kiel.dataStructure.Node;
import kiel.dataStructure.PseudoState;
import kiel.configMngr.Configuration;
import kiel.configMngr.ExecuteTransition;
import kiel.configMngr.MacroStep;
import kiel.configMngr.MicroStep;
import kiel.configMngr.OnEntry;
import kiel.configMngr.OnExit;
import kiel.configMngr.OnInside;
import kiel.configMngr.StateActivated;
import kiel.configMngr.StateDeactivated;
import kiel.configMngr.StateStatus;
import kiel.configMngr.StateSuspended;
import kiel.configMngr.TestTransition;
import kiel.configMngr.TransitionStatus;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelMessage;
import kiel.browser.view.rendering.Toolkit;
import kiel.preferences.LogFile;
import java.util.StringTokenizer;
import java.util.ArrayList;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.22 £ last modified £Date: 2007/05/04 12:46:34 £
 */
public class StateChartMarker implements Observer {
  /**
   *
   */
  private BrowserModel model;
  /**
   *
   */
  private Toolkit toolkit;
  /**
   *
   */
  private MacroStep actualMacroStep;

  /**
   *
   */
  private Configuration actualConf;

  /**
   *
   */
  private Configuration oldConf;

  /**
   *
   */
  private Hashtable stepMarkingTable = new Hashtable();

  /**
   *  Table of marked Elements.
   */
  private Hashtable table = new Hashtable();

  /**
   *
   */
  private boolean isMicroStepMode = false;


  /**
   *
   * @param aModel BrowserModel
   * @param aChart SVGStateChart
   */
  public StateChartMarker(final BrowserModel aModel,
                          final Toolkit aChart) {
    model = aModel;
    toolkit = aChart;
  }

  /**
   *
   * @param aMacroStep MacroStep
   */
  public final void setMacroStep(final MacroStep aMacroStep) {
    actualMacroStep = aMacroStep;
  }

  /**
   *
   * @param aConf Configuration
   */
  public final void setConfiguration(final Configuration aConf) {
    oldConf = actualConf;
    actualConf = aConf;
  }

  /**
```

```
     * Resets configuration. Need to be set if you restart a marking session.
     */
    public final void resetConfigurations() {
      oldConf = null;
      actualConf = null;
    }

    /**
     *
     * @param table Hashtable
     * @param style String
     * @param id String
     */
    private static void addStyleToTable(final Hashtable table,
                                        final String style,
                                        final String id) {
      if (!table.containsKey(id)) {
        table.put(id, style);
      }
    }


    /**
     * marks the elements according to the microsteps.
     */
    public final void handleActualMicroStep() {
      if (model.getMicroStep() != null) {
        if (oldConf != null && model.getMicroStep().equals(
          actualMacroStep.getMicroSteps().getFirst())) {
          handleConfiguration(oldConf);
        }
        handleMicroStep(model.getMicroStep());
        if (actualConf != null && model.getMicroStep().equals(
          actualMacroStep.getMicroSteps().getLast())) {
          handleConfiguration(actualConf);
        }
      }
    }

    /**
     * marks the elements according to the microsteps.
     */
    public final void handleAllMicroSteps() {
      MicroStep m;
      for (int i = 0; i < actualMacroStep.getMicroSteps().size(); i++) {
        m = (MicroStep) actualMacroStep.getMicroSteps().get(i);
        handleMicroStep(m);
      }
      handleConfiguration(actualConf);
    }

    /**
     *  marks states in active configuration.
     *  @param conf Configuration to handle
     */
    public final void handleConfiguration(final Configuration conf) {
        // added khe 2007-03-07
        if (BrowserProperties.isHighlightActiveStates()) {
            Node[] nodes = conf.getNodes();
            for (int i = 0; i < nodes.length; i++) {
                Node n = nodes[i];
                if (!(n instanceof PseudoState)) {
                    kiel.browser.view.rendering.Node s = toolkit.getElementById(
                            n.getID());
                    if (s != null) {
                        addStyleToTable(stepMarkingTable, s.getStyle(),
                                n.getID());
                        toolkit.getStyleChanger().changeStyle(s,
                                BrowserProperties.
                                getActualConfig());
                    } else {
                        model.getBrowserLogFile().log(LogFile.ERROR, "Element "
                                + n.getID()
                                + " is missing, possibly it is hidden in "
                                + "an collapsed state. Please autolayout.");
                    }
                }
            }
        }
    }

    /**
     * @param m MicroStep handles marking for microstep m
     */
    public final void handleMicroStep(final MicroStep m) {
      if (m instanceof TransitionStatus) {
        Edge e = ((TransitionStatus) m).getTransition();
        kiel.browser.view.rendering.Node s = toolkit.getElementById(e.getID());
        if (s != null) {
          if (m instanceof ExecuteTransition) {
            addStyleToTable(stepMarkingTable, s.getStyle(), e.getID());
            toolkit.getStyleChanger().changeStyle(s,
                                                  BrowserProperties.
                                                  getExecuteTransition());
          } else if (m instanceof TestTransition) {
            addStyleToTable(stepMarkingTable, s.getStyle(), e.getID());
            toolkit.getStyleChanger().changeStyle(s,
                                                  BrowserProperties.
                                                  getTestTransition());
          }
        }
      } else if (m instanceof StateStatus) {
        Node n = ((StateStatus) m).getState();
        kiel.browser.view.rendering.Node s = toolkit.getElementById(n.getID());
        if (s != null) {
          if (m instanceof OnInside) {
            addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
            toolkit.getStyleChanger().changeStyle(s,
                    BrowserProperties.getOnInside());
          } else if (m instanceof OnExit) {
            addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
            toolkit.getStyleChanger().changeStyle(s,
                    BrowserProperties.getOnExit());
          } else if (m instanceof OnEntry) {
            addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
            toolkit.getStyleChanger().changeStyle(s,
                    BrowserProperties.getOnEntry());
          } else if (m instanceof StateSuspended) {
            addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
            toolkit.getStyleChanger().changeStyle(s,
```

```
                BrowserProperties.getStateSuspended());
        } else if (m instanceof StateActivated) {
          addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
          toolkit.getStyleChanger().changeStyle(s,
                BrowserProperties.getStateActivated());
        } else if (m instanceof StateDeactivated) {
          addStyleToTable(stepMarkingTable, s.getStyle(), n.getID());
          toolkit.getStyleChanger().changeStyle(s,
                BrowserProperties.getStateDeactivated());
        }
      }
    }
  }

  /**
   * resetMarkedElements.
   */
  public final void resetStepMarking() {
    Iterator iter = stepMarkingTable.keySet().iterator();
    while (iter.hasNext()) {
      String id = (String) iter.next();
      kiel.browser.view.rendering.Node element = toolkit.getElementById(id);
      if (element != null) {
        toolkit.getStyleChanger().changeStyle(element,
                        (String) stepMarkingTable.get(id));
      }
    }
    stepMarkingTable.clear();
  }

  /**
   * reset focus elements.
   */
  public final void resetFocusMarking() {
    Iterator iter = table.keySet().iterator();
    while (iter.hasNext()) {
      String id = (String) iter.next();
      kiel.browser.view.rendering.Node element = toolkit.getElementById(id);
      if (element != null) {
        toolkit.getStyleChanger().changeStyle(element,
                        (String) table.get(id));
      }
    }
    table.clear();
  }

  /**
   *
   * @param o Observable
   * @param arg Object
   */
  public final void update(final Observable o, final Object arg) {
    if (arg.getClass() == ModelMessage.class) {
      ModelMessage message = (ModelMessage) arg;
      if (message.equals(BrowserModel.SIMRESET)
          || message.equals(BrowserModel.RESET)) {
        resetStepMarking();
        resetConfigurations();
        resetFocusMarking();
        toolkit.repaintCanvas();
      } else if (message.equals((BrowserModel.MODE_CHANGE))) {
        switch(model.getMode()) {
        case BrowserModel.EDIT:
//          System.out.println("Switching to EDIT MODE");
          // switching from simmode to edit mode
          resetStepMarking();
          resetConfigurations();

          if (model.getSelectedStates()[0] != null) {
            markNode1(model.getSelectedStates()[0].getID());
          }
          if (model.getSelectedStates()[1] != null) {
            markNode2(model.getSelectedStates()[1].getID());
          }
          if (model.getSelectedTransition() != null) {
            markEdge(model.getSelectedTransition().getID());
          }
          toolkit.repaintCanvas();
          break;
        case BrowserModel.SIMULATION:
//          System.out.println("Switching to SIM MODE");
          // switching from edit mode to sim mode
          resetFocusMarking();
          resetConfigurations();
          toolkit.repaintCanvas();
          break;
        }
//      } else if (message.equals((BrowserModel.UPDATE_CHART))) {
//        Iterator iter = new ArrayList(table.keySet()).iterator();
//        while(iter.hasNext()) {
//          releaseMarkedElement((String) iter.next());
//        }
//      } else if (message.equals((BrowserModel.SIMSTEP_STOP))) {
//        if (model.getSelectedStates()[0] != null) {
//          markNode1(model.getSelectedStates()[0].getID());
//        }
//        if (model.getSelectedStates()[1] != null) {
//          markNode2(model.getSelectedStates()[1].getID());
//        }
//        if (model.getSelectedTransition() != null) {
//          markEdge(model.getSelectedTransition().getID());
//        }
      } else if (message.equals((BrowserModel.SIMSTEP_COMPLETED))) {
//        System.out.print("HANDLING STEPS ... MARKING  ->");
//        if (model.getSelectedStates()[0] != null) {
//          releaseMarkedElement(model.getSelectedStates()[0].getID());
//        }
//        if (model.getSelectedStates()[1] != null) {
//          releaseMarkedElement(model.getSelectedStates()[1].getID());
//        }
//        if (model.getSelectedTransition() != null) {
//          releaseMarkedElement(model.getSelectedTransition().getID());
//        }
        if (isMicroStepMode) {
//          System.out.println("MicroSTEP!");
          handleActualMicroStep();
        } else {
//          System.out.println("MacroSTEP!");
          handleAllMicroSteps();
        }
```

```
      toolkit.repaintCanvas();
    } else if (message.equals(BrowserModel.NEW_CHART)) {
      resetConfigurations();
      toolkit.repaintCanvas();
    } else if (message.equals(BrowserModel.SIM_MODE_MACRO)) {
//      System.out.println("SWITCHING TO MACRO MODE");
      isMicroStepMode = false;
    } else if (message.equals(BrowserModel.SIM_MODE_MICRO)) {
//      System.out.println("SWITCHING TO MICRO MODE");
      isMicroStepMode = true;
    } else if (message.equals(BrowserModel.NEW_MICRO)) {
//      System.out.println("A new MICRO was SET");
      if (model.getMicroStep().equals(
        actualMacroStep.getMicroSteps().getFirst())) {
        resetStepMarking();
      }
    } else if (message.equals(BrowserModel.NEW_MACRO)) {
//      System.out.println("A new MACRO was SET");
      resetStepMarking();
      setMacroStep(model.getMacroStep());
    } else if (message.equals(BrowserModel.NEW_TOOLKIT)) {
      resetStepMarking();
      toolkit = model.getToolkit();
      if (isMicroStepMode && model.getMicroStep() != null) {
        handleActualMicroStep();
      } else if (actualMacroStep != null) {
        handleAllMicroSteps();
      }
      toolkit.repaintCanvas();
    } else if (message.equals(BrowserModel.NEW_CONFIGURATION)) {
      setConfiguration(model.getConfiguration());
//      System.out.println("SWITCHING CONFIGURATION FROM " + oldConf + " TO "
//                         + actualConf);
      if (BrowserProperties.isColorizeNewConfigurations()) {
          handleConfiguration(model.getConfiguration());
      }
    } else if (message.equals(BrowserModel.FOCUS_CHANGED)
               && model.getMode() == BrowserModel.EDIT) {
//      System.out.println("DRAWING NEW FOCUS IN " + model.getMode());
      if (model.getFocus().hasEdgeChanged()
          && model.getFocus().getEdge() != null) {
        markEdge(model.getFocus().getEdge().getID());
      }
      if (model.getFocus().hasNode1Changed()
          && model.getFocus().getNodes()[0] != null) {
        markNode1(model.getFocus().getNodes()[0].getID());
      }
      if (model.getFocus().hasNode2Changed()
         && model.getFocus().getNodes()[1] != null) {
        markNode2(model.getFocus().getNodes()[1].getID());
      }
    } else if (message.equals(BrowserModel.REMOVE_SELECTION)) {
//      System.out.println("Remove Selection on: " + message.getParameter());
      if (((String) message.getParameter()).length() > 0) {
        releaseMarkedElement((String) message.getParameter());
      } else {
        Iterator iter = new ArrayList(table.keySet()).iterator();
        while (iter.hasNext()) {
          releaseMarkedElement((String) iter.next());
        }
      }
    } else if (message.equals(BrowserModel.MARK_OBJECT)) {
      StringTokenizer token = new StringTokenizer((String)
                                                  message.getParameter(),
                                                  "||");
      markObject(token.nextToken(), token.nextToken());
    }
  }

  /**
   * @param string String
   */
  private void releaseMarkedElement(final String string) {
    kiel.browser.view.rendering.Node s = toolkit.getElementById(string);
    if (s != null) {
      String style = (String) table.remove(string);
      if (style != null) {
        toolkit.getStyleChanger().changeStyle(s, style);
        toolkit.repaintCanvas();
      }
    }
  }

  /**
   * @param aId String GraphicalObject id.
   * @param css CSS-conforming String.
   */
  private void markObject(final String aId, final String css) {
    kiel.browser.view.rendering.Node s = toolkit.getElementById(aId);
    if (s != null && table.get(aId) == null) {
      // add just for the first time
      table.put(aId, s.getStyle());
    }
    if (s != null) {
      toolkit.getStyleChanger().changeStyle(s, css);
      toolkit.repaintCanvas();
    }
  }

  /**
   * @param aString String id of node to mark
   */
  private void markEdge(final String aString) {
    kiel.browser.view.rendering.Node s = toolkit.getElementById(aString);
    if (s != null && table.get(aString) == null) {
      table.put(aString, s.getStyle());
      toolkit.getStyleChanger().changeStyle(s, "fill:none;stroke:"
                                            + BrowserCanvas.encodeColorForCSS(
                                              BrowserProperties.
                                              getEdgeMarkColor())
                                            + ";stroke-width:1.5pt;");
      toolkit.repaintCanvas();
    } else if (s != null) {
      table.remove(aString);
      table.put(aString, s.getStyle());
      toolkit.getStyleChanger().changeStyle(s, "fill:none;stroke:"
                                            + BrowserCanvas.encodeColorForCSS(
                                              BrowserProperties.
                                              getEdgeMarkColor())
```

```
                                                + ";stroke-width:1.5pt;");
      toolkit.repaintCanvas();
    }

  }

  /**
   * @param string String id of node to mark
   */
  private void markNode1(final String string) {
    kiel.browser.view.rendering.Node s = toolkit.getElementById(string);
    if (s != null && table.get(string) == null) {
      table.put(string, s.getStyle());
    }
    toolkit.getStyleChanger().changeStyle(s, "fill:"
        + BrowserCanvas.encodeColorForCSS(BrowserProperties.getNodeMarkColor())
        + ";stroke:black;stroke-width:1.0pt;");
    toolkit.repaintCanvas();
  }

  /**
   * @param string String id of node to mark
   */
  private void markNode2(final String string) {
    kiel.browser.view.rendering.Node s = toolkit.getElementById(string);
    if (s != null && table.get(string) == null) {
      table.put(string, s.getStyle());
    }
    toolkit.getStyleChanger().changeStyle(s, "fill:"
        + BrowserCanvas.encodeColorForCSS(BrowserProperties.
            getNodeMarkBrightColor())
        + ";stroke:black;stroke-width:1.0pt;");
    toolkit.repaintCanvas();
  }
}
```

F.5.17. LineNumber

```
package kiel.browser.view;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.HashMap;

import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.plaf.metal.MetalLookAndFeel;

import kiel.browser.Browser;
import kiel.util.languages.ILineNumber;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Shows line numbers to a textpane.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:49 £
 */
class LineNumber extends JComponent implements MouseListener, ILineNumber {
  /**
   *
   */
  private static final Color DEFAULT_BACKGROUND = MetalLookAndFeel.getControl();
  /**
   *
   */
  private static final Color DEFAULT_FOREGROUND = Color.black;
  /**
   *
   */
  private static final Font DEFAULT_FONT = new Font("monospaced", Font.PLAIN,
    12);
  /**
   *
   */
  private static final int BIG = 1000000;
  /**
   *
   */
  private static final int HEIGHT = Integer.MAX_VALUE - BIG;
  /**
   * Set left margin.
   */
  private static final int LEFT_MARGIN = 5;
  /**
   * Set left margin.
   */
  private static final int RIGHT_MARGIN = 15;
  /**
   *
   */
  private FontMetrics fontMetrics;
  /**
   *
   */
  private int lineHeight;
  /**
   *
   */
  private int currentDigits;
  /**
   *
   */
  private JComponent component;
  /**
   *
   */
  private int componentFontHeight;
  /**
   *
   */
  private int componentFontAscent;
  /**
   *
   */
  private HashMap icons = new HashMap();
  /**
   *
   */
  private HashMap text = new HashMap();

  /**
   *  Convenience constructor for Text Components.
   * @param aComponent component to add linenumbers.
   */
  public LineNumber(final JComponent aComponent) {
    if (aComponent == null) {
      setFont(DEFAULT_FONT);
      this.component = this;
    } else {
      setFont(aComponent.getFont());
      this.component = aComponent;
    }
    setBackground(DEFAULT_BACKGROUND);
    setForeground(DEFAULT_FOREGROUND);
    final int defaultLines = 99;
    setPreferredWidth(defaultLines);
    addMouseListener(this);
  }
```

```
    /**
     *  Calculate the width needed to display the maximum line number.
     * @param lines maximum line numbers
     */
    public void setPreferredWidth(final int lines) {
      int digits = String.valueOf(lines).length();

      //  Update sizes when number of digits in the line number changes

      if (digits != currentDigits && digits > 1) {
        currentDigits = digits;
        int width = fontMetrics.charWidth('0') * digits;
        Dimension d = getPreferredSize();
        d.setSize(LEFT_MARGIN + width + RIGHT_MARGIN, HEIGHT);
        setPreferredSize(d);
        setSize(d);
      }
    }

    /**
     *  Reset variables that are dependent on the font.
     *  @param font Font
     */
    public void setFont(final Font font) {
      super.setFont(font);
      fontMetrics = getFontMetrics(getFont());
      componentFontHeight = fontMetrics.getHeight();
      componentFontAscent = fontMetrics.getAscent();
    }

    /**
     *  The line height defaults to the line height of the font for this
     *  component.
     *  @return int
     */
    public int getLineHeight() {
      if (lineHeight == 0) {
        return componentFontHeight;
      } else {
        return lineHeight;
      }
    }

    /**
     *  Override the default line height with a positive value.
     *  For example, when you want line numbers for a JTable you could
     *  use the JTable row height.
     *  @param aLineHeight int
     */
    public void setLineHeight(final int aLineHeight) {
      if (aLineHeight > 0) {
        lineHeight = aLineHeight;
      }
    }

    /**
     *
     * @return int
     */
    public int getStartOffset() {
      return component.getInsets().top + componentFontAscent;
    }

    /**
     *
     * @param line int
     * @param icon ImageIcon
     * @param message String
     */
    public void addIconOnLine(final int line,
                              final ImageIcon icon,
                              final String message) {
      if (icons.containsKey(String.valueOf(line))) {
        icons.remove(String.valueOf(line));
        text.remove(String.valueOf(line));
      }
      icons.put(String.valueOf(line), icon);
      text.put(String.valueOf(line), message);
      this.repaint();
    }

    /**
     *
     * @param line int
     */
    public void removeIconOnLine(final int line) {
      if (icons.containsKey(String.valueOf(line))) {
        icons.remove(String.valueOf(line));
        text.remove(String.valueOf(line));
        this.repaint();
      }
    }

    /**
     * Clears list with icons.
     */
    public void clearIcons() {
      icons.clear();
      text.clear();
      this.repaint();
    }


    /**
     *
     * @param g Graphics
     */
    public void paintComponent(final Graphics g) {
      int aLineHeight = getLineHeight();
      int startOffset = getStartOffset();
      Rectangle drawHere = g.getClipBounds();

      // Paint the background

      g.setColor(getBackground());
      g.fillRect(drawHere.x, drawHere.y, drawHere.width, drawHere.height);

      //  Determine the number of lines to draw in the foreground.
```

```
        g.setColor(getForeground());
        int startLineNumber = (drawHere.y / aLineHeight) + 1;
        int endLineNumber = startLineNumber + (drawHere.height / aLineHeight);

        int start = (drawHere.y / aLineHeight) * aLineHeight + startOffset;
        final int iconheight = 9;
        int iconoffset = (aLineHeight - iconheight);

        for (int i = startLineNumber; i <= endLineNumber; i++) {
          String lineNumber = String.valueOf(i);
          int stringWidth = fontMetrics.stringWidth(lineNumber);
          int rowWidth = getSize().width;
          g.drawString(lineNumber, rowWidth - stringWidth - RIGHT_MARGIN, start);
          if (icons.containsKey(lineNumber)) {
            ImageIcon icon = (ImageIcon) icons.get(lineNumber);
            icon.paintIcon(this, g, rowWidth - iconheight,
                start - aLineHeight + iconoffset);
          }
          start += aLineHeight;
        }

        int rows = component.getSize().height / componentFontHeight;
        setPreferredWidth(rows);
      }

      /**
       *
       * @param e MouseEvent
       */
      public void mouseClicked(final MouseEvent e) {
        int line = ((int) e.getPoint().getY() / getLineHeight()) + 1;
        if (text.containsKey(String.valueOf(line))) {
          Browser.getKielFrame().handleException("Kit␣:␣"
                                                 + text.get(String.valueOf(line)),
                                                 false, null);
        }
      }

      /**
       * @param e MouseEvent
       */
      public void mouseEntered(final MouseEvent e) {
      }

      /**
       * @param e MouseEvent
       */
      public void mouseExited(final MouseEvent e) {
      }

      /**
       * @param e MouseEvent
       */
      public void mousePressed(final MouseEvent e) {
      }

      /**
       * @param e MouseEvent
       */
      public void mouseReleased(final MouseEvent e) {
      }
    }
```

F.5.18. Kiel2ToolkitConverter

```
package kiel.browser.view;

import java.awt.font.FontRenderContext;
import java.awt.font.TextLayout;
import java.awt.geom.AffineTransform;
import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.PseudoState;
import kiel.dataStructure.Region;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.StringLabel;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.Transition;
import kiel.dataStructure.action.ActionsStringLabel;
import kiel.dataStructure.eventexp.Event;
import kiel.configMngr.Configuration;
import kiel.graphicalInformations.CompositeStateLayoutInformation;
import kiel.graphicalInformations.DelimiterLineLayoutInformation;
import kiel.graphicalInformations.EdgeLayoutInformation;
import kiel.graphicalInformations.LabelLayoutInformation;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.MovetoPath;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;
import kiel.browser.Browser;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.browser.model.ModelMessage;
import kiel.browser.view.rendering.CircleNode;
import kiel.browser.view.rendering.GroupNode;
import kiel.browser.view.rendering.LineNode;
import kiel.browser.view.rendering.PathNode;
import kiel.browser.view.rendering.RectNode;
import kiel.browser.view.rendering.TextNode;
import kiel.browser.view.rendering.Toolkit;
import kiel.util.Benchmark;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.22 £ last modified £Date: 2007/06/04 06:58:19 £
 *
 */
public class Kiel2ToolkitConverter implements Observer {
  /**
   * A big whitespace.
   */
  private static final int BIGWHITESPACE = 20;
  /**
   *
   */
  private BrowserModel model;
  /**
   *
   */
  private Toolkit toolkit;
  /**
   *
   */
  private StateChart chart;
  /**
   *
   */
  private static String descFont;
  /**
   *
   */
  private static String headerFont;
  /**
   *
   */
  private static String labelFont;
  /**
   *
   */
  private static int labelFontSize;
  /**
   *
   */
  private static int headerFontSize;
```

```
    /**
     *
     */
    private static int descFontSize;
    /**
     *
     */
    private View oldView;
    /**
     *
     */
    private View actualView;
    /**
     *
     */
    private CompositeState rootState;
    /**
     *
     */
    private double actualRatio = 1.0d;
    /**
     * GraphicalObjectsLibrary (Handles Templates).
     */
    private static GraphicalObjectsLibrary lib = null;
    /**
     * Benchmarking drawing time.
     */
    private Benchmark drawingBench = new Benchmark();
    /**
     * Shown configuration.
     */
    private Configuration conf = null;

    /**
     * @param aModel BrowserModel
     * @param aToolkit aToolkit
     */
    public Kiel2ToolkitConverter(final BrowserModel aModel,
                                 final Toolkit aToolkit) {
      model = aModel;
      toolkit = aToolkit;
      chart = null;
      lib = GraphicalObjectsLibrary.getInstance(toolkit);
      drawingBench.setModuleName("BrowserDrawing Bechmark");
      drawingBench.printOutputToConsole(false);
    }

    /**
     *
     * @param o Observable
     * @param arg Object
     */
    public final void update(final Observable o, final Object arg) {
      if (arg.getClass() == ModelMessage.class) {
        ModelMessage message = (ModelMessage) arg;
        if (message.equals(BrowserModel.SIMRESET)
            || message.equals(BrowserModel.RESET)) {
          resetPropertyValues();
          //        model.startComputation();
          //        computeRatio();
          //        System.out.println("RESET ---> REDRAW CANVAS");
          //        draw();
          //        model.completeComputation();
        } else if (message.equals(BrowserModel.NEW_TOOLKIT)) {
          model.startComputation();
          toolkit = model.getToolkit();
          draw(toolkit);
          model.completeComputation();
        } else if (message.equals(BrowserModel.UPDATE_CHART)) {
          //        System.out.println("UPDATE CHART ");
          chart = model.getStateChart();
          setTemplateLibraryType();
        } else if (message.equals(BrowserModel.NEW_CHART)) {
          //        System.out.println("NEW CHART ");
          resetPropertyValues();
          chart = model.getStateChart();
          oldView = null;
          actualView = null;
          setTemplateLibraryType();
          if (BrowserProperties.benchmarkDrawing()) {
            drawingBench.printOutputToFile(new File(
                    XMLPreferences.getSafeWriteDirectory() + File.separator
                    + BrowserProperties.getDrawingBenchName()
                    + chart.getRootNode() + ".txt"));
          }
        } else if (message.equals(BrowserModel.NEW_CONFIGURATION)) {
          if (conf != null && conf.equals(model.getConfiguration())
            &&  BrowserProperties.benchmarkDrawing()) {
            drawingBench.start();
            drawingBench.stop();
          }
          conf = model.getConfiguration();
        } else if (message.equals(BrowserModel.NEW_VIEW)) {
          //        computeRatio();
          //        System.out.print("NEXT VIEW ---> KIEL CONVERTER DRAWS CANVAS? ");
          oldView = actualView;
          actualView = model.getView();
          if (oldView == null || !BrowserProperties.doAnimations()) {
            //      System.out.println("[YES]");
            model.startComputation();
            if (BrowserProperties.benchmarkDrawing()) {
              drawingBench.start();
            }
            draw(toolkit);
            if (BrowserProperties.benchmarkDrawing()) {
              drawingBench.stop();
            }
            model.completeComputation();
          } else {
            // no chart change but view changes so animator has to run....
            //        System.out.println("[NO]");
          }
        }
      }
    }

    /**
     *  Resets properties. Please pay attention that a reload causes possibly
     *  overwritten values to be defaults again (pseudo state width etc).
     *  @see #setTemplateLibraryType
```

```
   */
  private void resetPropertyValues() {
    Properties.reload();
    descFont = "\"" + Properties.getDescriptionFont().getFamily()
               + "\"";
    headerFont = "\"" + Properties.getActivityFont().getFamily()
                 + "\"";
    labelFont = "\"" + Properties.getLabelFont().getFamily() + "\"";

    labelFontSize = Properties.getLabelFont().getSize();
    headerFontSize = Properties.getActivityFont().getSize();
    descFontSize = Properties.getDescriptionFont().getSize();
    setTemplateLibraryType();
  }

  /**
   *
   * @see kiel.browser.view.GraphicalObjectsLibrary
   */
  private void setTemplateLibraryType() {
    if (model.getStateChart().getModelSource().
        startsWith("Matlab/Stateflow")) {
      model.getBrowserLogFile().log(LogFile.INFO,
                                    "Loading Stateflow templates...");
      lib.setTemplateType(GraphicalObjectsLibrary.STATEFLOW_TEMPLATES);
      model.getBrowserLogFile().log(LogFile.INFO,
                                    "...[done]");
    } else if (model.getStateChart().getModelSource().
               startsWith("Esterel Studio")) {
      model.getBrowserLogFile().log(
        LogFile.INFO, "Loading Esterel Studio templates...");
      lib.setTemplateType(GraphicalObjectsLibrary.ESTUDIO_TEMPLATES);
      model.getBrowserLogFile().log(LogFile.INFO,
                                    "...[done]");
    } else if (model.getStateChart().getModelSource().
          indexOf("UML") > -1) {
        model.getBrowserLogFile().log(
          LogFile.INFO, "Loading UML templates...");
        lib.setTemplateType(GraphicalObjectsLibrary.UML_TEMPLATES);
        model.getBrowserLogFile().log(LogFile.INFO,
                                      "...[done]");
    }  else {
      model.getBrowserLogFile().log(
        LogFile.INFO, "Loading Default templates...");
      lib.setTemplateType(GraphicalObjectsLibrary.DEFAULT_TEMPLATES);
      model.getBrowserLogFile().log(LogFile.INFO,
                                    "...[done]");
    }
  }


  /**
   * starts the whole rendering process.
   * @param t Toolkit
   */
  public final void draw(final Toolkit t) {
    if (model.getView() != null) {
      model.getBrowserLogFile().log(LogFile.INFO,
                                    "Starting rendering for View: "
                                    + model.getView().toString() + " using Tk:"
                                    + t.getClass());
      oldView = null;
      actualView = model.getView();
      rootState = chart.getRootNode();
      t.clearDocument();

      t.getRoot().setWidth(getWidth(actualView.
                                    getLayoutInformation(rootState))
                           + BrowserCanvas.BORDER);
      t.getRoot().setHeight(getHeight(actualView.
                                      getLayoutInformation(rootState))
                            + BrowserCanvas.BORDER);
      t.getRoot().setXPos(BrowserCanvas.BORDER);
      t.getRoot().setYPos(BrowserCanvas.BORDER);
      // Calling main drawing method
      t.getRoot().appendChild(createElements(rootState, null, t));

      if (BrowserProperties.doShowInterface()) {
        setTooltip();
      }
      t.repaintCanvas();
      t.showDocument();
    } else {
      model.getBrowserLogFile().log(LogFile.ERROR,
                                    "No view available to draw!");
    }
  }

  /**
   * Sets a tooltip for the whole canvas.
   */
  private void setTooltip() {
    if (chart != null) {
      Event e;
      StringBuffer s = new StringBuffer();
      boolean hasInSignals = false;
      boolean hasOutSignals = false;
      s.append("<html>");
      Iterator iter = chart.getInputEvents().iterator();
      if (iter.hasNext()) {
        s.append("<b>Input signals:</b>");
        s.append("<ul>");
        hasInSignals = true;
      } while (iter.hasNext()) {
        e = (Event) iter.next();
        s.append("<li>");
        s.append(e.toDeclaration());
        s.append("</li>");
      }
      if (hasInSignals) {
        s.append("</ul>");
      }
      iter = chart.getOutputEvents().iterator();
      if (iter.hasNext()) {
        s.append("<b>Output signals:</b>");
        s.append("<ul>");
        hasOutSignals = true;
      } while (iter.hasNext()) {
        e = (Event) iter.next();
```

```
        s.append("<li>");
        s.append(e.toDeclaration());
        s.append("</li>");
      }
      if (hasOutSignals) {
        s.append("</ul>");
      }
      s.append("<br></html>");
      if (hasInSignals || hasOutSignals) {
        toolkit.setTooltip(s.toString());
      } else {
        toolkit.setTooltip("No Signals");
      }
    }
  }

  /**
   * Returns the height considering if the state is collapsed.
   * @param layout NodeLayoutInformation
   * @return int height of state
   */
  private static int getHeight(final NodeLayoutInformation layout) {
    return layout.getHeight();
  }

  /**
   * Returns the width considering if the state is collapsed.
   * @param layout NodeLayoutInformation
   * @return int width of state
   */
  private static int getWidth(final NodeLayoutInformation layout) {
    return layout.getWidth();
  }

  /**
   * Main drawing function (parses recursive whole statechart).
   * @param n start Node
   * @param parent Parent SVG group
   * @param t Toolkit
   * @return SVG Group containg all childs and n
   */
  private GroupNode createElements(final kiel.dataStructure.Node n,
                                   final kiel.browser.view.rendering.Node
                                   parent,
                                   final Toolkit t) {
    GroupNode newGroup = drawNode(n, actualView, t);
    if (newGroup != null) {
      if (n instanceof kiel.dataStructure.CompositeState) {
        ArrayList subs = new ArrayList();
        if (n instanceof kiel.dataStructure.CompositeState) {
          subs.addAll(((kiel.dataStructure.CompositeState) n).getSubnodes());
        }
        NodeLayoutInformation layoutInf = actualView.getLayoutInformation(n);
        if (!((layoutInf instanceof CompositeStateLayoutInformation)
              && (((CompositeStateLayoutInformation) layoutInf).
                  isCollapsed()))) {
          // just to be sure
          if (subs != null) {
            kiel.dataStructure.Node dummy;
            for (int i = 0; i < subs.size(); i++) {
              dummy = (kiel.dataStructure.Node) subs.get(i);
              GroupNode childGroup = createElements(dummy, newGroup, t);
              if (childGroup != null) {
                newGroup.appendChild(childGroup);
              }
            }
          }
        } else {
          model.getBrowserLogFile().log(LogFile.DETAIL,
                                        "Config available but collapsed "
                                        + n.getID() + "/" + n.getName());
        }
      }
      ArrayList outTrans = (ArrayList) n.getOutgoingTransitions();
      if ((outTrans != null) && (parent != null)) {
        Edge edummy;
        for (int i = 0; i < outTrans.size(); i++) {
          edummy = (Edge) outTrans.get(i);
          if (edummy instanceof Transition
              && !((Transition) edummy).getLabel().toString().equals("")
              && !ModelHelper.canSimulateStringLabel()
              && ((Transition) edummy).getLabel() instanceof StringLabel) {
            model.disableSimulator();
            model.getBrowserLogFile().log(LogFile.ERROR,
                                          "Found Edge with unparsed Label ("
                                          + ((Transition) edummy).getLabel()
                                          + "), disabling simulation (chart: "
                                          + model.getStateChart().
                                          getRootNode()
                                          + ")");
          }
          GroupNode edgeGroup = drawEdge(edummy, actualView, t);
          if (edgeGroup != null) {
            parent.appendChild(edgeGroup);
          }
        }
      }
    }
    return newGroup;
  }

  /**
   * Takes a Node and "converts" (draws) it.
   * @param n Node to draw
   * @param t Toolkit
   * @param v View
   * @return SVG Group representing n
   */
  public static GroupNode drawNode(final kiel.dataStructure.Node n,
                                   final View v,
                                   final Toolkit t) {
    NodeLayoutInformation layoutInf = null;
    layoutInf = v.getLayoutInformation(n);

    if (layoutInf == null) {
      return null;
    }

    GroupNode g = loadTemplates(n, v, t);
    g.setId(BrowserCanvas.getGroupId(n));
```

```
    Iterator iter = g.getChildNodes().iterator();
    while (iter.hasNext()) {
      kiel.browser.view.rendering.Node e
        = (kiel.browser.view.rendering.Node) iter.next();
      if (e.getId().startsWith("rect_state")) {
        applyLayoutForRect(n, layoutInf, (RectNode) e);
        e.setTooltip(n.toString());
      } else if (e.getId().startsWith("circle_state")) {
        applyLayoutForCircle(n, layoutInf, (CircleNode) e);
        e.setTooltip(n.toString());
      } else if (e.getId().startsWith("relcirc_state")
                || e.getId().startsWith("relrect_state")) {
        applyLayoutForRelNode(n, layoutInf, e);
      } else if (e.getId().equalsIgnoreCase("line")) {
        int linePos = Properties.getCompositeStateHeaderHeight();
        int width = getWidth(layoutInf);
        if (n.getClass() == FinalANDState.class
            || n.getClass() == FinalORState.class) {
          linePos += Properties.getFinalCompositeStateOffset();
          e.setXPos(Properties.getFinalCompositeStateOffset());
          width -= Properties.getFinalCompositeStateOffset();
        }
        e.setYPos(linePos);
        ((LineNode) e).setEndYPos(linePos);
        ((LineNode) e).setEndXPos(width);
        t.getStyleChanger().changeStyle(e, lib.getStyle(lib.STYLE_LINE));
      } else if (e.getId().equalsIgnoreCase("label")) {
        applyLayoutForLabel(n, layoutInf, (TextNode) e, t);
      }
    }
    if (n instanceof State) {
      GroupNode activities = drawActivitiys((State) n,
                                            layoutInf,
                                            t);
      if (activities != null) {
        g.appendChild(activities);
      }
    }

    g.setXPos(layoutInf.getXPos());
    g.setYPos(layoutInf.getYPos());
    return g;
  }

  /**
   * @param n State
   * @param layoutInf layoutInfo actual
   * @param t Toolkit
   * @return Node
   */
  private static GroupNode drawActivitiys(final State n,
                                          final NodeLayoutInformation layoutInf,
                                          final Toolkit t) {

    if ((n.getClass() == Region.class && !n.getName().equals(""))
        || !n.getEntry().toString().equals("")
        || !n.getDoActivity().toString().equals("")
        || !n.getExit().toString().equals("")
        || !n.getBindAction().toString().equals("")
        || n.getVariables().size() > 0
        || n.getLocalEvents().size() > 0
        || (n.getInternalTransition() != null
            && n.getInternalTransition().getLabel() != null
            && !n.getInternalTransition().getLabel().toString().equals(""))) {
      GroupNode g = t.createGroupElement();
      g.setId("activities");
      int upper = Properties.getUpperOffset(n);
      int left = Properties.getLeftOffset(n);
      int header = 0;
      if (n instanceof ANDState
          || n instanceof ORState) {
        header += Properties.getCompositeStateHeaderHeight();
        if (n.getClass() == FinalANDState.class
            || n.getClass() == FinalORState.class) {
          header += Properties.getFinalCompositeStateOffset();
        }
      }
      ArrayList aList = new ArrayList();
      int nrLines = 0;
      if (n.getClass() == Region.class && !n.getName().equals("")) {
        nrLines++;
        aList.add(n.getName());
      }
      if (n.getVariables().size() > 0) {
        nrLines++;
        aList.add(Properties.getVariablesString((State) n));
      }
      if (n.getLocalEvents().size() > 0) {
        nrLines++;
        aList.add(Properties.getEventString((State) n));
      }
      if (!n.getEntry().toString().equals("")) {
        nrLines++;
        aList.add(Properties.getEntryString(n));
      }
      if (!n.getDoActivity().toString().equals("")) {
        nrLines++;
        aList.add(Properties.getDoActivityString(n));
      }
      if (!n.getExit().toString().equals("")) {
        nrLines++;
        aList.add(Properties.getExitString(n));
      }
      if (!n.getBindAction().toString().equals("")) {
        nrLines++;
        aList.add(Properties.getBindActionString(n));
      }
      if (n.getInternalTransition() != null
          && n.getInternalTransition().getLabel() != null
          && !n.getInternalTransition().getLabel().toString().equals("")) {
        nrLines++;
        aList.add(Properties.getInternalString(n));
      }

      float perLine = (float) (upper - header) / (float) nrLines;
      if (n.getClass() == Region.class) {
        LineNode linet = t.createLineElement();
        linet.setXPos(0);
        linet.setYPos(0);
        linet.setEndXPos(0);
```

```
    linet.setEndYPos(upper);
    linet.setId("region_limiter_left");
    t.getStyleChanger().changeStyle(linet, lib.getStyle(lib.STYLE_LINE));
    g.appendChild(linet);
    linet = t.createLineElement();
    linet.setXPos(getWidth(layoutInf));
    linet.setYPos(0);
    linet.setEndXPos(getWidth(layoutInf));
    linet.setEndYPos(upper);
    linet.setId("region_limiter_right");
    t.getStyleChanger().changeStyle(linet, lib.getStyle(lib.STYLE_LINE));

    g.appendChild(linet);
  }
  final int space = 3;
  for (int i = 0; i < nrLines; i++) {
    LineNode line = t.createLineElement();
    int width = getWidth(layoutInf);
    if (n.getClass() == FinalANDState.class
        || n.getClass() == FinalORState.class) {
      line.setXPos(Properties.getFinalCompositeStateOffset());
      width -= Properties.getFinalCompositeStateOffset();
    } else {
      line.setXPos(0);
    }
    line.setYPos((int) (header + (i + 1) * perLine));
    line.setEndYPos((int) (header + (i + 1) * perLine));
    line.setEndXPos(width);
    line.setId("hori_limit");
    t.getStyleChanger().changeStyle(line, lib.getStyle(lib.STYLE_LINE));
    g.appendChild(line);
    TextNode text = t.createTextElement();
    if (!ModelHelper.canSimulateStringLabel()
        && ((n.getEntry().getClass() == ActionsStringLabel.class
             && ((String) aList.get(i)).startsWith("On Entry"))
            || ((n.getExit().getClass() == ActionsStringLabel.class
                 && ((String) aList.get(i)).startsWith("On Exit")))
            || ((n.getDoActivity().getClass() == ActionsStringLabel.class
                 && ((String) aList.get(i)).startsWith("On Inside"))))
        || (((String) aList.get(i)).startsWith("Internal")
            && n.getInternalTransition().getLabel().getClass()
            == StringLabel.class)) {
      t.getStyleChanger().changeStyle(text,
                                      "font-family:" + headerFont
                                      + ";" + "font-size:"
                                      + headerFontSize + ";"
                                      + "color:"
                                      + BrowserCanvas.
                                      encodeColorForCSS(
                                        BrowserProperties.
                                        getStringLabelColor()));
    } else {
      t.getStyleChanger().changeStyle(text,
                                      "font-family:" + headerFont
                                      + ";" + "font-size:"
                                      + headerFontSize + ";");
    }
    text.setText((String) aList.get(i));
    text.setXPos(left);
    text.setYPos((int) (header + i * perLine) + 2);
//      System.out.println("Setting activity text to " + (String) aList.get(i)
//                         + " (" + left + "/" + (header + i * perLine) + ")");
      g.appendChild(text);
    }
    return g;
  }
  return null;
}

/**
 * @param edge edge to be converted to svg
 * @param v View
 * @param t Toolkit
 * @return a svg group representing a transition
 */
public static GroupNode drawEdge(final Edge edge,
                                 final View v,
                                 final Toolkit t) {
  EdgeLayoutInformation layoutInf = v.getLayoutInformation(edge);
  if (layoutInf == null) {
    return null;
  }
  GroupNode g = t.createGroupElement();
  g.setId(BrowserCanvas.getGroupId(edge));
  PathNode path = t.createPathElement();
  path.setPathElements(layoutInf.getAllPathElements());
  t.getStyleChanger().changeStyle(path, lib.getStyle(lib.STYLE_LINE));
  path.setId(edge.getID());
  path.setTooltip(edge.toString());
  PathNode pa = t.createPathElement();
  ArrayList p = new ArrayList(6);
  p.add(new MovetoPath(0, 0));
  p.add(new LinetoPath(0, 1));
  p.add(new LinetoPath(-10, 3));
  p.add(new LinetoPath(-10, -3));
  p.add(new LinetoPath(0, -1));
  p.add(new LinetoPath(0, 0));
  pa.setPathElements(p);
  t.getStyleChanger().changeStyle(pa, "fill:black");
  path.setMarkerEnd(pa);
  if (edge instanceof kiel.dataStructure.StrongAbortion) {
    CircleNode circle = t.createCircleElement();
    circle.setXPos(0);
    circle.setYPos(-3);
    circle.setWidth(6);
    circle.setHeight(6);
    t.getStyleChanger().changeStyle(circle, "fill:red;stroke:black");
    path.setMarkerStart(circle);
  } else if (edge instanceof kiel.dataStructure.NormalTermination) {
    pa = t.createPathElement();
    p = new ArrayList(4);
    p.add(new MovetoPath(0, -3));
    p.add(new LinetoPath(0, 3));
    p.add(new LinetoPath(10, 0));
    p.add(new LinetoPath(0, -3));
    pa.setPathElements(p);
    pa.setPathElements(p);
    t.getStyleChanger().changeStyle(pa, "fill:green;stroke:black");
    path.setMarkerStart(pa);
  }
```

```
    g.appendChild(path);
    // just transition have labels
    if (edge instanceof Transition) {
      drawPriorityLabel(edge, v, t, g);
      drawTransitionLabel(edge, v, t, g);
    }
    return g;
  }

  public static void drawTransitionLabel(final Edge edge,
                                         final View v,
                                         final Toolkit t,
                                         final GroupNode g) {
    // draw label
    if (((Transition) edge).getLabel() != null) {
      LabelLayoutInformation labelLayout = v.getLayoutInformation(
        ((Transition) edge).getLabel());
      if ((labelLayout != null) && (!labelLayout.isHidden())
          && ((Transition) edge).getLabel().toString().length() > 0) {
        TextNode txt = t.createTextElement();
        txt.setId(((Transition) edge).getLabel().getID());
        if (!ModelHelper.canSimulateStringLabel()
            && ((Transition) edge).getLabel() instanceof StringLabel) {
          t.getStyleChanger().changeStyle(txt, "font-family:" + labelFont
                                          + ";" + "font-size:"
                                          + labelFontSize + ";color:"
                                          + BrowserCanvas.
                                          encodeColorForCSS(
                                            BrowserProperties.
                                            getStringLabelColor()));
        } else {
          t.getStyleChanger().changeStyle(txt, "font-family:" + labelFont
                                          + ";" + "font-size:"
                                          + labelFontSize + ";");
        }
        txt.setText("" + ((Transition) edge).getLabel());
        TextLayout tl = new TextLayout(txt.getText(),
                                       Properties.getLabelFont(),
                                       new FontRenderContext(
                                         new AffineTransform(), true, true));
        txt.setXPos(labelLayout.getUpperLeft().getX());
        txt.setYPos(labelLayout.getUpperLeft().getY()
                    - (int) Math.ceil(tl.getDescent()));
        g.appendChild(txt);
      }
    }
  }

  /**
   * @param edge Edge
   * @param v View
   * @param t Toolkit
   * @param g GroupNode
   */
  public static void drawPriorityLabel(final Edge edge, final View v,
                                       final Toolkit t,
                                       final GroupNode g) {
    if ((edge.getSource().getOutgoingTransitions().size() > 1
        && ((Transition) edge).getPriority().getValue() > 0)
        || (((Transition) edge).getPriority().getValue() > 0
        && BrowserProperties.doAlwaysShowPriorities())) {
      // draw priority
      LabelLayoutInformation labelLayout =
        v.getLayoutInformation(((Transition) edge).getPriority());
      if ((labelLayout != null) && (!labelLayout.isHidden())) {
        TextNode txt = t.createTextElement();
        txt.setId(((Transition) edge).getPriority().getID());
        t.getStyleChanger().changeStyle(txt, "font-family:" + labelFont
                                        + ";" + "font-size:"
                                        + labelFontSize + ";");
        txt.setText("<" + ((Transition) edge).getPriority().getValue() + ">");
        TextLayout tl = new TextLayout(txt.getText(), Properties.getLabelFont(),
                                       new FontRenderContext(new
          AffineTransform(), true, true));
        txt.setXPos(labelLayout.getUpperLeft().getX());
        txt.setYPos(labelLayout.getUpperLeft().getY()
                    - (int) Math.ceil(tl.getDescent()));

//          RectNode rect1 = toolkit.createRectElement();
//          g.appendChild(rect1);
//          toolkit.getStyleChanger().changeStyle(rect1,
//                                  "fill:none;stroke:red;stroke-width:0.75px;");
//          rect1.setXPos(labelLayout.getUpperLeft().getX());
//          rect1.setYPos(labelLayout.getUpperLeft().getY());
//          rect1.setRx(0);
//          rect1.setRy(0);
//          rect1.setWidth(labelLayout.getWidth());
//          rect1.setHeight(labelLayout.getHeight());
        g.appendChild(txt);
      } else if (labelLayout == null) {
        Browser.getModel().getBrowserLogFile().log(LogFile.ERROR,
          "Priority label missing in Edge: "
          + edge);
      }
    }
  }

  /**
   * @param n Node
   * @param layoutInf NodeLayoutInformation
   * @param e SVGElement
   */
  private static void applyLayoutForCircle(final kiel.dataStructure.Node n,
                                           final NodeLayoutInformation
                                           layoutInf,
                                           final CircleNode e) {
    e.setId(n.getID());
    e.setWidth(layoutInf.getWidth());
    e.setHeight(layoutInf.getHeight());
  }

  /**
   * @param n Node
   * @param layoutInf NodeLayoutInformation
   * @param e SVGElement
   */
  private static void applyLayoutForRect(final kiel.dataStructure.Node n,
                                         final NodeLayoutInformation layoutInf,
```

```
                                        final RectNode e) {
    e.setId(n.getID());
    e.setWidth(getWidth(layoutInf));
    e.setHeight(getHeight(layoutInf));
  }

  /**
   *
   * @param n Node
   * @param layoutInf NodeLayoutInformation
   * @param e TextNode
   * @param t Toolkit
   */
  private static void applyLayoutForLabel(final kiel.dataStructure.Node n,
                                          final NodeLayoutInformation layoutInf,
                                          final TextNode e,
                                          final Toolkit t) {
    if (!(n instanceof PseudoState)) {
      if (n.getName() != null && n.getName().length() > 0) {
        e.setText(n.getName());
      } else {
        e.setText(null);
        // no text to draw
        return;
      }
    }

    float height = 0;
    float width = 0;
    float x;
    float y;
    String text = e.getText();

    t.getStyleChanger().changeStyle(e, "font-family:" + descFont + ";"
                                    + "font-size:" + descFontSize + ";");

    width = e.getWidth();
    height = e.getHeight();

    if (!((n instanceof kiel.dataStructure.ANDState)
          || (n instanceof kiel.dataStructure.ORState))) {
      x = ((float) layoutInf.getWidth() - width) / 2.0f;
      if (n instanceof State) {
        y = ((float) (layoutInf.getHeight()
                      - Properties.getUpperOffset((State) n)
                      - Properties.getLowerOffset((State) n))
             - height) / 2.0f;
        y += (float) Properties.getUpperOffset((State) n);
      } else {
        y = ((float) layoutInf.getHeight()
             - height) / 2.0f;
      }
      e.setXPos((int) Math.round(x));
      e.setYPos((int) Math.round(y));
    } else {
      e.setXPos((int) Math.round((layoutInf.getWidth() - width) / 2.0f));
      y = Properties.getCompositeStateHeaderHeight() - (int) height;
      if (n.getClass() == FinalANDState.class
          || n.getClass() == FinalORState.class) {
        y += Properties.getFinalCompositeStateOffset();
      }
      e.setYPos((int) Math.round(y - 1));
    }
  }

  /**
   * @param n Node to apply this layout for
   * @param layoutInf NodeLayoutInformation
   * @param e Node
   */
  private static void applyLayoutForRelNode(final kiel.dataStructure.Node n,
                                            final NodeLayoutInformation
                                            layoutInf,
                                            final kiel.browser.view.rendering.
                                            Node e) {
    final int doubled = -2;
    int newRelWidth = 0;
    int newRelHeight = 0;
    if (n.getClass() == FinalState.class
        || n.getClass() == FinalSimpleState.class) {
      newRelWidth = Properties.getFinalStateOffset();
      newRelHeight = Properties.getFinalStateOffset();
    } else if (n.getClass() == FinalANDState.class
               || n.getClass() == FinalORState.class) {
      newRelWidth = Properties.getFinalCompositeStateOffset();
      newRelHeight = Properties.getFinalCompositeStateOffset();
    }
    e.setXPos(newRelWidth);
    e.setYPos(newRelHeight);
    e.setWidth(getWidth(layoutInf) + doubled * newRelWidth);
    e.setHeight(getHeight(layoutInf) + doubled * newRelHeight);
    e.setId("relative");
  }


  /**
   *
   * @param n Node
   * @param v View
   * @param t Toolkit
   * @return GroupNode
   */
  private static GroupNode loadTemplates(final Node n, final View v,
                                         final Toolkit t) {
    GroupNode g;
    boolean collapsed = (v.getLayoutInformation(n).getClass()
                         == CompositeStateLayoutInformation.class
                         && ((CompositeStateLayoutInformation) v.
                             getLayoutInformation(n)).isCollapsed());
    if (n.getClass() == ANDState.class) {
      g = loadANDStateTemplate();
      if (!collapsed) {
        createDelimiterLines((ANDState) n, g, v, t);
      }
    } else if (n.getClass() == ORState.class) {
      g = loadORStateTemplate();
    } else if (n.getClass() == Region.class) {
      g = t.createGroupElement();
//      g.setId(n.getID());
      RectNode rect = t.createRectElement();
```

```
        rect.setRx(0);
        rect.setRy(0);
        rect.setXPos(0);
        rect.setYPos(0);
        rect.setId("rect_state");
        t.getStyleChanger().changeStyle(rect, "stroke:none");
        g.appendChild(rect);
//        if (BrowserProperties.getLogLevel() <= LogFile.DEBUG) {
//          SVGRectElement rect1 = doc.createRectElement();
//          rect1.setAttribute("x", "0");
//          rect1.setAttribute("y", "0");
//          rect1.setAttribute("width", "10");
//          rect1.setAttribute("height", "10");
//          rect1.setId("rect_state");
//          g.setId("nix");
//          g.appendChild(rect1);
//        }
      } else if (n.getClass() == InitialState.class) {
        g = loadInitialStateTemplate();
      } else if (n.getClass() == History.class) {
        g = loadHistoryStateTemplate();
      } else if (n.getClass() == DeepHistory.class) {
        g = loadDeepHistoryStateTemplate();
      } else if (n.getClass() == Choice.class
                 || n.getClass() == DynamicChoice.class) {
        g = loadChoiceStateTemplate();
      } else if (n.getClass() == Suspend.class) {
        g = loadSuspendStateTemplate();
      } else if (n.getClass() == Junction.class) {
        g = loadJunctionStateTemplate();
      } else if ((n.getClass() == FinalSimpleState.class)
                 || (n.getClass() == FinalState.class)) {
        g = loadFinalSimpleStateTemplate();
      } else if ((n.getClass() == FinalANDState.class)
                 || (n.getClass() == FinalORState.class)) {
        g = loadFinalCompositeStateTemplate();
        if (n.getClass() == FinalANDState.class && !collapsed) {
          createDelimiterLines((FinalANDState) n, g, v, t);
        }
      } else {
        g = loadSimpleStateTemplate();
      }
      return g;
    }

    /**
     * @param n ANDState  create delimiter lines for that state
     * @param g SVGGElement create lines in that SVG Group
     * @param v View
     * @param t Toolkit
     */
    private static void createDelimiterLines(final ANDState n,
                                             final GroupNode g,
                                             final View v,
                                             final Toolkit t) {
      Iterator iter = n.getDelimiterLines().iterator();
      DelimiterLine line = null;
      DelimiterLineLayoutInformation dLayout = null;
      while (iter.hasNext()) {
        line = (DelimiterLine) iter.next();
        dLayout = v.getLayoutInformation(line);
        if (dLayout != null) {
          LineNode l = t.createLineElement();
          l.setXPos(dLayout.getStart().getX());
          l.setYPos(dLayout.getStart().getY());
          l.setEndXPos(dLayout.getEnd().getX());
          l.setEndYPos(dLayout.getEnd().getY());
          l.setDashed();
          l.setId(line.getID());
          t.getStyleChanger().changeStyle(l, lib.getStyle(lib.STYLE_LINE));
          g.appendChild(l);
        }
      }
    }

    /**
     *
     * @return GraphicalObjectsLibrary
     */
    public static GraphicalObjectsLibrary getGraphicalLibrary() {
      return lib;
    }


    /**
     * @return Simple State template
     */
    private static GroupNode loadSimpleStateTemplate() {
      return lib.getTemplate(lib.SIMPLESTATE);
    }

    /**
     * @return Initial State template
     */
    private static GroupNode loadInitialStateTemplate() {
      return lib.getTemplate(lib.INITALSTATE);
    }

    /**
     * @return Final State template
     */
    private static GroupNode loadFinalSimpleStateTemplate() {
      return lib.getTemplate(lib.FINALSTATE);
    }

    /**
     * @return Final State template
     */
    private static GroupNode loadFinalCompositeStateTemplate() {
      return lib.getTemplate(lib.FINALCOMPOSITESTATE);
    }

    /**
     * @return Choice State template
     */
    private static GroupNode loadChoiceStateTemplate() {
      return lib.getTemplate(lib.CHOICESTATE);
    }

    /**
```

```
     * @return History State template
     */
    private static GroupNode loadHistoryStateTemplate() {
      return lib.getTemplate(lib.HISTORYSTATE);
    }

    /**
     * @return DeepHistory State template
     */
    private static GroupNode loadDeepHistoryStateTemplate() {
      return lib.getTemplate(lib.DEEPHISTORYSTATE);
    }

    /**
     * @return Suspend State template
     */
    private static GroupNode loadSuspendStateTemplate() {
      return lib.getTemplate(lib.SUSPENDSTATE);
    }

    /**
     * @return Suspend State template
     */
    private static GroupNode loadJunctionStateTemplate() {
      return lib.getTemplate(lib.JUNCTIONSTATE);
    }

    /**
     * @return And State template
     */
    private static GroupNode loadANDStateTemplate() {
      return lib.getTemplate(lib.ANDSTATE);
    }

    /**
     * @return Or State template
     */
    private static GroupNode loadORStateTemplate() {
      return lib.getTemplate(lib.ORSTATE);
    }
  }
```

F.5.19. MyDocumentFilter

```
package kiel.browser.view;

import java.io.IOException;
import java.io.PushbackReader;
import java.io.StringReader;

import javax.swing.ImageIcon;
import javax.swing.JTextPane;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.DocumentFilter;
import javax.swing.text.DocumentFilter.FilterBypass;
import javax.swing.text.StyledDocument;

import kiel.dataStructure.StateChart;
import kiel.browser.Browser;
import kiel.browser.model.languages.KitLanguage;
import kiel.util.DocumentReader;
import kiel.util.kit.lexer.Lexer;
import kiel.util.kit.lexer.LexerException;
import kiel.util.kit.node.EOF;
import kiel.util.kit.node.Start;
import kiel.util.kit.node.TAdd;
import kiel.util.kit.node.TBind;
import kiel.util.kit.node.TBoolean;
import kiel.util.kit.node.TChoice;
import kiel.util.kit.node.TCombine;
import kiel.util.kit.node.TComment;
import kiel.util.kit.node.TDeephistory;
import kiel.util.kit.node.TDim;
import kiel.util.kit.node.TDo;
import kiel.util.kit.node.TDouble;
import kiel.util.kit.node.TDynamicchoice;
import kiel.util.kit.node.TEdge;
import kiel.util.kit.node.TEntry;
import kiel.util.kit.node.TExit;
import kiel.util.kit.node.TFinal;
import kiel.util.kit.node.TFloat;
import kiel.util.kit.node.TFork;
import kiel.util.kit.node.THistory;
import kiel.util.kit.node.TIdentifier;
import kiel.util.kit.node.TInitial;
import kiel.util.kit.node.TInput;
import kiel.util.kit.node.TInt;
import kiel.util.kit.node.TInteger;
import kiel.util.kit.node.TInternal;
import kiel.util.kit.node.TJoin;
import kiel.util.kit.node.TJunction;
import kiel.util.kit.node.TLBrace;
import kiel.util.kit.node.TLBracket;
import kiel.util.kit.node.TLReg;
import kiel.util.kit.node.TLabel;
import kiel.util.kit.node.TLocalevent;
import kiel.util.kit.node.TLocalvariable;
import kiel.util.kit.node.TModel;
import kiel.util.kit.node.TMult;
import kiel.util.kit.node.TNormaltermination;
import kiel.util.kit.node.TNumber;
import kiel.util.kit.node.TOutput;
import kiel.util.kit.node.TParallel;
import kiel.util.kit.node.TPriority;
import kiel.util.kit.node.TRBrace;
import kiel.util.kit.node.TRBracket;
import kiel.util.kit.node.TRReg;
import kiel.util.kit.node.TStatechart;
import kiel.util.kit.node.TString;
import kiel.util.kit.node.TStrongabortion;
import kiel.util.kit.node.TSuspend;
import kiel.util.kit.node.TSync;
import kiel.util.kit.node.TType;
import kiel.util.kit.node.TVar;
import kiel.util.kit.node.TVersion;
import kiel.util.kit.node.TWeakabortion;
import kiel.util.kit.node.TWith;
import kiel.util.kit.node.Token;
import kiel.util.kit.parser.Parser;
import kiel.util.kit.parser.ParserException;
import kiel.util.languages.ILineNumber;
import kiel.util.languages.UpdateThread;
import kiel.util.main.KitToKielTranslation;
import javax.swing.JComponent;
import kiel.browser.BrowserProperties;
import javax.swing.AbstractAction;
import java.awt.event.ActionEvent;
import kiel.util.kit.node.TSuspension;
import kiel.util.kit.node.TConditional;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Filter reacts if document changes.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.9 £ last modified £Date: 2006/05/21 21:39:33 £
 */
public class MyDocumentFilter extends DocumentFilter {
  /**
   * String with changed line.
   */
  private String changedLine = null;
  /**
   * Start of line offset.
   */
  private int startOfLine;
  /**
   * Text Editor pane.
   */
```

```
  private JTextPane pane;
  /**
   * Line number component.
   */
  private ILineNumber line;
  /**
   * Update thread.
   */
  private static UpdateThread thread;
  /**
   * New chart.
   */
  private StateChart chart = null;


  /**
   * Icon for lexer error.
   */
  private static ImageIcon lexerIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/lexer.gif");
  /**
   * Icon for parser error.
   */
  private static ImageIcon parserIcon = ResourceLoader.createImageIcon(
    "kiel/browser/images/parser.gif");

  /**
   * @param p JTextPane
   * @param l ILineNumber
   * @param ut UpdateThread
   */
  public MyDocumentFilter(final JTextPane p,
                          final ILineNumber l,
                          final UpdateThread ut) {
    super();
    pane = p;
    line = l;
    thread = ut;
    thread.notifyKeyboardAction();
    pane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
                                  BrowserProperties.getKeyStroke(
                                   "Browser.KeyStroke.ForceUpdate"), "forceUp");
    pane.getActionMap().put("forceUp", new AbstractAction() {
      public void actionPerformed(final ActionEvent e) {
//        System.out.println("Foring Update");
        thread.forceUpdate();
      }
    });


  }

  /**
   * @param fb FilterBypass
   * @param offset int
   * @param string String
   * @param attr AttributeSet
   * @throws BadLocationException if error
   */
  public final void insertString(final DocumentFilter.FilterBypass fb,
                                 final int offset,
                                 final String string,
                                 final AttributeSet attr)
    throws BadLocationException {
//    System.out.println("INSERT: " + string + " off: " + offset);
    super.insertString(fb, offset, string, attr);

    highlightLine(fb, offset, string);
    accepted(fb.getDocument());
    pane.setCaretPosition(offset + 1);
    thread.notifyKeyboardAction();
  }

  /**
   * @param fb FilterBypass
   * @param offset int
   * @param length int
   * @param text String
   * @param attrs AttributeSet
   * @throws BadLocationException if error
   */
  public final void replace(final DocumentFilter.FilterBypass fb,
                            final int offset,
                            final int length,
                            final String text,
                            final AttributeSet attrs) throws
    BadLocationException {
//    System.out.println("REPLACED: " + text + " off: " + offset
//                       + " l: "
//                       + length);
    super.replace(fb, offset, length, text, attrs);
    callLexerAndParser(fb, offset, text);
    pane.setCaretPosition(offset + 1);
    thread.notifyKeyboardAction();
  }

  /**
   * @param fb FilterBypass
   * @param offset int
   * @param length int
   * @throws BadLocationException if error
   */
  public final void remove(final DocumentFilter.FilterBypass fb,
                           final int offset,
                           final int length) throws BadLocationException {
//    System.out.println("REMOVED: off: " + offset + " l: "
//                       + length);
    super.remove(fb, offset, length);
    callLexerAndParser(fb, offset, "");
    pane.setCaretPosition(offset);
    thread.notifyKeyboardAction();
  }


  /**
   * @param fb FilterBypass
   * @param offset int
   * @param text String
   * @throws BadLocationException if error
   */
```

```
  private void callLexerAndParser(final FilterBypass fb,
                                  final int offset,
                                  final String text)
    throws BadLocationException {
    highlightLine(fb, offset, text);
    if (accepted(fb.getDocument())) {
//      long start = System.currentTimeMillis();
      chart = trans.getConvertedChart();
      thread.setStatechart(chart, fb.getDocument());
//      System.out.println("Setting STATECHART time elapsed: "
//                         + (System.currentTimeMillis() - start) + " ms");
    }
  }

  /**
   * Buffer size for parser.
   */
  private static final int BUFFER = 1024;

  /**
   *  As the name says.
   */
  private KitToKielTranslation trans;

  /**
   * Test if the parser accepts the document.
   * @param doc Document
   * @return boolean
   */
  public final boolean accepted(final Document doc) {
//    long start = System.currentTimeMillis();
    Parser parser = new Parser(new Lexer(new PushbackReader(
      new DocumentReader(doc), BUFFER)));
    try {
      Start tree = parser.parse();
      trans = Browser.getKielFrame().getKitToKiel();
      tree.apply(trans.getTranslation());
    } catch (IOException ex1) {
      return false;
    } catch (LexerException ex1) {
      return false;
    } catch (ParserException ex1) {
      String message = ex1.getMessage();
      String[] answers = message.split("[\\[,\\]:]");
      int lineNr = Integer.parseInt(answers[1].trim());
      int pos = Integer.parseInt(answers[2].trim());
      final int tokenSplit = 4;
      String token = answers[tokenSplit].trim();
//      System.out.println("PARSER: Token \""
//                         + token + "\" in [" + lineNr + "," + pos
//                         + "] nr" + answers.length + " " + message);
      line.addIconOnLine(lineNr, parserIcon, ex1.getMessage());
      return false;
    }
//    System.out.println("Accept time elapsed: "
//                       + (System.currentTimeMillis() - start) + " ms");
    line.clearIcons();
    return true;
  }

  /**
   * Does the systax highlighting.
   * @param fb FilterBypass
   * @param offset int
   * @param text String
   * @throws BadLocationException if error
   */
  private void highlightLine(final FilterBypass fb,
                             final int offset,
                             final String text) throws
    BadLocationException {
    changedLine = null;
    getChangedLine(fb, offset, text.length());
    if (changedLine != null) {
      Lexer lexer = null;
      StyledDocument doc = (StyledDocument) fb.getDocument();
      lexer = new Lexer(new PushbackReader(new StringReader(
        changedLine)));
      Token t = null;
      line.removeIconOnLine(getLine(doc, startOfLine));
      int error = 0;
      do {
        try {
          t = lexer.next();
          if (t == null) {
            System.err.println("ERROR t is NULL");
            break;
          }
//          System.err.println("Working on Token " + t);
          if (t.getClass() == TIdentifier.class) {
            super.replace(fb,
                          t.getPos() + startOfLine - 1,
                          t.getText().length(),
                          t.getText(), KitLanguage.IDS);
          } else if (t.getClass() == TStatechart.class
                     || t.getClass() == TAdd.class
                     || t.getClass() == TBind.class
                     || t.getClass() == TBoolean.class
                     || t.getClass() == TChoice.class
                     || t.getClass() == TCombine.class
                     || t.getClass() == TConditional.class
                     || t.getClass() == TDeephistory.class
                     || t.getClass() == TDo.class
                     || t.getClass() == TDouble.class
                     || t.getClass() == TDynamicchoice.class
                     || t.getClass() == TEntry.class
                     || t.getClass() == TExit.class
                     || t.getClass() == TFinal.class
                     || t.getClass() == TFloat.class
                     || t.getClass() == TFork.class
                     || t.getClass() == THistory.class
                     || t.getClass() == TInitial.class
                     || t.getClass() == TInput.class
                     || t.getClass() == TInteger.class
                     || t.getClass() == TInternal.class
                     || t.getClass() == TJoin.class
                     || t.getClass() == TJunction.class
                     || t.getClass() == TLabel.class
                     || t.getClass() == TLocalevent.class
                     || t.getClass() == TLocalvariable.class
```

```
                || t.getClass() == TModel.class
                || t.getClass() == TMult.class
                || t.getClass() == TNormaltermination.class
                || t.getClass() == TOutput.class
                || t.getClass() == TPriority.class
                || t.getClass() == TStrongabortion.class
                || t.getClass() == TSuspension.class
360             || t.getClass() == TSuspend.class
                || t.getClass() == TSync.class
                || t.getClass() == TType.class
                || t.getClass() == TVar.class
                || t.getClass() == TVersion.class
                || t.getClass() == TWeakabortion.class
                || t.getClass() == TWith.class) {
              super.replace(fb,
                            t.getPos() + startOfLine - 1,
                            t.getText().length(),
370                         t.getText(), KitLanguage.KEYWORD);
            } else if (t.getClass() == TComment.class) {
              super.replace(fb,
                            t.getPos() + startOfLine - 1,
                            t.getText().length(),
                            t.getText(), KitLanguage.COMMENT);
            } else if (t.getClass() == TString.class) {
              super.replace(fb,
                            t.getPos() + startOfLine - 1,
                            t.getText().length(),
380                         t.getText(), KitLanguage.STRING);
            } else if (t.getClass() == TEdge.class
                || t.getClass() == TLBrace.class
                || t.getClass() == TLBracket.class
                || t.getClass() == TLReg.class
                || t.getClass() == TParallel.class
                || t.getClass() == TRBrace.class
                || t.getClass() == TRBracket.class
                || t.getClass() == TRReg.class) {
              super.replace(fb,
390                         t.getPos() + startOfLine - 1,
                            t.getText().length(),
                            t.getText(), KitLanguage.STRUCTURE);
            } else if (t.getClass() == TNumber.class
                || t.getClass() == TInt.class
                || t.getClass() == TDim.class) {
              super.replace(fb,
                            t.getPos() + startOfLine - 1,
                            t.getText().length(),
400                         t.getText(), KitLanguage.NUMBER);
            } else {
              super.replace(fb,
                            t.getPos() + startOfLine - 1,
                            t.getText().length(),
                            t.getText(), KitLanguage.STD);
            }
          } catch (IOException ex1) {
            t = null;
          } catch (LexerException ex1) {
            String message = ex1.getMessage();
410         String[] answers = message.split("[\\[,\\]:]");
            //  int line = Integer.parseInt(answers[1].trim());
            int pos = Integer.parseInt(answers[2].trim());
            final int tokenSplit = 4;
            String token = answers[tokenSplit].trim();
            if (token.equals("")) {
              token = ",";
            }
      //      System.out.println("Found wrong token \""
      //                         + token + "\" in [" + (pos + startOfLine - 1)
420   //                         + "] " + ex1.getMessage());
            super.replace(fb, pos + startOfLine - 1,
                          token.length(), token, KitLanguage.LEXERROR);
            line.addIconOnLine(getLine(doc, pos + startOfLine - 1),
                               lexerIcon, ex1.getMessage());
            changedLine = changedLine.substring(pos + token.length() - 1);
            lexer = new Lexer(new PushbackReader(new StringReader(
              changedLine)));
            startOfLine = pos + startOfLine + token.length() - 1;
      //      System.out.println("Setting new lexer with text " + changedLine
430   //                         + " offs:" + startOfLine);
            error++;
            final int maxErrors = 10;
            if (error > maxErrors) {
              System.exit(-1);
            }
          }
        } while (t != null && t.getClass() != EOF.class);
      }
      //    System.out.println("--- END ----");
440   }

      /**
       * Gets a line from a the given offset.
       * @param doc Document
       * @param offset int
       * @return int
       * @throws BadLocationException if error
       */
      private static int getLine(final Document doc, final int offset) throws
450     BadLocationException {
        final int searchframe = 10;

        int line = 1;
        int start = -searchframe;
        int newLinePos = -1;
        int tested = 0;
        final int end = doc.getEndPosition().getOffset();

      //    System.out.println("CALLIN getline ");
460     do {
          start += searchframe;
          String text = null;
          if (start + searchframe > end) {
            text = doc.getText(start, end - start);
          } else {
            text = doc.getText(start, searchframe);
          }
      //    System.out.println("Testing Text: " + text);
          newLinePos = -1;
470       do {
            newLinePos = text.indexOf('\n', newLinePos + 1);
      //      System.out.println("Getting IndexOf newline: " + newLinePos);
```

```
          if (newLinePos != -1) {
            line++;
            tested = start + newLinePos;
//            System.out.println("Got Line " + line + " width off " + tested);
            if (tested >= offset) {
              line--;
            }
          }
        } while (newLinePos != -1);
      } while (tested < offset);
//    System.out.println("RESUKT: Getting line " + line + " from offset "
//                       + offset);
    return line;
  }

  /**
   * Searches fpr a newline.
   * @param searchframe int
   * @param rest int
   * @param doc Document
   * @param start int
   * @return int
   * @throws BadLocationException if errors
   */
  private static int forwardSearchNewline(final int searchframe,
                                          final int rest,
                                          final Document doc,
                                          final int start) throws
    BadLocationException {
    int localStart = start;
    int newlinePos = -1;
    String search;
    do {
      if (rest >= searchframe) {
        search = doc.getText(localStart, searchframe);
      } else {
        search = doc.getText(localStart, rest);
      }
//            System.out.println("Searching for newline in " + search);
      newlinePos = search.indexOf('\n');
      if (newlinePos < 0) {
        localStart += search.length();
      }
    } while (newlinePos < 0);
    return newlinePos + localStart;
  }

  /**
   * Searches for a newline (backwards).
   * @param searchframe int
   * @param doc Document
   * @param start int
   * @return int
   * @throws BadLocationException if error
   */
  private static int backwardSearchNewline(final int searchframe,
                                           final Document doc,
                                           final int start) throws
    BadLocationException {
    int localStart = start;
    int newlinePos = -1;
    String search;
    do {
      if (localStart >= searchframe) {
        search = doc.getText(localStart - searchframe, searchframe);
      } else {
        search = doc.getText(0, localStart);
      }
      //  System.out.println("Searching for newline in " + search);
      newlinePos = search.lastIndexOf('\n');
      if (newlinePos < 0) {
        localStart -= search.length();
        if (search.length() < searchframe) {
          return 0;
        }
      }
    } while (newlinePos < 0);
    return newlinePos + localStart - search.length() + 1;
  }

  /**
   *
   * @param fb FilterBypass
   * @param offs int
   * @param length int
   */
  public final void getChangedLine(final FilterBypass fb,
                                   final int offs,
                                   final int length) {
    final int search = 10;
    int rest = fb.getDocument().getEndPosition().getOffset()
               - (offs + length);
    int before = offs;
    Document doc = fb.getDocument();
    int start = offs + length;
    try {
//            System.out.println("FORWARD SEARCH:");
      int newlinePos2 = forwardSearchNewline(search, rest,
                                             doc, start);
//            System.out.println("BACKWARD SEARCH:");
      int newlinePos1 = backwardSearchNewline(search, doc,
                                              start);
      changedLine = doc.getText(newlinePos1, newlinePos2 - newlinePos1);
      startOfLine = newlinePos1;
//            System.out.println("RESULT:");
//      System.out.println("Changed Line: " + changedLine
//                         + " ("
//                         + changedLine.length() + ") start: "
//                         + startOfLine);
      if (changedLine != null && changedLine.length() == 0) {
        changedLine = null;
        return;
      }
    } catch (BadLocationException ex) {
      ex.printStackTrace();
      changedLine = null;
    }
    changedLine += '\n';
  }
}
```

F.6. kiel.browser.view.rendering

F.6.1. Node

```
package kiel.browser.view.rendering;

import java.util.Collection;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Interface for a node.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.2 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public interface Node {
  /**
   *
   * @param childGroup Node
   */
  void appendChild(Node childGroup);
  /**
   * Returns a list of all children.
   * @return Collection
   */
  Collection getChildNodes();
  /**
   * @return String
   */
  String getId();
  /**
   * Id should be unique. But this is not tested.
   * @param anId String
   */
  void setId(String anId);
  /**
   * @param node Node
   */
  void removeChild(Node node);
  /**
   * @param width int
   */
  void setWidth(int width);
  /**
   * @param height int
   */
  void setHeight(int height);
  /**
   * @param x int
   */
  void setXPos(int x);
  /**
   * @param y int
   */
  void setYPos(int y);
  /**
   * @return int
   */
  int getWidth();
  /**
   * @return int
   */
  int getHeight();
  /**
   * @return int
   */
  int getX();
  /**
   * @return int
   */
  int getY();
  /**
   * @return String
   */
  String getStyle();
  /**
   * @return Node
   */
  Node getParent();
  /**
   * @param aParent Node
   */
  void setParent(Node aParent);
  /**
   * Set Tooltip.
   * @param str String
   */
  void setTooltip(String str);
  /**
   * Animate height.
   * @param newHeight int
   */
  void animateHeightTo(int newHeight);
  /**
   * Animate width.
   * @param newWidth int
   */
  void animateWidthTo(int newWidth);
  /**
   * Animate to given position.
   * @param newX int
   * @param newY int
   */
  void animatePositionTo(int newX, int newY);
}
```

F.6.2. GroupNode

```
package kiel.browser.view.rendering;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Describes an interface for a grouping node.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public interface GroupNode extends Node {
  /**
   * Animated remove of this node and his children.
   * @param removeAfterAnimation boolean remove from tree after disapearing.
   */
  void animatedRemove(boolean removeAfterAnimation);
  /**
   * Animated insertion of this node and its children.
   */
  void animatedInsert();
}
```

F.6.3. PathNode

```
package kiel.browser.view.rendering;

import java.util.ArrayList;
import java.util.Collection;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.2 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public interface PathNode extends Node {
  /**
   * @param allPathElements Collection
   */
  void setPathElements(Collection allPathElements);

  /**
   * setMarkerEnd.
   *
   * @param pa Node
   */
  void setMarkerEnd(Node pa);
  /**
   * setMarkerEnd.
   *
   * @param pa Node
   */
  void setMarkerStart(Node pa);

  /**
   * Morph from one path to another.
   * @param sourcePath ArrayList
   * @param targetPath ArrayList
   * @param source Node
   * @param target Node
   */
  void morphPath(ArrayList sourcePath,
                 ArrayList targetPath,
                 Node source,
                 Node target);
}
```

F.6.4. CircleNode

```
package kiel.browser.view.rendering;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Descripes an interface for a circle.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public interface CircleNode extends Node {
}
```

F.6.5. RectNode

```
package kiel.browser.view.rendering;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/02/08 07:07:50 $
 */
public interface RectNode extends Node {
  /**
   *
   * @param rx int
   */
  void setRx(int rx);
  /**
   *
   * @param ry int
   */
  void setRy(int ry);
  /**
   *
   * @return int
   */
  int getRx();
  /**
   *
   * @return int
   */
  int getRy();
}
```

F.6.6. TextNode

```
package kiel.browser.view.rendering;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Node that shows text.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.2 £ last modified £Date: 2006/02/08 07:07:50 £
 */

public interface TextNode extends Node {
  /**
   *
   * @param text String
   */
  void setText(String text);

  /**
   * @return String
   */
  String getText();
}
```

F.6.7. LineNode

```
package kiel.browser.view.rendering;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Interface for a line.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version $Revision: 1.3 $ last modified $Date: 2006/02/08 07:07:49 $
 */
public interface LineNode extends Node {
  /**
   * @param x int End position x.
   */
  void setEndXPos(int x);
  /**
   * @param y int End position y.
   */
  void setEndYPos(int y);

  /**
   * @return int
   */
  int getEndY();
  /**
   * @return int
   */
  int getEndX();

  /**
   * Animate line to the given coords.
   * @param newX int
   * @param newY int
   * @param newX2 int
   * @param newY2 int
   */
  void animateLine(int newX, int newY, int newX2, int newY2);

  /**
   * Draw a dashed line.
   */
  void setDashed();
}
```

F.6.8. Toolkit

```
package kiel.browser.view.rendering;

import java.awt.Dimension;

import javax.swing.JComponent;
import javax.swing.text.Document;
import java.awt.Rectangle;
import java.awt.Graphics;


/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £ last modified £Date: 2006/03/26 20:21:11 £
 */
public interface Toolkit {
  /**
   *
   * @return JComponent
   */
  JComponent getComponent();

  /**
   *
   * @return JComponent
   */
  JComponent getCanvas();

  /**
   * redraw.
   */
  void repaintCanvas();

  /**
   *
   * @return Document
   */
  Document getDocument();

  /**
   * start animation.
   */
  void startPlayback();

  /**
   *
   * @return StyleChanger
   */
  StyleChanger getStyleChanger();

  /**
   * Clears canvas.
   */
  void clearDocument();

  /**
   * Creates a textual form of the drawing.
   */
  void showDocument();

  /**
   *
   * @return Rectangle of drawing.
   */
  Rectangle getCanvasBounds();

  /**
   * Tooltip for whole canvas.
   * @param tooltip String
   */
  void setTooltip(final String tooltip);

  /**
   * Drawing root.
   * @return Node
   */
  Node getRoot();

  /**
   * Search for element in canvas.
   * @param string String
   * @return Node
   */
  Node getElementById(final String string);

  /**
   *
   * @return PathNode
   */
  PathNode createPathElement();

  /**
   *
   * @return LineNode
   */
  LineNode createLineElement();

  /**
   *
   * @return CircleNode
   */
  CircleNode createCircleElement();

  /**
   *
```

```
     * @return RectNode
     */
    RectNode createRectElement();

    /**
     * Creates new Groupnode.
     * @return GroupNode
     */
120 GroupNode createGroupElement();

    /**
     * Creates new Textnode.
     * @return TextNode
     */
    TextNode createTextElement();

    /**
130  * Zoom to canvas size.
     */
    void fitToScreen();

    /**
     * Resets zoom.
     */
    void resetZoom();

    /**
140  * Toggles the always fit flag.
     * @param alwaysFitToScreen boolean
     */
    void setAlwaysFitToScreen(boolean alwaysFitToScreen);

    /**
     *  Request focos on canvas.
     */
    void requestFocus();

150 /**
     *  Prints the canvas.
     * @param g Graphics
     */
    void printCanvas(Graphics g);
}
```

F.6.9. StyleChanger

```
package kiel.browser.view.rendering;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Interface for style changer method.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.1 £ last modified £Date: 2006/02/08 10:57:27 £
 */
public interface StyleChanger {
  /**
   * @param node SVGElement to change style for
   * @param cssString String representing new style
   */
  void changeStyle(final Node node, final String cssString);
}
```

F.7. kiel.browser.view.piccolo

F.7.1. PiccoloToolkit

```
package kiel.browser.view.piccolo;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.geom.Point2D;
import java.util.Hashtable;
import java.util.Iterator;

import javax.swing.JComponent;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.Document;

import kiel.dataStructure.Edge;
import edu.umd.cs.piccolo.PCamera;
import edu.umd.cs.piccolo.PCanvas;
import edu.umd.cs.piccolo.PNode;
import edu.umd.cs.piccolo.event.PBasicInputEventHandler;
import edu.umd.cs.piccolo.event.PInputEvent;
import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.nodes.PText;
import edu.umd.cs.piccolo.util.PAffineTransform;
import edu.umd.cs.piccolo.util.PBounds;
import edu.umd.cs.piccolox.swing.PScrollPane;
import kiel.browser.model.BrowserModel;
import kiel.browser.view.BrowserCanvas;
import kiel.browser.view.rendering.StyleChanger;
import kiel.browser.view.rendering.CircleNode;
import kiel.browser.view.rendering.GroupNode;
import kiel.browser.view.rendering.LineNode;
import kiel.browser.view.rendering.Node;
import kiel.browser.view.rendering.PathNode;
import kiel.browser.view.rendering.RectNode;
import kiel.browser.view.rendering.TextNode;
import kiel.browser.view.rendering.Toolkit;
import kiel.browser.BrowserProperties;
import kiel.graphicalInformations.NodeLayoutInformation;
import edu.umd.cs.piccolox.event.PStyledTextEventHandler;
import edu.umd.cs.piccolo.event.PDragEventHandler;
import java.awt.Rectangle;
import java.awt.Graphics;
import javax.swing.JFrame;
import edu.umd.cs.piccolo.util.PPaintContext;
import java.awt.Graphics2D;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Toolkit implementation for the piccolo toolkit.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.18 £ last modified £Date: 2006/05/21 21:39:33 £
 */
public class PiccoloToolkit implements Toolkit {
  /**
   *  Canvas.
   */
  private PCanvas canvas;
  /**
   * Scroll pane.
   */
  private PScrollPane scroll;
  /**
   * root.
   */
  private PicGroup root;
  /**
   * Change the style.
   */
  private static PicStyleCanger changer = new PicStyleCanger();
  /**
   * All elements.
   */
  private static Hashtable elements;
  /**
   * Data Model.
   */
  private BrowserModel model;
  /**
   * Tooltip Node.
   */
  private PText tooltipNode = new PText();
  /**
   * Flag.
   */
  private boolean alwaysFitToScreen = false;

  /**
   * Event handler for picking nodes.
   */
  private PBasicInputEventHandler eventHandler = new PBasicInputEventHandler() {
    private boolean isState1 = true;

    public void mouseMoved(final PInputEvent event) {
      updateToolTip(event);
    }

    public void mouseDragged(final PInputEvent event) {
```

```
      updateToolTip(event);
    }

    public void mousePressed(final PInputEvent event) {
      PNode node = updateToolTip(event);
      boolean found = false;
      if (node != null) {
        Iterator iter = elements.values().iterator();
        while (iter.hasNext() && !found) {
          PicNode p = (PicNode) iter.next();
          if (p.getPNode().equals(node)) {
            found = true;
            Object o = model.getStateChart().getObjectFromID(p.getId());
            if (o != null) {
              model.setMode(BrowserModel.EDIT);
              if (o instanceof Edge) {
                if (model.getSelectedTransition() != null
                    && model.getSelectedTransition().equals(o)) {
                  model.setFocus(model.getSelectedStates()[0],
                                 model.getSelectedStates()[1], null);
                } else {
                  model.setFocus(model.getSelectedStates()[0],
                                 model.getSelectedStates()[1],
                                 (Edge) o);
                }
              } else {
                kiel.dataStructure.Node[] nodes = model.getSelectedStates();
                if (nodes[0] == o) {
//                  System.out.println("Removing Selektion on Node1 " + o);
                  nodes[0] = null;
                } else if (nodes[1] == o) {
//                  System.out.println("Removing Selektion on Node2 " + o);
                  nodes[1] = null;
                } else if (isState1) {
//                  System.out.println("Adding Selektion on Node1 " + o);
                  nodes[0] = (kiel.dataStructure.Node) o;
                  isState1 = !isState1;
                } else {
//                  System.out.println("Adding Selektion on Node2 " + o);
                  nodes[1] = (kiel.dataStructure.Node) o;
                  isState1 = !isState1;
                }
                model.setFocus(nodes[0], nodes[1],
                               model.getSelectedTransition());
              }
            }
          }
        }
      }
    }


    public PNode updateToolTip(final PInputEvent event) {
      PNode n = event.getInputManager().getMouseOver().getPickedNode();
//      System.out.println(event.getPickedNode());
      PNode r = null;
      if (!parentInvisible(n)) {
        String tooltipString = (String) n.getClientProperty("tooltip");
//        System.out.println("Tooltip client: " + tooltipString);
        Point2D p = event.getCanvasPosition();
        event.getPath().canvasToLocal(p, canvas.getCamera());
        final int offset = 8;

        if (BrowserProperties.doShowTooltip()) {
          tooltipNode.setText(tooltipString);
          tooltipNode.setOffset(p.getX() + offset,
                                p.getY() - offset);
        }
        r = n;
      }
      return r;
    }
  };


  /**
   *
   * @param aModel BrowserModel
   */
  public PiccoloToolkit(final BrowserModel aModel) {
    model = aModel;
    canvas = new PCanvas();
    // canvas.setDoubleBuffered(true);
    root = new PicGroup(new PNode()) {
      public void setHeight(final int height) {
      }

      public void setWidth(final int width) {
      }
    };
    canvas.getLayer().addChild(root.getPNode());
    elements = new Hashtable();
    scroll = new PScrollPane(canvas);
    canvas.getCamera().addInputEventListener(eventHandler);
    scroll.addComponentListener(new ComponentListener() {
      public void componentHidden(ComponentEvent e) {
      }
      public void componentMoved(ComponentEvent e) {
      }
      public void componentResized(ComponentEvent e) {
        if (alwaysFitToScreen) {
          fitToScreen();
        }
      }
      public void componentShown(ComponentEvent e) {
      }
    });

//    canvas.removeInputEventListener(canvas.getPanEventHandler());
//    canvas.addInputEventListener(new PDragEventHandler());
  }

  /**
   * clearDocument.
   */
  public final void clearDocument() {
    canvas.getLayer().removeAllChildren();
    root = new PicGroup(new PNode()) {
      public void setHeight(final int height) {
```

```
    }

    public void setWidth(final int width) {
    }
  };
  root.setId("RootGroup");
  canvas.getLayer().addChild(root.getPNode());
  elements.clear();

  tooltipNode.setPickable(false);
  tooltipNode.setPaint(BrowserProperties.getNodeMarkColor());
  tooltipNode.setTextPaint(Color.BLACK);
  canvas.getCamera().addChild(tooltipNode);
}

/**
 *
 * @param n PNode
 * @return boolean
 */
private boolean parentInvisible(final PNode n) {
  if (n == root.getPNode()) {
    return false;
  }
  if (!n.getVisible()) {
    return true;
  } else {
    PNode parent = n.getParent();
    if (parent != null) {
      return parentInvisible(parent);
    }
    return false;
  }
}

/**
 * createCircleElement.
 *
 * @return CircleNode
 */
public final CircleNode createCircleElement() {
  return new PicCircle(new PPath());
}

/**
 * createGroupElement.
 *
 * @return GroupNode
 */
public final GroupNode createGroupElement() {
  return new PicGroup(new PNode());
}

/**
 * createLineElement.
 *
 * @return LineNode
 */
public final LineNode createLineElement() {
  return new PicLine(new PPath());
}

/**
 * createPathElement.
 *
 * @return PathNode
 */
public final PathNode createPathElement() {
  return new PicPath(new PPath());
}

/**
 * createRectElement.
 *
 * @return RectNode
 */
public final RectNode createRectElement() {
  return new PicRect(new PPath());
}

/**
 * createTextElement.
 *
 * @return TextNode
 */
public final TextNode createTextElement() {
  return new PicText(new PText(""));
}

/**
 * getComponent.
 *
 * @return Component
 */
public final JComponent getComponent() {
  return scroll;
}

/**
 * getCanvasDimension.
 *
 * @return Dimension
 */
public final Rectangle getCanvasBounds() {
  PCamera cam = canvas.getCamera();
//    System.out.println("View: " + cam.getViewBounds());
//    System.out.println("View: " + cam.getViewBounds().getMaxX() + ","
//                        + cam.getViewBounds().getMaxY());
//    System.out.println("Global trans: " + cam.getGlobalTranslation());
//    System.out.println("Global: " + cam.getGlobalBounds());
//    System.out.println("-->Full: " + root.getPNode().getFullBounds());
//    System.out.println("nothing: " + cam.getBounds());
//    System.out.println("scroll: " + scroll.getBounds());
  double scale = cam.getViewScale();
  Rectangle r = new Rectangle((int) Math.round(cam.getViewBounds().getMinX()),
                              (int) Math.round(cam.getViewBounds().getMinY()),
                              (int) root.getPNode().getFullBounds().width,
                              (int) root.getPNode().getFullBounds().height);
  return r;
//    return viewBounds.getBounds();
```

```
  }

  /**
   * getDocument.
   *
   * @return Document null because has no document representation
   */
  public final Document getDocument() {
    Document doc = new DefaultStyledDocument();
    addToDocument(root, doc);
    return doc;
  }

  /**
   *
   */
  private int off = 0;
  /**
   * Recursion showing by spaces.
   */
  private String space = "";

  /**
   * @param n PicNode
   * @param doc Document
   */
  private void addToDocument(final PicNode n, final Document doc) {
    String text = space + n.getId() + ":␣" + n.getPNode().getChildrenCount()
                  + "␣vs␣" + n.getChildNodes().size();
    space += "␣";
    try {
      doc.insertString(doc.getLength(), text + "\n", null);
    } catch (BadLocationException ex) {
      System.err.println("Exception␣:␣" + ex);
    }
    if (n.getChildNodes().size() > 0) {
      Iterator iter = n.getChildNodes().iterator();
      while (iter.hasNext()) {
        addToDocument((PicNode) iter.next(), doc);
      }
    }
    space = space.substring(1);
  }

  /**
   * getElementById.
   *
   * @param string String
   * @return Node
   */
  public final Node getElementById(final String string) {
    return (Node) elements.get(string);
  }

  /**
   * getRoot.
   *
   * @return Node
   */
  public final Node getRoot() {
    return root;
  }

  /**
   * getStyleChanger.
   *
   * @return StyleChanger
   */
  public final StyleChanger getStyleChanger() {
    return changer;
  }

  /**
   * repaintCanvas.
   */
  public final void repaintCanvas() {
    root.getPNode().invalidatePaint();
    root.getPNode().validateFullPaint();
  }

  /**
   * setTooltip.
   *
   * @param tooltip String
   */
  public final void setTooltip(final String tooltip) {
    canvas.setToolTipText(tooltip);
  }

  /**
   * showDocument.
   */
  public final void showDocument() {
    if (alwaysFitToScreen) {
      fitToScreen();
    }
  }


  /**
   * startPlayback.
   */
  public final void startPlayback() {
    // starts alone after setting animations
    //    canvas.setDoubleBuffered(true);
    if (alwaysFitToScreen) {
      fitToScreen();
    }
  }

  /**
   * @param node Node
   * @param oldId String
   */
  public static void updateListOfNodes(final Node node, final String oldId) {
    elements.remove(oldId);
    if (node != null) {
      elements.put(node.getId(), node);
    }
  }
```

```
    /**
     * Fits drawing into canvas.
     */
    public final void fitToScreen() {
      PCamera cam = canvas.getCamera();
      PBounds viewBounds = cam.getViewBounds();
      PAffineTransform newTransform = cam.getViewTransform();
      newTransform.translate(viewBounds.getMinX(), viewBounds.getMinY());
      NodeLayoutInformation layout = model.getView().getLayoutInformation(
        model.getStateChart().getRootNode());
      double chartWidth = layout.getWidth()
                + 2 * BrowserCanvas.BORDER;
      double chartHeight = layout.getHeight()
                + 2 * BrowserCanvas.BORDER;

      double s = Math.min(viewBounds.getWidth()
                / chartWidth,
                viewBounds.getHeight()
                / chartHeight);

      newTransform.scaleAboutPoint(s, cam.getGlobalFullBounds().getX(),
                cam.getGlobalFullBounds().getY());
      cam.animateViewToTransform(newTransform,
                BrowserProperties.getAnimationDuration());
    }

    /**
     * Set zoom back to 1:1.
     */
    public final void resetZoom() {
      PCamera cam = canvas.getCamera();
      PBounds viewBounds = cam.getViewBounds();
      PAffineTransform newTransform = cam.getViewTransform();
//      NodeLayoutInformation layout = model.getView().getLayoutInformation(
//        model.getStateChart().getRootNode());
//      double chartWidth = layout.getWidth()
//                + 2 * BrowserCanvas.BORDER;
//      double chartHeight = layout.getHeight()
//                + 2 * BrowserCanvas.BORDER;
//      double s = 1 / cam.getViewScale();

      newTransform.translate(viewBounds.getMinX(), viewBounds.getMinY());
      newTransform.scaleAboutPoint(s, cam.getGlobalFullBounds().getX(),
                cam.getGlobalFullBounds().getY());
      cam.animateViewToTransform(newTransform,
        BrowserProperties.getAnimationDuration());
    }

    /**
     * @return JComponent
     */
    public final JComponent getCanvas() {
      return canvas;
    }

    /**
     * @param always boolean
     */
    public final void setAlwaysFitToScreen(final boolean always) {
      alwaysFitToScreen = always;
    }

    /**
     *  Request focus on canvas.
     */
    public final void requestFocus() {
      canvas.requestFocus();
    }

    /**
     *
     * @param g Graphics
     */
    public void printCanvas(Graphics g) {
      PCanvas pc = new PCanvas();
      PNode node = (PNode) root.getPNode();
      pc.getLayer().addChild(node);
      pc.setBounds(0, 0, (int) Math.round(root.getPNode().getFullBounds().width)
                + 2 * BrowserCanvas.BORDER,
                (int) Math.round(root.getPNode().getFullBounds().height)
                + 2 * BrowserCanvas.BORDER);
      node.invalidatePaint();
      node.validateFullPaint();
      pc.invalidate();
      PBounds viewBounds = pc.getCamera().getViewBounds();
      pc.paint(g);
      canvas.getLayer().removeAllChildren();
      canvas.getLayer().addChild(node);
      repaintCanvas();
    }
  }
```

F.7.2. PicNode

```
package kiel.browser.view.piccolo;

import java.awt.BasicStroke;
import java.util.ArrayList;
import java.util.Collection;

import edu.umd.cs.piccolo.PNode;
import edu.umd.cs.piccolo.activities.PActivity;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.Node;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Implements a Node for the Piccolo Toolkit.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.5 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public class PicNode implements Node {
  /**
   * The Piccolo node.
   */
  private PNode node;
  /**
   * All Children.
   */
  private ArrayList childs;
  /**
   * Unique id.
   */
  private String id;
  /**
   * Style.
   */
  private String style;
  /**
   * Parent node.
   */
  private Node parent;
  /**
   * New position, new width, new height.
   */
  private int nh, nw, nx, ny;
  /**
   *  New width new height animation.
   */
  private PActivity nwnh = null;

  /**
   *
   * @param aNode PNode
   */
  public PicNode(final PNode aNode) {
    node = aNode;
    childs = new ArrayList();
    id = "";
    style = "";
    parent = null;
    nh = 0;
    nw = 0;
    nx = 0;
    ny = 0;
  }

  /**
   *
   * @return BasicStroke
   */
  public static final BasicStroke getDefaultStroke() {
    final float lineWidth = 1.26f;
    int lineCap = BasicStroke.CAP_BUTT;
    int lineJoin = BasicStroke.JOIN_MITER;
    final float miterLimit = 4f;
    float[] dashArray = null;
    float dashOffset = 0f;
    BasicStroke stroke = new BasicStroke(lineWidth, lineCap, lineJoin,
                                         miterLimit, dashArray, dashOffset);
    return stroke;
  }

  /**
   *
   * @return PNode
   */
  public final PNode getPNode() {
    return node;
  }

  /**
   * appendChild.
   *
   * @param childGroup Node
   */
  public final void appendChild(final Node childGroup) {
    node.addChild(((PicNode) childGroup).getPNode());
    childs.add(childGroup);
    childGroup.setParent(this);
  }

  /**
   * getChildNodes.
   *
   * @return Collection
   */
  public final Collection getChildNodes() {
    return childs;
  }
```

```
    /**
     * getId.
     *
     * @return String
     */
    public final String getId() {
      return id;
    }

    /**
     * removeChild.
     *
     * @param child Node
     */
    public final void removeChild(final Node child) {
      childs.remove(child);
      PiccoloToolkit.updateListOfNodes(null, child.getId());
      if (((PicNode) child).getPNode().isDescendentOfRoot()
          && ((PicNode) child).getPNode().isDescendentOf(node)) {
        node.removeChild(((PicNode) child).getPNode());
      }
    }

    /**
     * setHeight.
     *
     * @param height int
     */
    public void setHeight(final int height) {
    //    System.out.println("" + getId() + ": Changing Height to h: " + height);
      nh = height;
      node.setHeight(height);
    }

    /**
     * setId.
     *
     * @param anId String
     */
    public final void setId(final String anId) {
      String old = id;
      id = anId;
      PiccoloToolkit.updateListOfNodes(this, old);
    }

    /**
     * setWidth.
     *
     * @param width int
     */
    public void setWidth(final int width) {
      nw = width;
      node.setWidth(width);
    }

    /**
     * setXPos.
     *
     * @param x int
     */
    public void setXPos(final int x) {
      nx = x;
      node.setOffset(x, ny);
    }

    /**
     * setYPos.
     *
     * @param y int
     */
    public void setYPos(final int y) {
      ny = y;
      node.setOffset(nx, y);
    }

    public final void setStyle(final String aCSSString) {
      style = aCSSString;
    }

    /**
     *
     * @return String
     */
    public final String getStyle() {
      return style;
    }

    /**
     * @return Node
     */
    public final Node getParent() {
      return parent;
    }

    /**
     * @param aParent Node
     */
    public final void setParent(final Node aParent) {
      parent = aParent;
    }

    /**
     * @return int
     */
    public final int getWidth() {
      return (int) Math.floor(node.getWidth());
    }

    /**
     * @return int
     */
    public final int getHeight() {
      return (int) Math.floor(node.getHeight());
    }
```

```
    /**
     * @return int
     */
    public int getX() {
      return (int) node.getX();
    }

    /**
     * @return int
     */
    public int getY() {
      return (int) node.getY();
    }

    /**
     *
     * @param newHeight int
     */
    public void animateHeightTo(final int newHeight) {
      nh = newHeight;
      updateAnimation();
    }

    /**
     * @param newWidth int
     */
    public void animateWidthTo(final int newWidth) {
      nw = newWidth;
      updateAnimation();
    }

    /**
     *  Updates animation.
     */
    private void updateAnimation() {
      if (nwnh != null) {
        nwnh.terminate();
      }
      nwnh = node.animateToBounds(nx, ny, nw, nh,
                                  BrowserProperties.getAnimationDuration());
    }

    /**
     * @param newX int
     * @param newY int
     */
    public void animatePositionTo(final int newX, final int newY) {
      nx = newX;
      ny = newY;
      updateAnimation();
    }

    /**
     * @param str String
     */
    public final void setTooltip(final String str) {
      getPNode().addClientProperty("tooltip", str);
    }
  }
```

F.7.3. PicGroup

```
package kiel.browser.view.piccolo;

import java.util.Iterator;

import edu.umd.cs.piccolo.PNode;
import edu.umd.cs.piccolo.activities.PActivity;
import edu.umd.cs.piccolo.activities.PInterpolatingActivity;
import edu.umd.cs.piccolo.util.PAffineTransform;
import edu.umd.cs.piccolo.util.PUtil;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.GroupNode;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Implements GroupNode in Piccolo.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £ last modified £Date: 2006/03/07 22:02:05 £
 */
public class PicGroup extends PicNode implements GroupNode {
  /**
   * Position.
   */
  private int xPos, yPos;
  /**
   * New position.
   */
  private int nx, ny;
  /**
   * Animation for position.
   */
  private PActivity posAni = null;
  /**
   * Animation for transformation.
   */
  private PInterpolatingActivity transAni = null;

  /**
   * @param g PNode
   */
  public PicGroup(final PNode g) {
    super(g);
    xPos = 0;
    yPos = 0;
    nx = xPos;
    ny = yPos;
  }

  /**
   * Updateing offset.
   */
  private void update() {
    ((PNode) getPNode()).setOffset(xPos, yPos);
  }

  /**
   * setXPos.
   *
   * @param x int
   */
  public final void setXPos(final int x) {
    xPos = x;
    nx = xPos;
    update();
  }

  /**
   * setYPos.
   *
   * @param y int
   */
  public final void setYPos(final int y) {
    yPos = y;
    ny = yPos;
    update();
  }

  /**
   * Animate position.
   * @param newX int
   * @param newY int
   */
  public final void animatePositionTo(final int newX,
                                      final int newY) {
    nx = newX;
    ny = newY;
    updateAnimation();
  }

  /**
   * Update Animation.
   */
  private void updateAnimation() {
    if (posAni != null) {
      posAni.terminate();
    }
    posAni = getPNode().animateToPositionScaleRotation(nx, ny, 1.0, 0.0,
             BrowserProperties.getAnimationDuration());
    xPos = nx;
    yPos = ny;
  }

  /**
   * Remove.
   */
  private void remove() {
//    System.out.println("___________REMOVING__________");
```

```
    getParent().removeChild(this);
  }

  /**
   * @param removeAfterAnimation boolean
   */
  public final void animatedRemove(final boolean removeAfterAnimation) {
    if (transAni != null) {
      transAni.terminate();
    }
    if (removeAfterAnimation) {
      PiccoloToolkit.updateListOfNodes(null, getId());
      getParent().getChildNodes().remove(this);
    }

    PAffineTransform t = getPNode().getTransform();
    final double scale = 0.1d;
    t.setScale(scale);
    t.setOffset(0.0d, 0.0d);
    getPNode().animateToTransform(t, BrowserProperties.getAnimationDuration());

    transAni = new PInterpolatingActivity(
      BrowserProperties.getAnimationDuration(),
      PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
        protected void activityStarted() {
          super.activityStarted();
        }

        public void setRelativeTargetValue(final float zeroToOne) {
          getPNode().setTransparency(1 - zeroToOne);
        }

        protected void activityFinished() {
          super.activityFinished();
          if (!removeAfterAnimation) {
            getPNode().setTransparency(0.0f);
            getPNode().setVisible(false);
          } else {
            getPNode().getParent().removeChild(getPNode());
          }
        }
      };
      getPNode().addActivity(transAni);
  }

  /**
   * @param start PNode
   * @param zeroToOne float
   */
  private void applyAllChilds(final PNode start,
                              final float zeroToOne) {
    Iterator iter = start.getChildrenIterator();
    PNode child = null;
    while (iter.hasNext()) {
      child = (PNode) iter.next();
      child.setTransparency(zeroToOne);
      applyAllChilds(child, zeroToOne);
    }
  }

  /**
   * Animated insertion.
   */
  public final void animatedInsert() {
    if (transAni != null) {
      transAni.terminate();
    }

    PAffineTransform t = getPNode().getTransform();
    t.setScale(1.0);
    t.setOffset(xPos, yPos);
    getPNode().animateToTransform(t, BrowserProperties.getAnimationDuration());

    // @todo optimisable because setTransparency works for all descents
    transAni = new PInterpolatingActivity(
      BrowserProperties.getAnimationDuration() / 2,
      PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
        protected void activityStarted() {
          super.activityStarted();
          getPNode().setVisible(true);
        }


        public void setRelativeTargetValue(final float zeroToOne) {
          getPNode().setTransparency(zeroToOne);
        }

        protected void activityFinished() {
          super.activityFinished();
          getPNode().setTransparency(1.0f);
        }
      };
      getPNode().addActivity(transAni);
  }

}
```

F.7.4. PicLine

```
package kiel.browser.view.piccolo;

import java.awt.BasicStroke;

import edu.umd.cs.piccolo.activities.PActivity;
import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.util.PUtil;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.LineNode;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Implements a LineNode in Piccolo.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public class PicLine extends PicNode implements LineNode {
  /**
   * Start- and Endpoint.
   */
  private int x1, x2, y1, y2;
  /**
   * New Start- and Endpoint.
   */
  private int nx, ny, nx2, ny2;
  /**
   * Animation to new EndPosition.
   */
  private PActivity newEndPos = null;

  /**
   * @param aNode PPath
   */
  public PicLine(final PPath aNode) {
    super(aNode);
    x1 = 0;
    y1 = 0;
    x2 = 0;
    y2 = 0;
    nx2 = x2;
    ny2 = y2;
    nx = x1;
    ny = y1;
    aNode.setStroke(getDefaultStroke());
  }

  /**
   * Redraws the line.
   */
  private void updateLine() {
    ((PPath) getPNode()).reset();
    ((PPath) getPNode()).moveTo(x1, y1);
    ((PPath) getPNode()).lineTo(x2, y2);
  }

  /**
   * @return int
   */
  public final int getX() {
    return x1;
  }

  /**
   * @return int
   */
  public final int getY() {
    return y1;
  }

  /**
   * @param x int
   */
  public final void setXPos(final int x) {
    x1 = x;
    updateLine();
  }

  /**
   * @param y int
   */
  public final void setYPos(final int y) {
    y1 = y;
    updateLine();
  }

  /**
   * @param x int
   */
  public final void setEndXPos(final int x) {
    x2 = x;
    updateLine();
  }

  /**
   * @param y int
   */
  public final void setEndYPos(final int y) {
    y2 = y;
    updateLine();
  }

  /**
   *  Set the line dashed.
   */
  public final void setDashed() {
    final float d1 = 6.0f;
```

```
    final float d2 = 3.0f;
    BasicStroke stroke = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
                                         BasicStroke.JOIN_MITER, 1.0f,
                                         new float[] {d1, d2}, 1.0f);
    if (((PPath) getPNode()).getStroke() != null) {
      stroke = (BasicStroke) ((PPath) getPNode()).getStroke();
    }
    BasicStroke newStroke = new BasicStroke(stroke.getLineWidth(),
                                            stroke.getEndCap(),
                                            stroke.getLineJoin(),
                                            stroke.getMiterLimit(),
                                            new float[] {d1, d2}, 1.0f);
    ((PPath) getPNode()).setStroke(newStroke);
  }

  /**
   * Animate Line.
   * @param newX int
   * @param newY int
   * @param newX2 int
   * @param newY2 int
   */
  public final void animateLine(final int newX, final int newY,
                                final int newX2, final int newY2) {
    nx = newX; // animate x1/y1 to this
    ny = newY;
    nx2 = newX2; // animate x2/y2 to this
    ny2 = newY2;
    if (newEndPos != null) {
      newEndPos.terminate();
    }
    newEndPos = new PActivity(BrowserProperties.getAnimationDuration(),
                              PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
      private float dx2 = nx2 - x2;
      private float dy2 = ny2 - y2;
      private float dx = nx - x1;
      private float dy = ny - y1;

      protected void activityStep(final long elapsedTime) {
        super.activityStep(elapsedTime);
        float ax2 = x2, ay2 = y2, f = (float) elapsedTime
                                       / (float) getDuration();
        float ax1 = x1, ay1 = y1;
        ax2 += dx2 * f;
        ay2 += dy2 * f;
        ax1 += dx * f;
        ay1 += dy * f;

        ((PPath) getPNode()).reset();
        ((PPath) getPNode()).moveTo(ax1, ay1);
        ((PPath) getPNode()).lineTo(ax2, ay2);
      }

      protected void activityFinished() {
        super.activityFinished();
        x2 = nx2;
        y2 = ny2;
        x1 = nx;
        y1 = ny;
        ((PPath) getPNode()).reset();
        ((PPath) getPNode()).moveTo(x1, y1);
        ((PPath) getPNode()).lineTo(x2, y2);
      }
    };
    getPNode().addActivity(newEndPos);
  }

  /**
   *
   * @return int
   */
  public final int getEndY() {
    return y2;
  }

  /**
   * @return int
   */
  public final int getEndX() {
    return x2;
  }
}
```

F.7.5. PicCircle

```
package kiel.browser.view.piccolo;

import edu.umd.cs.piccolo.activities.PInterpolatingActivity;
import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.util.PBounds;
import edu.umd.cs.piccolo.util.PUtil;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.CircleNode;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/03/26 20:21:11 £
 */
public class PicCircle extends PicNode implements CircleNode {
  /**
   * X, y, width and height.
   */
  private int x, y, w, h;
  /**
   * New x, y, width and height.
   */
  private int nh, nw, nx, ny;
  /**
   * new width new height animation.
   */
  private PInterpolatingActivity nwnh = null;

  /**
   * @param aNode PPath
   */
  public PicCircle(final PPath aNode) {
    super(aNode);
    x = 0;
    y = 0;
    w = 0;
    h = 0;
    aNode.setStroke(getDefaultStroke());
  }

  /**
   * Draws circle. With Quads.
   */
  private void createCircle() {
    ((PPath) getPNode()).reset();
    ((PPath) getPNode()).moveTo(x + w / 2.0f, y);
    ((PPath) getPNode()).quadTo(x + w, y, x + w, y + h / 2.0f);
    ((PPath) getPNode()).quadTo(x + w, y + h, x + w / 2.0f, y + h);
    ((PPath) getPNode()).quadTo(x, y + h, x, y + h / 2.0f);
    ((PPath) getPNode()).quadTo(x, y, x + w / 2.0f, y);
  }

  /**
   * setWidth.
   *
   * @param height int
   */
  public final void setHeight(final int height) {
    h = height;
    nh = h;
    createCircle();
  }

  /**
   * setWidth.
   *
   * @param width int
   */
  public final void setWidth(final int width) {
    w = width;
    nw = w;
    createCircle();
  }

  /**
   * setXPos.
   *
   * @param aXpos int
   */
  public final void setXPos(final int aXpos) {
    x = aXpos;
    nx = x;
    createCircle();
  }

  /**
   * setYPos.
   *
   * @param aYpos int
   */
  public final void setYPos(final int aYpos) {
    y = aYpos;
    ny = y;
    createCircle();
  }

  /**
   * @param newHeight int
   */
  public final void animateHeightTo(final int newHeight) {
    nh = newHeight;
    updateAnimation();
  }

  /**
   * @param newWidth int
   */
```

```
    public final void animateWidthTo(final int newWidth) {
      nw = newWidth;
      updateAnimation();
    }

    /**
     * @param newX int
     * @param newY int
     */
    public final void animatePositionTo(final int newX, final int newY) {
      nx = newX;
      ny = newY;
      updateAnimation();
    }

    /**
     * Updates the animation.
     */
    private void updateAnimation() {
    if (nwnh != null) {
        nwnh.terminate();
      }
      nwnh = new PInterpolatingActivity(BrowserProperties.getAnimationDuration(),
                                        PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
        private PBounds src;
        private PBounds dst;

        protected void activityStarted() {
          src = new PBounds(x, y, w, h);
          dst = new PBounds(nx, ny, nw, nh);
          super.activityStarted();
        }

        public void setRelativeTargetValue(final float zeroToOne) {
          x = (int) Math.round(src.x + (zeroToOne * (dst.x - src.x)));
          y = (int) Math.round(src.y + (zeroToOne * (dst.y - src.y)));
          w = (int) Math.round(src.width + (zeroToOne * (dst.width - src.width)));
          h = (int) Math.round(src.height
                               + (zeroToOne * (dst.height - src.height)));
          createCircle();
        }

      };
      getPNode().addActivity(nwnh);

    }

}
```

F.7.6. PicPath

```
package kiel.browser.view.piccolo;

import java.awt.Color;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

import kiel.graphicalInformations.CurvetoPath;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.MovetoPath;
import kiel.graphicalInformations.PathElement;
import kiel.graphicalInformations.Point;
import kiel.graphicalInformations.QuadraticCurvetoPath;
import edu.umd.cs.piccolo.activities.PInterpolatingActivity;
import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.util.PBounds;
import edu.umd.cs.piccolo.util.PUtil;
import kiel.browser.BrowserException;
import kiel.browser.BrowserProperties;
import kiel.browser.view.BrowserCanvas;
import kiel.browser.view.rendering.Node;
import kiel.browser.view.rendering.PathNode;
import kiel.util.Line;
import kiel.util.LinesDoNotIntersectException;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.9 £ last modified £Date: 2006/05/17 15:35:02 £
 */
public class PicPath extends PicNode implements PathNode {
  /**
   *
   */
  private ArrayList elements = new ArrayList();
  /**
   *
   */
  private PicNode startMarker = null;
  /**
   *
   */
  private PicNode stopMarker = null;
  /**
   *
   */
  private boolean startAdded = false;
  /**
   *
   */
  private PInterpolatingActivity morpher = null;
  /**
   *
   */
  private ArrayList aSourcePath = new ArrayList();
  /**
   *
   */
  private ArrayList aTargetPath = new ArrayList();
  /**
   *
   */
  private int ma = 2;

  /**
   *
   * @param aNode PPath
   */
  public PicPath(final PPath aNode) {
    super(aNode);
    aNode.setStroke(getDefaultStroke());
  }

  /**
   *
   * @param allPathElements Collection
   */
  public final void setPathElements(final Collection allPathElements) {
    elements.clear();
    elements.addAll(allPathElements);
    ((PPath) getPNode()).reset();
    Iterator iter = allPathElements.iterator();
    while (iter.hasNext()) {
      PathElement path = (PathElement) iter.next();
      if (path.getClass() == LinetoPath.class) {
        ((PPath) getPNode()).lineTo(path.getTargetPoint().getX(),
                                    path.getTargetPoint().getY());
      } else if (path.getClass() == MovetoPath.class) {
        ((PPath) getPNode()).moveTo(path.getTargetPoint().getX(),
                                    path.getTargetPoint().getY());
      } else if (path.getClass() == QuadraticCurvetoPath.class) {
        ((PPath) getPNode()).quadTo(path.getControlPoint1().getX(),
                                    path.getControlPoint1().getY(),
                                    path.getTargetPoint().getX(),
                                    path.getTargetPoint().getY());
      } else if (path.getClass() == CurvetoPath.class) {
        ((PPath) getPNode()).curveTo(path.getControlPoint1().getX(),
                                     path.getControlPoint1().getY(),
                                     path.getControlPoint2().getX(),
                                     path.getControlPoint2().getY(),
                                     path.getTargetPoint().getX(),
                                     path.getTargetPoint().getY());
      }
    }
```

```
    }

    /**
     *
     * @param pa Node
     */
    public final void setMarkerEnd(final Node pa) {
      if (elements.size() >= 2) {
        Point end = ((PathElement) elements.get(elements.size() - 1)).
                    getTargetPoint();
        Point before = ((PathElement) elements.get(elements.size() - 2)).
                       getTargetPoint();
        computeAngle(pa, before, end);
        ((PicNode) pa).getPNode().setOffset(end.getX(), end.getY());
        getPNode().addChild(((PicNode) pa).getPNode());
        stopMarker = (PicNode) pa;
      }
    }

    /**
     *
     * @param pa Node
     * @param start Point
     * @param end Point
     */
    private void computeAngle(final Node pa, final Point start, final Point end) {
      // resets rotation
      ((PicNode) pa).getPNode().setTransform(null);

      // compute new rotation
      Point paVec = new Point(1, 0);
      Point pVec = new Point(end.getX() - start.getX(),
                             end.getY() - start.getY());
      double r = (paVec.getX() * pVec.getX() + paVec.getY() * pVec.getY())
            / (Math.sqrt(pVec.getX() * pVec.getX() + pVec.getY() * pVec.getY()));
      if (pVec.getY() < 0) {
        ((PicNode) pa).getPNode().rotate(-Math.acos(r));
      } else {
        ((PicNode) pa).getPNode().rotate(Math.acos(r));
      }

      // show line angle is computed for
//    System.out.println("Angel : " + Math.toDegrees(Math.acos(r)) + " " + pVec);
//      if (BrowserProperties.getLogLevel() <= 1) {
        PPath l = PPath.createLine(end.getX(), end.getY(), start.getX(),
                                   start.getY());
//        l.setStrokePaint(Color.BLUE);
//        getPNode().addChild(l);
//      }
    }

    /**
     *
     * @param pa Node
     */
    public final void setMarkerStart(final Node pa) {
      if (elements.size() >= 2) {
        Point end = ((PathElement) elements.get(1)).getTargetPoint();
        Point start = ((PathElement) elements.get(0)).getTargetPoint();
        computeAngle(pa, start, end);
        ((PicNode) pa).getPNode().setOffset(start.getX(), start.getY());
        getPNode().addChild(((PicNode) pa).getPNode());
        startMarker = (PicNode) pa;
        startAdded = true;
      }
    }


    /**
     * Debug method for showing a point on the canvas.
     * @param p Point
     */
    private void showPoint(final Point p) {
      final int length = 5;
      if (ma % 2 == 0) {
        ((PPath) getPNode()).moveTo(p.getX() - length, p.getY());
        ((PPath) getPNode()).lineTo(p.getX() + length, p.getY());
        ((PPath) getPNode()).moveTo(p.getX(), p.getY() - length);
        ((PPath) getPNode()).lineTo(p.getX(), p.getY() + length);
      } else {
        ((PPath) getPNode()).moveTo(p.getX() - length, p.getY() - length);
        ((PPath) getPNode()).lineTo(p.getX() + length, p.getY() + length);
        ((PPath) getPNode()).moveTo(p.getX() - length, p.getY() + length);
        ((PPath) getPNode()).lineTo(p.getX() + length, p.getY() - length);
      }
      ma++;
    }

    /**
     *
     * @param node PBounds
     * @param line Point
     * @return Point
     * @throws LinesDoNotIntersectException ex
     */
    private static Point getPointOnBounds(PBounds node, Point line)
      throws  LinesDoNotIntersectException {
      Point p = null, h, s;
      Point nodecenter = new Point((int) node.getCenter2D().getX(),
                                   (int) node.getCenter2D().getY());

      //    showPoint(nodecenter);

      Line r;
      Line l = new Line(nodecenter, line);

      if (line.getY() < nodecenter.getY()) {
        h = new Point((int) (node.x + node.width), (int) node.y);
        s = new Point((int) node.x, (int) node.y);
      } else {
        s = new Point((int) (node.x + node.width),
                      (int) (node.y + node.height));
        h = new Point((int) node.x,
                      (int) (node.y + node.height));
      }
      try {
        r = new Line(s, h);
        p = l.intersectInRange(r, s, h);
      } catch (LinesDoNotIntersectException ex) {
        if (line.getX() > nodecenter.getX()) {
```

```
        s = new Point((int) (node.x + node.width),
                      (int) (node.y + node.height));
        h = new Point((int) (node.x + node.width), (int) node.y);
      } else {
        s = new Point((int) node.x, (int) node.y);
        h = new Point((int) node.x,
                      (int) (node.y + node.height));
      }
      r = new Line(s, h);
      p = l.intersectInRange(r, s, h);
    }
    return p;
  }

  private static Point animateFromTo(PathElement src,
                                     PathElement dst,
                                     float elapsed,
                                     PPath path) {
    if (src.getClass() != dst.getClass()) {
      throw new RuntimeException("Source and destination element are not the "
                                 + "same");
    }
    Point p = null;
    if (src.getClass() == LinetoPath.class) {
      p = new Point(Math.round(src.getTargetPoint().getX() + (elapsed
        * (dst.getTargetPoint().getX() - src.getTargetPoint().getX()))),
                    Math.round(src.getTargetPoint().getY() + (elapsed
        * (dst.getTargetPoint().getY() - src.getTargetPoint().getY()))));
      path.lineTo(p.getX(), p.getY());
    } else if (src.getClass() == CurvetoPath.class) {
      p = new Point(Math.round(src.getTargetPoint().getX() + (elapsed
        * (dst.getTargetPoint().getX() - src.getTargetPoint().getX()))),
                    Math.round(src.getTargetPoint().getY() + (elapsed
        * (dst.getTargetPoint().getY() - src.getTargetPoint().getY()))));
      path.curveTo(Math.round(src.getControlPoint1().getX() + (elapsed
        * (dst.getControlPoint1().getX() - src.getControlPoint1().getX()))),
                   Math.round(src.getControlPoint1().getY() + (elapsed
        * (dst.getControlPoint1().getY() - src.getControlPoint1().getY()))),
                   Math.round(src.getControlPoint2().getX() + (elapsed
        * (dst.getControlPoint2().getX() - src.getControlPoint2().getX()))),
                   Math.round(src.getControlPoint2().getY() + (elapsed
        * (dst.getControlPoint2().getY() - src.getControlPoint2().getY()))),
                   p.getX(), p.getY());
    } else {
      System.err.println("ERR: " + src.getClass() + " " + dst.getClass());
    }
    return p;
  }

  public void morphPath(final ArrayList sourcePath,
                        final ArrayList targetPath,
                        final Node source,
                        final Node target) {
    aSourcePath.clear();
    aTargetPath.clear();

    if (morpher != null) {
      morpher.terminate();
    }
    try {
      BrowserCanvas.correctPath(sourcePath, targetPath);
    } catch (BrowserException ex) {
      System.err.println("ERR: " + ex);
      return;
    }

    if (source == null | target == null) {
//      System.out.println("ANIMATION IMPOSSIBLE BECAUSE TARGET OR SOURCE= NULL");
      return;
    }

    if (sourcePath.size() == targetPath.size() && sourcePath.size() > 0) {
      for (int j = 1; j < sourcePath.size() - 1; j++) {
        aSourcePath.add(sourcePath.get(j));
        aTargetPath.add(targetPath.get(j));
      }

      morpher = new PInterpolatingActivity(
        BrowserProperties.getAnimationDuration(),
        PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
        private Point2D a1;
        private Point2D a2;
        private Point sp;
        private Point tp;
        private PBounds srcBounds;

        protected void activityStarted() {
          if (startMarker != null && startAdded) {
            ((PPath) getPNode()).removeChild(startMarker.getPNode());
          }
          ((PPath) getPNode()).reset();
          startAdded = false;
        }

        public void setRelativeTargetValue(float zeroToOne) {
          PathElement src = (PathElement) sourcePath.get(0);
          PathElement dst = (PathElement) targetPath.get(0);
          a2 = new java.awt.Point(Math.round(src.getTargetPoint().getX()
                                             + (zeroToOne
                                                * (dst.getTargetPoint().getX()
            - src.getTargetPoint().getX()))),
                                  Math.round(src.getTargetPoint().getY()
                                             + (zeroToOne
                                                * (dst.getTargetPoint().getY()
            - src.getTargetPoint().getY()))));
          src = (PathElement) sourcePath.get(sourcePath.size() - 1);
          dst = (PathElement) targetPath.get(targetPath.size() - 1);
          a1 = new java.awt.Point(Math.round(src.getTargetPoint().getX()
                                             + (zeroToOne
                                                * (dst.getTargetPoint().getX()
            - src.getTargetPoint().getX()))),
                                  Math.round(src.getTargetPoint().getY()
                                             + (zeroToOne
                                                * (dst.getTargetPoint().getY()
            - src.getTargetPoint().getY()))));
          try {
            ((PPath) getPNode()).reset();
            sp = getPointOnBounds(new PBounds(
              ((PicNode) source).getPNode().getParent().getOffset().getX(),
              ((PicNode) source).getPNode().getParent().getOffset().getY(),
```

```
      ((PicNode) source).getPNode().getWidth(),
      ((PicNode) source).getPNode().getHeight()),
                          new Point((int) a2.getX(),
                                    (int) a2.getY()));
    tp = getPointOnBounds(new PBounds(
      ((PicNode) target).getPNode().getParent().getOffset().getX(),
      ((PicNode) target).getPNode().getParent().getOffset().getY(),
      ((PicNode) target).getPNode().getWidth(),
      ((PicNode) target).getPNode().getHeight())
                          , new Point((int) a1.getX(),
                                      (int) a1.getY()));
    Point p = new Point((int) a2.getX(),
                        (int) a2.getY());

    //            showPoint(p);
    //            showPoint(sp);

    ((PPath) getPNode()).moveTo(sp.getX(), sp.getY());
    Point before = sp;
    for (int i = 0; i < aTargetPath.size(); i++) {
      before = animateFromTo((PathElement) aSourcePath.get(i),
                             (PathElement) aTargetPath.get(i),
                             zeroToOne,
                             ((PPath) getPNode()));
    }
    ((PPath) getPNode()).lineTo(tp.getX(), tp.getY());
    if (stopMarker != null) {
      computeAngle(stopMarker, before, tp);
      ((PicNode) stopMarker).getPNode().setOffset(tp.getX(), tp.getY());
    } else {
      throw new RuntimeException(
        "stop marker is null, but should always be set hmm strange");
    }
  } catch (LinesDoNotIntersectException ex) {
      System.err.println("" + source.getId() + "->" + target.getId()
                         + " " + ex.getMessage());
    }
  }

  protected void activityFinished() {
    super.activityFinished();
    setPathElements(targetPath);

    if (stopMarker != null) {
      Point end = ((PathElement) targetPath.get(targetPath.size() - 1)).
                  getTargetPoint();
      Point before = ((PathElement) targetPath.get(
                       targetPath.size() - 2)).getTargetPoint();
      computeAngle(stopMarker, before, end);
      ((PicNode) stopMarker).getPNode().setOffset(end.getX(), end.getY());
    } else {
      throw new RuntimeException(
        "stop marker is null but should be always set hmm strange");
    }
    if (startMarker != null) {
      setMarkerStart(startMarker);
    }
  }
};
getPNode().addActivity(morpher);
} else {
System.err.println("ERR:" + sourcePath.size() + " vs. "
                   + targetPath.size());
}
}
}
```

F.7.7. PicRect

```
package kiel.browser.view.piccolo;

import edu.umd.cs.piccolo.activities.PInterpolatingActivity;
import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.util.PBounds;
import edu.umd.cs.piccolo.util.PUtil;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.RectNode;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Implements a RectNode for Piccolo.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public class PicRect extends PicNode implements RectNode {
  /**
   *
   */
  private int x, y, w, h, rx, ry;
  /**
   *
   */
  private int nh, nw, nx, ny;
  /**
   *
   */
  private PInterpolatingActivity nwnh = null;

  /**
   *
   * @param aNode PPath
   */
  public PicRect(final PPath aNode) {
    super(aNode);
    x = 0;
    y = 0;
    w = 0;
    h = 0;
    rx = 0;
    ry = 0;
    aNode.setStroke(getDefaultStroke());
  }

  /**
   *
   */
  private void updateRect() {
    ((PPath) getPNode()).reset();
    if (rx == 0 && ry == 0) {
      ((PPath) getPNode()).moveTo(x, y);
      ((PPath) getPNode()).lineTo(x, y + h);
      ((PPath) getPNode()).lineTo(x + w, y + h);
      ((PPath) getPNode()).lineTo(x + w, y);
      ((PPath) getPNode()).lineTo(x, y);
    } else {
      ((PPath) getPNode()).moveTo(x, y + ry);
      ((PPath) getPNode()).lineTo(x, y + h - ry);
      ((PPath) getPNode()).quadTo(x, y + h, x + rx, y + h);
      ((PPath) getPNode()).lineTo(x + w - rx, y + h);
      ((PPath) getPNode()).quadTo(x + w, y + h, x + w, y + h - ry);
      ((PPath) getPNode()).lineTo(x + w, y + ry);
      ((PPath) getPNode()).quadTo(x + w, y, x + w - rx, y);
      ((PPath) getPNode()).lineTo(x + rx, y);
      ((PPath) getPNode()).quadTo(x, y, x, y + ry);
    }
  }

  /**
   * setWidth.
   *
   * @param height int
   */
  public final void setHeight(final int height) {
    h = height;
    nh = h;
    updateRect();
  }

  /**
   * setWidth.
   *
   * @param width int
   */
  public final void setWidth(final int width) {
    w = width;
    nw = w;
    updateRect();
  }

  /**
   * setXPos.
   *
   * @param aXpos int
   */
  public final void setXPos(final int aXpos) {
    x = aXpos;
    nx = x;
    updateRect();
  }

  /**
   * setYPos.
   *
   * @param aYpos int
   */
```

```
    public final void setYPos(final int aYpos) {
      y = aYpos;
      ny = y;
      updateRect();
    }

    /**
     * @param aRx int
     */
    public final void setRx(final int aRx) {
      rx = aRx;
      updateRect();
    }

    /**
     * @param aRy int
     */
    public final void setRy(final int aRy) {
      ry = aRy;
      updateRect();
    }

    /**
     * @return int
     */
    public final int getRx() {
      return rx;
    }

    /**
     * @return int
     */
    public final int getRy() {
      return ry;
    }

    /**
     * @param newHeight int
     */
    public final void animateHeightTo(final int newHeight) {
      nh = newHeight;
      updateAnimation();
    }

    /**
     * @param newWidth int
     */
    public final void animateWidthTo(final int newWidth) {
      nw = newWidth;
      updateAnimation();
    }

    /**
     * @param newX int
     * @param newY int
     */
    public final void animatePositionTo(final int newX,
                                        final int newY) {
      nx = newX;
      ny = newY;
      updateAnimation();
    }

    /**
     * Update animation.
     */
    private void updateAnimation() {
      if (nwnh != null) {
        nwnh.terminate();
      }
      nwnh = new PInterpolatingActivity(BrowserProperties.getAnimationDuration(),
                                        PUtil.DEFAULT_ACTIVITY_STEP_RATE) {
        private PBounds src;
        private PBounds dst;

        protected void activityStarted() {
          src = new PBounds(x, y, w, h);
          dst = new PBounds(nx, ny, nw, nh);
          super.activityStarted();
          getPNode().startResizeBounds();
        }

        public void setRelativeTargetValue(final float zeroToOne) {
          x = (int) Math.round(src.x + (zeroToOne * (dst.x - src.x)));
          y = (int) Math.round(src.y + (zeroToOne * (dst.y - src.y)));
          w = (int) Math.round(src.width + (zeroToOne * (dst.width - src.width)));
          h = (int) Math.round(src.height
                               + (zeroToOne * (dst.height - src.height)));
          updateRect();
          ((PPath) getPNode()).updateBoundsFromPath();
        }

        protected void activityFinished() {
          super.activityFinished();
          getPNode().endResizeBounds();
        }

      };
      getPNode().addActivity(nwnh);
    }


}
```

F.7.8. PicStyleCanger

```
package kiel.browser.view.piccolo;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;

import edu.umd.cs.piccolo.nodes.PPath;
import edu.umd.cs.piccolo.nodes.PText;
import kiel.browser.view.rendering.StyleChanger;
import kiel.browser.view.rendering.Node;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Interprets CSS values for Piccolo.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.2 £ last modified £Date: 2006/02/08 07:07:49 £
 */
public class PicStyleCanger implements StyleChanger {
  /**
   * @param cssString String
   * @return String[]
   */
  private String[] splitWholeCSS(final String cssString) {
    return cssString.split(";");
  }

  /**
   * @param cssString String
   * @return String[]
   */
  private String[] splitCSS(final String cssString) {
    return cssString.split(":");
  }

  /**
   * Compute color from name or rgb tripel.
   * @param c String
   * @return Color
   */
  private Color getColor(final String c) {
    String color = c.trim();
    final int lastHexVal = 7;
    final int hex = 16;
    final int shift3rd = 16;
    final int shift2nd = 8;
    final int rMask = 0xFF0000;
    final int gMask = 0x00FF00;
    final int bMask = 0x0000FF;
    if (!color.startsWith("#")) {
      if (color.equalsIgnoreCase("black")) {
        return Color.BLACK;
      } else if (color.equalsIgnoreCase("gray")
                 || color.equalsIgnoreCase("grey")) {
        return Color.GRAY;
      } else if (color.equalsIgnoreCase("white")) {
        return Color.WHITE;
      } else if (color.equalsIgnoreCase("blue")) {
        return Color.BLUE;
      } else if (color.equalsIgnoreCase("yellow")) {
        return Color.YELLOW;
      } else if (color.equalsIgnoreCase("green")) {
        return Color.GREEN;
      } else if (color.equalsIgnoreCase("red")) {
        return Color.RED;
      } else if (color.equalsIgnoreCase("pink")) {
        return Color.PINK;
      } else if (color.equalsIgnoreCase("none")) {
        return null;
      }
      System.err.println("Color: " + color + " cannot be converted");
      return null;
    }
    color = color.substring(1, lastHexVal);
    int col = 0;
    try {
      col = Integer.parseInt(color, hex); // Parse hex code
    } catch (NumberFormatException ex) {
      return Color.BLACK;
    }
    int r = (col & rMask) >> shift3rd;
    int g = (col & gMask) >> shift2nd;
    int b = col & bMask;
    return new Color(r, g, b);
  }


  /**
   * changeStyle.
   *
   * @param node Node
   * @param string String
   */
  public final void changeStyle(final Node node, final String string) {
    if (!string.equals("")) {
      ((PicNode) node).setStyle(string);
      if (node.getClass() == PicText.class) {
        String[] cssEntries = splitWholeCSS(string);
        // Defaults
        String fontFamily = null;
        final int defaultFontSize = 12;
        int fontSize = defaultFontSize;
        Color col = Color.BLACK;

        for (int i = 0; i < cssEntries.length; i++) {
          String[] vals = splitCSS(cssEntries[i]);
```

```
            if (vals[0].trim().equalsIgnoreCase("font-family")) {
              fontFamily = vals[1];
            } else if (vals[0].trim().equalsIgnoreCase("font-size")) {
              fontSize = Integer.parseInt(vals[1]);
            } else if (vals[0].trim().equalsIgnoreCase("color")) {
              col = getColor(vals[1]);
            }
          }
          ((PText) ((PicText) node).getPNode()).setFont(
            new Font(fontFamily, Font.PLAIN, fontSize));
          ((PText) ((PicText) node).getPNode()).setTextPaint(col);
        } else if (node.getClass() == PicRect.class
                   || node.getClass() == PicCircle.class
                   || node.getClass() == PicLine.class
                   || node.getClass() == PicPath.class) {
          String[] cssEntries = splitWholeCSS(string);
          // Defaults
          Color fill = null;
          Color strokeColor = null;
          float strokeWidth = 1.0f;
          final float niceLookingWidth = 1.26f;

          for (int i = 0; i < cssEntries.length; i++) {
            String[] vals = splitCSS(cssEntries[i]);
            if (vals[0].trim().equalsIgnoreCase("fill")) {
              fill = getColor(vals[1].trim());
            } else if (vals[0].trim().equalsIgnoreCase("stroke")) {
              strokeColor = getColor(vals[1].trim());
            } else if (vals[0].trim().equalsIgnoreCase("stroke-width")) {
              String v = vals[1].trim();
              String w = "1.0";
              if (v.endsWith("px") || v.endsWith("pt")) {
                w = v.substring(0, v.length() - 2);
              }
              strokeWidth = Float.parseFloat(w) * niceLookingWidth;
            } else {
              if (vals.length == 2) {
                System.err.println(node + " wants to apply " + vals[0].trim()
                                   + ":"
                                   + vals[1].trim());
              } else {
                System.err.println(node + " wants to apply just [1]"
                                   + vals[0].trim());
              }
            }
          }
          ((PPath) ((PicNode) node).getPNode()).setPaint(fill);
          ((PPath) ((PicNode) node).getPNode()).setStrokePaint(strokeColor);
          BasicStroke stroke = (BasicStroke) ((PPath)
                                  ((PicNode) node).getPNode()).getStroke();
          if (stroke == null) {
            stroke = new BasicStroke();
          }
          ((PPath) ((PicNode) node).getPNode()).setStroke(
            new BasicStroke(strokeWidth,
                            stroke.getEndCap(),
                            stroke.getLineJoin(),
                            stroke.getMiterLimit(),
                            stroke.getDashArray(),
                            stroke.getDashPhase()));

        } else {
          System.err.println("Wants to changes style from Node: " + node.getId()
                             + " Class " + node.getClass());
          System.err.println(" - to " + string);
        }
      }
    }
  }
```

F.7.9. PicText

```
package kiel.browser.view.piccolo;

import edu.umd.cs.piccolo.activities.PActivity;
import edu.umd.cs.piccolo.nodes.PText;
import kiel.browser.BrowserProperties;
import kiel.browser.view.rendering.TextNode;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Implements a TextNode for Piccolo.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.5 £ last modified £Date: 2006/03/07 22:02:05 £
 */
public class PicText extends PicNode implements TextNode {
  /**
   *  New position.
   */
  private int nx, ny;
  /**
   *  Position animation.
   */
  private PActivity posAni = null;

  /**
   * @param aNode PText
   */
  public PicText(final PText aNode) {
    super(aNode);
//    aNode.setPickable(true);
    nx = 0;
    ny = 0;
  }

  /**
   * @param text String
   */
  public final void setText(final String text) {
    ((PText) getPNode()).setText(text);
    if (text == null) {
      ((PText) getPNode()).setVisible(false);
    } else {
      ((PText) getPNode()).setVisible(true);
    }
  }

  /**
   * @return String
   */
  public final String getText() {
    return ((PText) getPNode()).getText();
  }

  /**
   * @param x int
   */
  public final void setXPos(final int x) {
    super.setXPos(x);
    nx = x;
  }

  /**
   * @param y int
   */
  public final void setYPos(final int y) {
    super.setYPos(y);
    ny = y;
  }

  /**
   * @param newX int
   * @param newY int
   */
  public final void animatePositionTo(final int newX,
                                      final int newY) {
    nx = newX;
    ny = newY;
    if (posAni != null) {
      posAni.terminate();
    }
    posAni = getPNode().animateToPositionScaleRotation(nx, ny, 1.0, 0.0,
      BrowserProperties.getAnimationDuration());
  }

}
```

F.8. kiel.browser.view.svg

F.8.1. SVGText

```
package kiel.browser.view.svg;

import java.awt.Font;

import kiel.graphicalInformations.Properties;
import kiel.browser.view.rendering.TextNode;
import org.w3c.dom.Text;
import org.w3c.dom.svg.SVGTextElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGText extends SVGNode implements TextNode {
  /**
   *
   */
  private String text = "";
  /**
   *
   */
  private int y = 0;
  /**
   *
   */
  private int x = 0;
  /**
   *
   * @param aText SVGTextElement
   */
  public SVGText(final SVGTextElement aText) {
    super(aText);
    aText.setAttribute("style", "text-anchor:start");
  }

  /**
   *
   * @return Font
   */
  private  Font getFont() {
    String style = getSVGElement().getAttribute("style");
    if (style.indexOf(Properties.getActivityFont().getFamily()) >= 0) {
      return Properties.getActivityFont();
    } else if (style.indexOf(Properties.getDescriptionFont().getFamily())
            >= 0) {
      return Properties.getDescriptionFont();
    }
    return Properties.getLabelFont();
  }

  /**
   *
   * @return int
   */
  public final int getWidth() {
    if (text == null) {
      return 0;
    }
    return Properties.getWidth(getFont(), text);
  }

  /**
   *
   * @return int
   */
  public final int getHeight() {
    if (text == null) {
      return 0;
    }
    return Properties.getHeight(getFont(), text);
  }

  /**
   *
   * @return String
   */
  public final String getText() {
    return text;
  }

  /**
   *
   * @param xpos int
   */
  public final void setXPos(final int xpos) {
    x = xpos;
    update();
  }

  /**
   *
   * @param ypos int
   */
  public final void setYPos(final int ypos) {
    y = ypos;
    update();
```

```
    }

    /**
     * updates infos.
     */
110 private void update() {
      super.setXPos(x - (getWidth() / 2));
      super.setYPos(y + getHeight());
    }

    /**
     *
     * @param aText String
     */
                                                  public final void setText(final String aText) {
                                                    if (aText != null) {
                                                      text = aText;
                                                      for (int i = 0; i < getSVGElement().getChildNodes().getLength(); i++) {
                                                        getSVGElement().removeChild(getSVGElement().getChildNodes().item(i));
                                                      }
                                                      Text textContent = getSVGElement().getOwnerDocument().
                                                                         createTextNode(text);
                                                      getSVGElement().appendChild(textContent);
                                                      update();
                                                    }
                                                  }
                                                }
                                              }
120
130
```

F.8.2. SVGCircle

```
package kiel.browser.view.svg;

import kiel.browser.view.rendering.CircleNode;
import org.w3c.dom.svg.SVGEllipseElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGCircle extends SVGNode implements CircleNode {
  /**
   * Width.
   */
  private int w = 0;
  /**
   * Height.
   */
  private int h = 0;
  /**
   * Position.
   */
  private int xPos = 0;
  /**
   * Position.
   */
  private int yPos = 0;
  /**
   *
   * @param aCircle SVGEllipseElement
   */
  public SVGCircle(final SVGEllipseElement aCircle) {
    super(aCircle);
  }

  /**
   *
   * @param width int
   */
  public final void setWidth(final int width) {
    w = width;
    getSVGElement().setAttribute("rx", "" + (width / 2));
    setXPos(xPos);
  }

  /**
   *
   * @param height int
   */
  public final void setHeight(final int height) {
  h = height;
    getSVGElement().setAttribute("ry", "" + (height / 2));
    setYPos(yPos);
  }

  /**
   *
   * @param x int
   */
  public final void setXPos(final int x) {
    xPos = x;
    getSVGElement().setAttribute("cx", "" + (x + w / 2));
  }

  /**
   *
   * @param y int
   */
  public final void setYPos(final int y) {
    yPos = y;
    getSVGElement().setAttribute("cy", "" + (y + h / 2));
  }
}
```

F.8.3. SVGGroup

```
package kiel.browser.view.svg;

import kiel.browser.view.rendering.GroupNode;
import org.w3c.dom.svg.SVGGElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGGroup extends SVGNode implements GroupNode {
  /**
   *
   */
  private int xPos = 0;
  /**
   *
   */
  private int yPos = 0;

  /**
   *
   * @param g SVGGElement
   */
  public SVGGroup(final SVGGElement g) {
    super(g);
  }

  /**
   *
   * @param x int
   */
  public final void setXPos(final int x) {
    xPos = x;
    getSVGElement().setAttribute("transform", "translate(" + xPos + "," + yPos);
  }

  /**
   *
   * @param y int
   */
  public final void setYPos(final int y) {
    yPos = y;
    getSVGElement().setAttribute("transform", "translate(" + xPos + "," + yPos);
  }

  /**
   * NOT IMPLEMENTED FOR SVG.
   */
  public final void animatedRemove() {
  }

  /**
   * NOT IMPLEMENTED FOR SVG.
   */
  public final void animatedInsert() {
  }

  /**
   * NOT IMPLEMENTED FOR SVG.
   * @param removeAfterAnimation boolean
   *
   */
  public final void animatedRemove(final boolean removeAfterAnimation) {
  }
}
```

F.8.4. SVGLine

```
package kiel.browser.view.svg;

import kiel.browser.view.rendering.LineNode;
import org.w3c.dom.svg.SVGLineElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGLine extends SVGNode implements LineNode {
  /**
   *
   * @param aLine SVGLineElement
   */
  public SVGLine(final SVGLineElement aLine) {
    super(aLine);
  }

  /**
   *
   * @param x int
   */
  public final void setXPos(final int x) {
    getSVGElement().setAttribute("x1", "" + x);
  }

  /**
   *
   * @param y int
   */
  public final void setYPos(final  int y) {
    getSVGElement().setAttribute("y1", "" + y);
  }

  /**
   *
   * @param x int
   */
  public final void setEndXPos(final int x) {
    getSVGElement().setAttribute("x2", "" + x);
  }

  /**
   *
   * @param y int
   */
  public final void setEndYPos(final int y) {
    getSVGElement().setAttribute("y2", "" + y);
  }

  /**
   *  Set line dashed.
   */
  public final void setDashed() {
    getSVGElement().setAttribute("stroke-dasharray", "6,3");
  }

  /**
   * NOT IMPLEMENTED.
   * @param newX int
   * @param newY int
   */
  public final void animateEndPosition(final int newX,
                                       final int newY) {
  }
  /**
   * NOT IMPLEMENTED.
   *
   * @param newX int
   * @param newY int
   * @param newX2 int
   * @param newY2 int
   */
  public final  void animateLine(final  int newX, final int newY,
                          final int newX2, final int newY2) {
  }

  /**
   *
   * @return int
   */
  public final int getEndY() {
    return 0;
  }

  /**
   *
   * @return int
   */
  public final int getEndX() {
  return 0;
  }
}
```

F.8.5. SVGNode

```
package kiel.browser.view.svg;

import java.util.ArrayList;
import java.util.Collection;

import kiel.browser.view.rendering.Node;
import org.w3c.dom.svg.SVGElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGNode implements Node {
  /**
   * SVG Element.
   */
  private SVGElement e;
  /**
   * List of children.
   */
  private ArrayList childs = new ArrayList();

  /**
   *
   * @param element SVGElement
   */
  public SVGNode(final SVGElement element) {
    e = element;
  }

  /**
   *
   * @return SVGElement
   */
  public final SVGElement getSVGElement() {
    return e;
  }

  /**
   *
   * @param childGroup GroupNode
   */
  public final void appendChild(final Node childGroup) {
    childs.add(childGroup);
    e.appendChild(((SVGNode) childGroup).getSVGElement());
  }

  /**
   *
   * @return Object
   */
  public final Collection getChildNodes() {
    return childs;
  }

  /**
   * getId.
   *
   * @return String
   */
  public final String getId() {
    return e.getId();
  }

  /**
   * removeChild.
   *
   * @param node Node
   */
  public final void removeChild(final Node node) {
    childs.remove(node);
    e.removeChild(((SVGNode) node).getSVGElement());
  }

  /**
   * setId.
   *
   * @param anId String
   */
  public final void setId(final String anId) {
    String old = e.getId();
    e.setId(anId);
    SVGToolkit.updateNode(this, old);
  }

  /**
   *
   * @param width int
   */
  public void setWidth(final int width) {
    getSVGElement().setAttribute("width", "" + width);
  }

  /**
   *
   * @param height int
   */
  public void setHeight(final int height) {
    getSVGElement().setAttribute("height", "" + height);
  }

  /**
   *
   * @param x int
```

```
     */
    public void setXPos(final int x) {
      getSVGElement().setAttribute("x", "" + x);
    }

    /**
     *
     * @param y int
     */
    public void setYPos(final int y) {
      getSVGElement().setAttribute("y", "" + y);
    }

    /**
     *
     * @return String
     */
    public final String getStyle() {
      return getSVGElement().getAttribute("style");
    }

    /**
     *
     * @return Node
     */
    public final Node getParent() {
      return null;
    }

    /**
     *
     * @param aParent Node
     */
    public final void setParent(final Node aParent) {
    }

    /**
     *
     * @return int
     */
    public int getWidth() {
      return 0;
    }

    /**
     *
     * @return int
     */
    public int getHeight() {
      return 0;
    }

    /**
     *
     * @return int
     */
    public int getX() {
      return 0;
    }

    /**
     *
     * @return int
     */
    public int getY() {
      return 0;
    }

    /**
     * NOT IMPLEMENTED.
     *
     * @param newHeight int
     */
    public final void animateHeightTo(final int newHeight) {
    }

    /**
     * NOT IMPLEMENTED.
     *
     * @param newWidth int
     */
    public final void animateWidthTo(final int newWidth) {
    }

    /**
     * NOT IMPLEMENTED.
     *
     * @param newX int
     * @param newY int
     */
    public final void animatePositionTo(final int newX,
                                        final int newY) {
    }

    /**
     * NOT IMPLEMENTED.
     * @param str String
     */
    public final void setTooltip(final String str) {
    }
}
```

F.8.6. SVGPath

```
package kiel.browser.view.svg;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

import kiel.browser.view.rendering.Node;
import kiel.browser.view.rendering.PathNode;
import org.w3c.dom.svg.SVGPathElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGPath extends SVGNode implements PathNode {
  /**
   *
   * @param aPath SVGPathElement
   */
  public SVGPath(final SVGPathElement aPath) {
    super(aPath);
  }

  /**
   *
   * @param allPathElements Collection
   */
  public final void setPathElements(final Collection allPathElements) {
    StringBuffer buf = new StringBuffer();
    Iterator iter = allPathElements.iterator();
    while (iter.hasNext()) {
      buf.append(iter.next().toString());
      buf.append('␣');
    }
    getSVGElement().setAttribute("d", buf.toString());
  }

  /**
   * @param pa Node
   */
  public final void setMarkerEnd(final Node pa) {
    getSVGElement().setAttribute("marker-end", "url(#arrow)");
  }

  /**
   * NOT IMPL.
   *
   * @param pa Node
   */
  public final void setMarkerStart(final Node pa) {
  }

  /**
   * NOT IMPL.
   *
   * @param sourcePath Collection
   * @param targetPath Collection
   * @param source Node
   * @param target Node
   */
  public final void morphPath(final Collection sourcePath,
                              final Collection targetPath,
                              final Node source,
                              final Node target) {
  }

  /**
   * NOT IMPL.
   * @param sourcePath ArrayList
   * @param targetPath ArrayList
   * @param source Node
   * @param target Node
   */
  public final void morphPath(final ArrayList sourcePath,
                              final ArrayList targetPath,
                              final Node source,
                              final Node target) {
  }

}
```

F.8.7. SVGRect

```
package kiel.browser.view.svg;

import kiel.browser.view.rendering.RectNode;
import org.w3c.dom.svg.SVGRectElement;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public class SVGRect extends SVGNode implements RectNode {
  /**
   *
   * @param aRect SVGRectElement
   */
  public SVGRect(final SVGRectElement aRect) {
    super(aRect);
    aRect.setAttribute("rx", "10");
    aRect.setAttribute("ry", "10");
  }

  /**
   *
   * @param rx int
   */
  public final void setRx(final int rx) {
  }

  /**
   *
   * @param ry int
   */
  public final void setRy(final int ry) {
  }

  /**
   *
   * @return int
   */
  public final  int getRx() {
    return 0;
  }

  /**
   *
   * @return int
   */
  public final int getRy() {
    return 0;
  }
}
```

F.8.8. SVGToolkit

```
package kiel.browser.view.svg;

import java.awt.Color;
import java.awt.Dimension;
import java.util.Hashtable;

import javax.swing.JComponent;
import javax.swing.text.Document;

import kiel.browser.view.rendering.StyleChanger;
import kiel.browser.view.rendering.CircleNode;
import kiel.browser.view.rendering.GroupNode;
import kiel.browser.view.rendering.LineNode;
import kiel.browser.view.rendering.Node;
import kiel.browser.view.rendering.PathNode;
import kiel.browser.view.rendering.RectNode;
import kiel.browser.view.rendering.TextNode;
import kiel.browser.view.rendering.Toolkit;
import org.csiro.svg.dom.SVGDocumentImpl;
import org.csiro.svg.viewer.Canvas;
import org.w3c.dom.svg.SVGEllipseElement;
import org.w3c.dom.svg.SVGGElement;
import org.w3c.dom.svg.SVGLineElement;
import org.w3c.dom.svg.SVGMarkerElement;
import org.w3c.dom.svg.SVGPathElement;
import org.w3c.dom.svg.SVGRectElement;
import org.w3c.dom.svg.SVGSVGElement;
import org.w3c.dom.svg.SVGTextElement;
import java.awt.Rectangle;
import java.awt.Graphics;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £ last modified £Date: 2006/03/26 20:21:11 £
 */
public class SVGToolkit implements Toolkit {
  /**
   * svg doc.
   */
  private SVGDocumentImpl doc;

  /**
   * canvas.
   */
  private Canvas canvas;

  /**
   * root node.
   */
  private SVGNode root;

  /**
   * Utility class that changes the style of an svg element.
   */
  private static StyleChanger changer = new CSSStyleChanger();

  /**
   * table with kiel 2 svg elements.
   */
  private static Hashtable svg2kiel;

  /**
   * Creates new SVG Toolkit.
   */
  public SVGToolkit() {
    canvas = new org.csiro.svg.viewer.Canvas();
    canvas.setBackground(Color.white);
    canvas.setDoubleBuffer(true);
    doc = new org.csiro.svg.dom.SVGDocumentImpl();
    root = new SVGNode((SVGSVGElement) doc.createElement("svg"));
    doc.appendChild(root.getSVGElement());
    canvas.setSVGDocument((SVGDocumentImpl) doc);
    SVGMarkerElement m = (SVGMarkerElement) doc.createElement("marker");
    m.setId("arrow");
    m.setAttribute("viewBox", "0 0 10 10");
    m.setAttribute("refX", "10");
    m.setAttribute("refY", "3");
    m.setAttribute("markerUnits", "strokeWidth");
    m.setAttribute("markerWidth", "10");
    m.setAttribute("markerHeight", "10");
    m.setAttribute("orient", "auto");
    SVGPathElement pa = (SVGPathElement) doc.createElement("path");
    pa.setAttribute("d", "M 0,0 l 10,3 l -10,3 z");
    pa.setAttribute("fill", "black");
    m.appendChild(pa);
    doc.getRootElement().appendChild(m);
    svg2kiel = new Hashtable();
  }

  /**
   *
   * @return Component that shows the SVGStatechart.
   */
  public final JComponent getComponent() {
    return canvas;
  }

  /**
   * Call this to repaint the canvas.
   */
  public final void repaintCanvas() {
//    canvas.fitToWindow();
    canvas.repaint();
  }
```

```
  /**
   *
   * @return SVGDocument
   */
  public final Document getDocument() {
    return null;
  }

  /**
   * Start playback if there are animations.
   */
  public final void startPlayback() {
    ((SVGDocumentImpl) doc).setChanged();
    //    ((SVGDocumentImpl)doc).resetBeginTime();
    canvas.repaint();
  }

  /**
   *
   * @return SVGStyleChanger to change style
   */
  public final StyleChanger getStyleChanger() {
    return changer;
  }

  /**
   * Clears the document and prepares a new one.
   */
  public final void clearDocument() {
    doc = null;
    root = null;
    System.gc();
    doc = new org.csiro.svg.dom.SVGDocumentImpl();
    root = new SVGNode((SVGSVGElement) doc.createElement("svg"));
    doc.appendChild(root.getSVGElement());
    canvas.setSVGDocument((SVGDocumentImpl) doc);
    SVGMarkerElement m = (SVGMarkerElement) doc.createElement("marker");
    m.setId("arrow");
    m.setAttribute("viewBox", "0␣0␣10␣10");
    m.setAttribute("refX", "10");
    m.setAttribute("refY", "3");
    m.setAttribute("markerUnits", "strokeWidth");
    m.setAttribute("markerWidth", "10");
    m.setAttribute("markerHeight", "10");
    m.setAttribute("orient", "auto");
    SVGPathElement pa = (SVGPathElement) doc.createElement("path");
    pa.setAttribute("d", "M␣0,0␣l␣10,3␣l␣-10,3␣z");
    pa.setAttribute("fill", "black");
    m.appendChild(pa);
    root.getSVGElement().appendChild(m);
  }

  /**
   * shows the created SVG Document.
   */
  public final void showDocument() {
    canvas.setSVGDocument((SVGDocumentImpl) doc);
  }

  /**
   *
   * @return Rectangle
   */
  public Rectangle getCanvasBounds() {
    return new Rectangle(canvas.getWidth(), canvas.getHeight());
  }

  /**
   *
   * @param tooltip String
   */
  public final void setTooltip(final String tooltip) {
    canvas.setToolTipText(tooltip);
  }

  /**
   *
   * @return SVGElement
   */
  public final Node getRoot() {
//    return doc.getRootElement();
    return root;
  }

  /**
   * getElementById.
   *
   * @param string String
   * @return SVGElement
   */
  public final Node getElementById(final String string) {
    return (Node) svg2kiel.get(string);
  }

  /**
   *
   * @return SVGMarkerElement
   */
//  public final SVGMarkerElement createMarkerElement() {
//    return (SVGMarkerElement) doc.createElement("marker");
//  }

  /**
   *
   * @return SVGPathElement
   */
  public final PathNode createPathElement() {
    return new SVGPath((SVGPathElement) doc.createElement("path"));
  }

  /**
   *
   * @return SVGLineElement
   */
  public final LineNode createLineElement() {
    return new SVGLine((SVGLineElement) doc.createElement("line"));
  }

  /**
```

```
     *
     * @return SVGCircleElement
     */
    public final CircleNode createCircleElement() {
      return new SVGCircle((SVGEllipseElement) doc.createElement("ellipse"));
    }

    /**
     *
     * @return SVGRectElement
     */
    public final RectNode createRectElement() {
      return new SVGRect((SVGRectElement) doc.createElement("rect"));
    }

    /**
     *
     * @return SVGAnimateElement
     */
    // public final SVGAnimateElement createAnimationElement() {
    //   return (SVGAnimateElement) doc.createElement("animate");
    // }

    /**
     *
     * @return SVGGElement
     */
    public final GroupNode createGroupElement() {
      return new SVGGroup((SVGGElement) doc.createElement("g"));
    }

    /**
     *
     * @return SVGTextElement
     */
    public final TextNode createTextElement() {
      return new SVGText((SVGTextElement) doc.createElement("text"));
    }

    /**
     *
     * @param node Node
     * @param old String
     */
    public static void updateNode(final Node node,
        final String old) {
      svg2kiel.remove(old);
      svg2kiel.put(node.getId(), node);
    }

    /**
     *
     */
    public void fitToScreen() {
    }

    /**
     *
     * @return JComponent
     */
    public final JComponent getCanvas() {
      return null;
    }

    /**
     *
     */
    public void resetZoom() {
    }

    /**
     *
     * @param alwaysFitToScreen boolean
     */
    public final void setAlwaysFitToScreen(final boolean alwaysFitToScreen) {
    }

    /**
     *
     */
    public void requestFocus() {
    }

    public void printCanvas(Graphics g) {
    }
}
```

F.8.9. CSSStyleChanger

```
package kiel.browser.view.svg;

import kiel.browser.view.rendering.StyleChanger;
import kiel.browser.view.rendering.Node;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 07:07:50 £
 */
public final class CSSStyleChanger implements StyleChanger {
  /**
   * changeStyle.
   *
   * @param node SVGElement
   * @param cssString String
   */
  public void changeStyle(final Node node,
                          final String cssString) {
    String oldstyle = ((SVGNode) node).getSVGElement().getAttribute("style");
    ((SVGNode) node).getSVGElement().setAttribute("style", cssString + ";"
                                                  + oldstyle);
  }
}
```

F.9. kiel.browser.controller

F.9.1. StateChartWalker

```
package kiel.browser.controller;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Edge;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.StatechartFocus;

/**
 * <p>Title: Kiel Browser Controller Class.</p>
 *
 * <p>Description: This class implements a node/edge selection mechanism.
 *   By calling the public methods you can walk through the statechart.
 *   Whenever a new node/transition is entered the models select methods are
 *   called. These selected nodes/edge can be made visible by some BrowserGUI
 *   classes.
 * </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.10 £ last modified £Date: 2006/05/22 19:21:52 £
 */
public class StateChartWalker implements Observer {
  /**
   * Model that send us messages about new charts etc.
   */
  private BrowserModel model;
  /**
   * Constant for this key.
   */
  private static final int LEFT = 0;
  /**
   * Constant for this key.
   */
  private static final int RIGHT = 1;
  /**
   * Constant for this key.
   */
  private static final int UP = 2;
  /**
   * Constant for this key.
   */
  private static final int DOWN = 3;
  /**
   * Constant for this key.
   */
  private static final int PG_UP = 4;
  /**
   * Constant for this key.
   */
  private static final int PG_DOWN = 5;
  /**
   * Constant for this key.
   */
  private static final int FIRST = 6;
  /**
   * Constant for this key.
   */
  private static final int LAST = 7;
  /**
   * Constant for this key.
   */
  private static final int ABS_FIRST = 8;
  /**
   * Constant for this key.
   */
  private static final int ABS_LAST = 9;
  /**
   * Constant for this key.
   */
  private static final int HIST_FW = 10;
  /**
   * Constant for this key.
   */
  private static final int HIST_BK = 11;
  /**
   * Constant for this key.
   */
  private static final int NEXT_ST = 12;
  /**
   * Constant for this key.
   */
  private static final int NEXT_TR = 13;
  /**
   * Constant for this key.
   */
  private static final int NEXT_PS = 14;
  /**
   * The pressed key.
   */
```

```
  private int key;
  /**
   * The actual node1.
   */
  private Node n1;
  /**
   * The actual node1.
   */
  private Node n2;
  /**
   * Flag.
   */
  private boolean walk1 = true;
  /**
   * Flag.
   */
  private boolean setFocus = false;
  /**
   * The actual edge.
   */
  private Edge e;
  /**
   * List of object to walk.
   */
  private ArrayList objs = new ArrayList();
  /**
   * Actual position in objs list.
   */
  private int pos = -1;
  /**
   * History list.
   */
  private ArrayList history = new ArrayList();
  /**
   * Actual position in history.
   */
  private int histPos = -1;
  /**
   * For sorting the transitions.
   */
  private Comparator edgeSort = new PriorityComparator();
  /**
   * For sorting the subnodes.
   */
  private Comparator nodeSort = new PriorityComparator();

  /**
   * Connects this class to the data model.
   * @param aModel BrowserModel
   */
  public StateChartWalker(final BrowserModel aModel) {
    model = aModel;
    model.addObserver(this);
    n1 = null;
    e = null;
  }

  /**
   * Called when down was pressed.
   */
  public final void gotoFirstPressed() {
    System.out.println("Goto First Pressed");
    key = FIRST;
  }

  /**
   * Called when down was pressed.
   */
  public final void gotoAbsolutFirstPressed() {
    System.out.println("Abs First pressed");
    key = ABS_FIRST;
  }

  /**
   * Called when down was pressed.
   */
  public final void gotoDeepestLastPressed() {
    System.out.println("Abs Last Pressed");
    key = ABS_LAST;
  }

  /**
   * Called when down was pressed.
   */
  public final void gotoLastKeyPressed() {
    System.out.println("Last Pressed");
    key = LAST;
  }

  /**
   * Called when down was pressed.
   */
  public final void nextStatePressed() {
    System.out.println("Next State Pressed");
    key = NEXT_ST;
  }

  /**
   * Called when down was pressed.
   */
  public final void nextTransitionPressed() {
    System.out.println("Next Transition Pressed");
    key = NEXT_TR;
  }

  /**
   * Called when down was pressed.
   */
  public final void nextPseudoPressed() {
    System.out.println("Next Pseudo Pressed");
    key = NEXT_PS;
  }

  /**
   * Called when down was pressed.
   */
  public final void historyBackPressed() {
    System.out.println("History Back Pressed");
    key = HIST_BK;
    if (history.size() > 0) {
```

```
      int hPos = histPos - 1;
      if (hPos <= -1) {
        hPos = 0;
      }
      Object o = history.get(hPos);
      if (o instanceof Edge) {
        e = (Edge) o;
        n1 = null;
      } else {
        n1 = (Node) o;
        e = null;
      }
      setFocus = true;
      model.setFocus(n1, null, e);
      setFocus = false;
      histPos = hPos;
    }
  }

  /**
   * Called when down was pressed.
   */
  public final void historyForwardPressed() {
    System.out.println("History Forward Pressed");
    key = HIST_FW;
    if (history.size() > 0) {
      int hPos = histPos + 1;
      if (hPos >= history.size()) {
        hPos = history.size() - 1;
      }
      Object o = history.get(hPos);
      if (o instanceof Edge) {
        e = (Edge) o;
        n1 = null;
      } else {
        n1 = (Node) o;
        e = null;
      }
      setFocus = true;
      model.setFocus(n1, null, e);
      setFocus = false;

      histPos = hPos;
    }
  }

  /**
   * Called when down was pressed.
   */
  public final void specialKey1Pressed() {
    System.out.println("Special Key1 Pressed");
    walk1 = !walk1;
  }

  /**
   * Called when down was pressed.
   */
  public final void specialKey2Pressed() {
    System.out.println("Special Key2 Pressed");
  }

  /**
   * Called when down was pressed.
   */
  public final void pageUpPressed() {
    System.out.println("Page Up Pressed");
    key = PG_UP;
    handleCurserKeys();
  }

  /**
   * Called when down was pressed.
   */
  public final void pageDownPressed() {
    System.out.println("Page Down Pressed");
    key = PG_DOWN;
    handleCurserKeys();
  }

  /**
   * Called when down was pressed.
   */
  public final void leftPressed() {
    System.out.println("Left Pressed");
    key = LEFT;
    handleCurserKeys();
  }

  /**
   * Called when down was pressed.
   */
  public final void rightPressed() {
    System.out.println("Right Pressed");
    key = RIGHT;
    handleCurserKeys();
  }

  /**
   * Called when down was pressed.
   */
  public final void upPressed() {
    System.out.println("Up Pressed");
    key = UP;
    handleCurserKeys();
  }

  /**
   * Called when down was pressed.
   */
  public final void downPressed() {
    System.out.println("Down Pressed");
    key = DOWN;
    handleCurserKeys();
  }

  /**
   * Here we go with our curser keys.
   */
  private void handleCurserKeys() {
```

```
      if (model.getStateChart() != null
          && model.getStateChart().getRootNode() != null) {
          doNodesAndEdgesWalking();
      }
    }

    /**
     * Computes the "first" state of an composite state.
     *  Which is defined as the first initial or the first in the list
     *  of substates.
     * @param root CompositeState
     * @return Node
     */
    private static Node getFirstState(final CompositeState root) {
      Iterator iter = root.getSubnodes().iterator();
      boolean found = false;
      Node first = null;
      while (iter.hasNext()) {
        first = (Node) iter.next();
        if (first.getClass() == InitialState.class) {
          found = true;
          break;
        }
      }
      if (!found && root.getSubnodes().size() > 0) {
        first = (Node) root.getSubnodes().get(0);
      } else if (!found) {
        first = root;
      }
      return first;
    }

    /**
     * Select node and transition by walking through the statechart.
     */
    private void doNodesAndEdgesWalking() {
//      System.out.println("n: " + n1 + " e: " + e);
      if (n1 == null && e == null) {
        // enter statechart
        CompositeState root = model.getStateChart().getRootNode();
        n1 = getFirstState(root);
      } else {
        switch (key) {
        case UP:
          if (objs.size() > 0) {
            pos++;
            if (pos >= objs.size()) {
              pos = 0;
            }
            Object o = objs.get(pos);
            if (o instanceof Edge) {
              e = (Edge) o;
            } else {
              n1 = (Node) o;
            }
          }
          break;
        case DOWN:
          if (objs.size() > 0) {
            pos--;
            if (pos <= -1) {
              pos = objs.size() - 1;
            }
            Object o = objs.get(pos);
            if (o instanceof Edge) {
              e = (Edge) o;
            } else {
              n1 = (Node) o;
            }
          }
          break;
        case LEFT:
          if (e == null) {
            if (n1.getIncomingTransitions().size() > 0) {
              objs.clear();
              objs.addAll(n1.getIncomingTransitions());
              Collections.sort(objs, edgeSort);
              e = (Edge) objs.get(0);
              n1 = null;
              pos = 0;
            }
          } else if (model.getSelectedTransition() != null) {
            n1 = model.getSelectedTransition().getSource();
            e = null;
            if (n1.getClass() == InitialState.class
                && n1.getParent() != null
                && n1.getParent().getClass() == Region.class) {
              CompositeState root = (CompositeState) n1.getParent().getParent();
              if (root.getSubnodes().size() > 0) {
                Iterator iter = root.getSubnodes().iterator();
                CompositeState region = null;
                objs.clear();
                while (iter.hasNext()) {
                  region = (CompositeState) iter.next();
                  objs.add(getFirstState(region));
                }
                pos = objs.indexOf(n1);
              }
            } else {
              objs.clear();
              pos = -1;
            }
          }
          break;
        case RIGHT:
          if (e == null) {
            if (n1.getOutgoingTransitions().size() > 0) {
              objs.clear();
              objs.addAll(n1.getOutgoingTransitions());
              Collections.sort(objs, edgeSort);
              e = (Edge) objs.get(0);
              n1 = null;
              pos = 0;
            }
          } else {
            n1 = model.getSelectedTransition().getTarget();
            e = null;
            // a transition target should not be an initial state
            // so no choices here (in kit this is impossible but the datastructure
            // doesn't allows this)
```

```
      objs.clear();
      pos = -1;
    }
    break;
  case PG_DOWN:
    if (n1 != null) {
      if (n1 instanceof ANDState) {
        CompositeState root = (CompositeState) n1;
        if (root.getSubnodes().size() > 0) {
          Iterator iter = root.getSubnodes().iterator();
          CompositeState region = null;
          objs.clear();
          while (iter.hasNext()) {
            region = (CompositeState) iter.next();
            objs.add(getFirstState(region));
          }
          n1 = (Node) objs.get(0);
        }
      } else if (n1 instanceof ORState) {
        CompositeState root = (CompositeState) n1;
        n1 = getFirstState(root);
        objs.clear();
      }
    }
    break;
  case PG_UP:
    CompositeState parent;
    if (n1 != null) {
      parent = n1.getParent();
    } else {
      parent = e.getSource().getParent();
    }
    if (parent != null) {
      // if (parent.getClass() == Region.class) {
      //   parent = parent.getParent();
      // }
      n1 = parent;
      e = null;
      objs.clear();
    }
    break;
  default:
  }
}

while (histPos < history.size() - 1) {
  history.remove(history.size() - 1);
}
if (e != null
    && (model.getSelectedTransition() == null
        || !e.getID().equals(model.getSelectedTransition().getID()))) {
  history.add(e);
} else if (n1 != null
    && ((model.getSelectedStates()[0] == null
        || !n1.getID().equals(model.getSelectedStates()[0].getID()))
        || (model.getSelectedStates()[1] == null
        || !n1.getID().equals(model.getSelectedStates()[1].getID())))) {
  history.add(n1);
}
histPos = history.size() - 1;

if (walk1) {
  setFocus = true;
  model.setFocus(n1, model.getSelectedStates()[1], e);
  setFocus = false;
} else {
  setFocus = true;
  model.setFocus(model.getSelectedStates()[0], n1, e);
  setFocus = false;
}
}

/**
 * Node selection method (old way of selection).
 */
private void doNodeWalking() {
  if (n1 == null) {
    // enter statechart
    computeObjsFromComposite(model.getStateChart().getRootNode());
  } else {
    switch (key) {
    case UP:
      pos++;
      if (pos >= objs.size()) {
        pos = 0;
      }
      n1 = (Node) objs.get(pos);
      break;
    case DOWN:
      pos--;
      if (pos < 0) {
        pos = objs.size() - 1;
      }
      n1 = (Node) objs.get(pos);
      break;
    case RIGHT:
      if (n1 instanceof CompositeState) {
        computeObjsFromComposite((CompositeState) n1);
      }
      break;
    case LEFT:
      // System.out.println("n's parent %s " + n1.getParent());
      if (n1.getParent().getParent() != null) {
        computeObjsFromComposite(n1.getParent().getParent());
      }
      break;
    default:
    }
  }
  setFocus = true;
  model.setFocus(n1,
                 model.getSelectedStates()[1],
                 model.getSelectedTransition());
  setFocus = false;
}

/**
 * @param root CompositeState
 */
```

```
  private void computeObjsFromComposite(final CompositeState root) {
    Iterator iter = root.getSubnodes().iterator();
    objs.clear();
    objs.addAll(root.getSubnodes());
    if (root instanceof ANDState && root.getSubnodes().size() > 0) {
      n1 = (Node) root.getSubnodes().get(0);
    } else {
      Node child;
      while (iter.hasNext()) {
        child = (Node) iter.next();
        if (child.getClass() == InitialState.class) {
          n1 = child;
          objs.remove(child);
          objs.add(0, child);
          break;
        }
      }
      if (n1 == null && root.getSubnodes().size() > 0) {
        n1 = (Node) root.getSubnodes().get(0);
      }
    }
    pos = 0;
  }

  /**
   * @param o Observable
   * @param arg Object
   */
  public final void update(final Observable o, final Object arg) {
    if (arg.equals(BrowserModel.NEW_CHART)
//        || arg.equals(BrowserModel.UPDATE_CHART)
        || arg.equals(BrowserModel.RESET)) {
//      System.out.println("Resetting walker");
      setFocus = true;
      model.setFocus(null, null, null);
      setFocus = false;
      n1 = null;
      e = null;
      pos = -1;
      objs.clear();
      history.clear();
      histPos = -1;
    } else if (arg.equals(BrowserModel.FOCUS_CHANGED) && !setFocus
             && (model.getFocus().areNodesChanged()
                 || model.getFocus().hasEdgeChanged())) {
      StatechartFocus focus = model.getFocus();
//      System.out.println("New Focus: " + focus);
      Node node = n1;
      Edge edge = e;
      n1 = null;
      e = null;
      pos = -1;
      objs.clear();
      history.clear();
      histPos = -1;
      if (focus.areNodesChanged()) {
        if (focus.hasNode1Changed() && focus.getNodes()[0] != null
            && focus.getNodes()[0] != node) {
          n1 = focus.getNodes()[0];
//          System.out.println("Setting walker focus to " + n1);
        } else if (focus.hasNode2Changed() && focus.getNodes()[1] != null
            && focus.getNodes()[1] != node) {
          n1 = focus.getNodes()[1];
//          System.out.println("Setting walker focus to " + n1);
        }
      }
      if (n1 == null && focus.hasEdgeChanged() && focus.getEdge() != null) {
//                && focus.getEdge() != edge) {
//        System.out.println("Setting walker focus to " + e);
        e = focus.getEdge();
      }
    }
  }
}
```

F.9.2. MacroStepAction

```
package kiel.browser.controller;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.reflect.InvocationTargetException;
import java.util.Collection;

import javax.swing.SwingUtilities;

import kiel.configMngr.Configuration;
import kiel.configMngr.MacroStep;
import kiel.configMngr.StateActivated;
import kiel.browser.BrowserException;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.browser.view.BrowserToolbar;
import kiel.simulator.Simulator;
import kiel.preferences.LogFile;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.9 £ last modified £Date: 2007/02/08 13:55:18 £
 */
public class MacroStepAction extends Thread implements ActionListener {
  /**
   * The model calling simlator for stepping.
   */
  private BrowserModel model = null;
  /**
   * Sleeptime between microsteps.
   */
  private int sleeptime;
  /**
   * Input events/signals.
   */
  private Collection inputs = null;


  /**
   *
   * @param aModel BrowserModel to get simulator from.
   */
  public MacroStepAction(final BrowserModel aModel) {
    super();
    model = aModel;
    sleeptime = BrowserProperties.getMicroStepSleepTime();
    start();
  }

  /**
   * This threads does the MacroStep automatic stepping.
   * The Thread runs forever. It waits until it's notified to do the automatic
   * stepping.
   */
  public final synchronized void run() {
    final MicroStepAction myAction = new MicroStepAction(model);
    while (true) {
      try {
        wait();
      } catch (InterruptedException ex) {
        System.err.println("Wait returned with an exception " + ex);
      }
      if (myAction.doMicroStep(inputs)) {
        for (int i = model.getMicroStepNumber() + 1;
                    i < model.getMacroStep().getMicroSteps().size(); i++) {
          if (BrowserProperties.animateConfigsInMicroSteps()
              && model.getMacroStep().getMicroSteps().get(i) != null
              && model.getMacroStep().getMicroSteps().get(i).getClass()
              == StateActivated.class
              && !((StateActivated) model.getMacroStep().getMicroSteps().get(i)).
              getConfiguration().equals(model.getConfiguration())) {
            sleeptime = BrowserProperties.getAnimationDuration();
          }
          try {
            sleep(sleeptime);
          } catch (InterruptedException ex1) {
            ex1.printStackTrace();
          }
          sleeptime = BrowserProperties.getMicroStepSleepTime();
          try {
            SwingUtilities.invokeAndWait(new Runnable() {
              public void run() {
                myAction.doMicroStep(inputs);
              }
            });
          } catch (InvocationTargetException ex3) {
            ex3.printStackTrace();
          } catch (InterruptedException ex3) {
            ex3.printStackTrace();
          }
        }
        if (model.getMacroStep().getMicroSteps().size() > 0) {
          // doMicroStep calls perhaps some Swing Classes that must be called
          // from the dispatching thread so just to be sure.
          SwingUtilities.invokeLater(new Runnable() {
            public void run() {
              myAction.doMicroStep(inputs);
            }
          });
        }
      }
      model.quitSimStep();
```

```
    }
  }

  /**
   * Invoked when an action occurs.
   *
   * @param e ActionEvent
   */
  public final synchronized void actionPerformed(final ActionEvent e) {
    model.setMode(BrowserModel.SIMULATION);
    if (e.getActionCommand().equals(BrowserToolbar.CONTINUE)) {
      doMacroStep(null);
    } else {
      doMacroStep(ModelHelper.getInputFromGui());
    }
    if (!BrowserProperties.doStepThrough()) {
      model.quitSimStep();
    }
  }

  /**
   * This is the main macrostep action.
   * This method set also the statechart if the simulator is not run before.
   *
   * @param input Collection with selected input events/signals.
   */
  public final synchronized void doMacroStep(final Collection input) {
    Simulator sim = model.getSimulator();
    if (sim == null) {
      model.startSimulation();
      sim = model.getSimulator();
      if (sim == null) {
        model.sendExceptionMessage(new BrowserException("No simulator found!"));
        return;
      }
    }
    if (!BrowserProperties.doStepThrough()) {
      // Not in delay mode
      model.startComputation();
      MacroStep step = null;
      int i = 0;
      if (model.simulatingMacroSteps()) {
        step = ModelHelper.createMacroStep(input);
        if (step == null) {
          // quiting method because we got no macrostep
          model.completeComputation();
          return;
        }
      } else {
        step = model.getMacroStep();
        i = model.getMicroStepNumber() + 1;
      }
      Configuration conf = ModelHelper.getConfigurationFromMacroStep();
      if (conf != null) {
        model.setConfiguration(conf);
      } else {
        model.setConfiguration(model.getConfiguration());
      }
      if (step.getMicroSteps() != null
          && step.getMicroSteps().size() > 0) {
        for (; i < step.getMicroSteps().size(); i++) {
          model.getSimulatorLogFile().log(LogFile.INFO,
                                          "Step " + (i + 1) + "/"
                                          + step.getMicroSteps().size()
                                          + ": " + step.getMicroSteps().get(i));
        }
      }
      model.setStatus("Step completed");
      ModelHelper.resetEmitsInTables(true);
      ModelHelper.updateTablesFromSimOutput();
      model.sendTableRefresh();
      model.showSimStep();
      // set mactostep on again
      model.setMacroStepSimulation(true);
      model.completeComputation();
    } else {
      inputs = input;
      // let the thread run
      notify();
    }
  }
}
```

F.9.3. MenuController

```
// £Id: MenuController.java,v 1.6 2005/11/28 10:41:25 miwi Exp £
package kiel.browser.controller;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import kiel.browser.model.BrowserModel;

/**
 * <p>Description: An ActionListener for the browser menu.
 * Changes the BrowserModel.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.6 £ last modified £Date: 2005/11/28 10:41:25 £
 */
public class MenuController
    implements ActionListener {
  /**
   * data model for this controller.
   */
  private BrowserModel model;

  /**
   * @param m BrowserModel model to change
   */
  public MenuController(final BrowserModel m) {
    model = m;
  }

  /**
   * @param actionEvent ActionEvent
   */
  public final void actionPerformed(final ActionEvent actionEvent) {
//    String source = actionEvent.getActionCommand();
//    Node rState = model.getStateChart().getRootNode();
//    if (source.equals(BrowserMenuBar.EXPORTJPG)) { // Export as jpg
//      if (rState.getName() != null
//          && !rState.getName().equals("")) {
//        model.exportAsJpg(rState.getName());
//      } else {
//        model.exportAsJpg("MyChart.jpg");
//      }
//    } else if (source.equals(BrowserMenuBar.EXPORTEPS)) { // Exportieren
//      if (rState.getName() != null
//          && !rState.getName().equals("")) {
//        model.exportAsEps(rState.getName());
//      } else {
//        model.exportAsEps("MyChart.eps");
//      }
//    } else if (source.equals(BrowserMenuBar.EXPORTSVG)) { // Exportieren
//      if (rState.getName() != null
//          && !rState.getName().equals("")) {
//        model.exportAsSvg(rState.getName());
//      } else {
//        model.exportAsSvg("MyChart.eps");
//      }
//    } else if (source.startsWith(BrowserMenuBar.SWITCH_CONF)) {
//      String confname = source.substring(BrowserMenuBar.SWITCH_CONF.length());
//      model.getBrowserLogFile().log(LogFile.INFO, "Model changes Config to "
//                                    + confname);
//      if (model.getAllConfigurations() != null) {
//        for (int i = 0; i < model.getAllConfigurations().length; i++) {
//          if (model.getAllConfigurations()[i].toString().equals(confname)) {
//            model.setConfiguration(model.getAllConfigurations()[i]);
//            break;
//          }
//        }
//      } else {
//        model.getBrowserLogFile().log(LogFile.ERROR,
//                        "Call setConfigurations before changing configs");
//      }
//    }
  }
}
```

F.9.4. MicroStepAction

```
package kiel.browser.controller;

import java.awt.event.ActionEvent;
import java.util.Collection;

import javax.swing.AbstractAction;

import kiel.configMngr.MicroStep;
import kiel.configMngr.StateActivated;
import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.simulator.Simulator;
import kiel.preferences.LogFile;
import kiel.browser.BrowserException;

/**
 * <p>Title: Kiel Browser Action Class acts is called every microstep.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.12 £ last modified £Date: 2007/02/08 13:55:18 £
 */
public class MicroStepAction extends AbstractAction {
  /**
   * The model calling simlator for stepping.
   */
  private BrowserModel model = null;

  /**
   * Number of actual step.
   */
  private int actualStep;

  /**
   * Constructor set model and resets step.
   * @param aModel BrowserModel
   */
  public MicroStepAction(final BrowserModel aModel) {
    super();
    model = aModel;
  }


  /**
   * Invoked when an action occurs.
   *
   * @param e ActionEvent
   */
  public final void actionPerformed(final ActionEvent e) {
    model.setMode(BrowserModel.SIMULATION);
    if (model.simulatingMacroSteps()) {
      doMicroStep(ModelHelper.getInputFromGui());
    } else {
      doMicroStep(null);
    }
    model.quitSimStep();
  }

  /**
   * This is the main microstep action.
   * This method set also the statechart if the simulator is not run before.
   *
   * @param input Collection with selected input events/signals.
   * @return true if no error was doing microstep computation.
   */
  public final boolean doMicroStep(final Collection input) {
    Simulator sim = model.getSimulator();
    if (sim == null) {
      model.startSimulation();
      sim = model.getSimulator();
      if (sim == null) {
        model.sendExceptionMessage(new BrowserException("No simulator found!"));
        return false;
      }
    }
    model.startComputation();

    MicroStep m = null;
    if (model.simulatingMacroSteps()) {
      model.getSimulatorLogFile().log(LogFile.INFO,
                      "-->Starting MicroStep with precomputing macroStep<--")
      ;
      if (ModelHelper.createMacroStep(input) == null) {
        // quiting method because we got no macrostep
        model.completeComputation();
        return false;
      }
      if (!BrowserProperties.animateConfigsInMicroSteps()) {
//          System.out.println("Setting CONFIG");
        model.setConfiguration(ModelHelper.getConfigurationFromMacroStep());
      }
      model.setMacroStepSimulation(false);
      m = model.getMicroStep(); // first microstep
      ModelHelper.resetEmitsInTables(true);
      ModelHelper.updateTablesFromSimOutput();
      model.sendTableRefresh();
    } else {
      m = model.nextMicroStep(); // second till last microstep
    }
    if (m != null) {
      if (m.getClass() == StateActivated.class
          && BrowserProperties.animateConfigsInMicroSteps()) {
//          System.out.println("MICROSTEP ACTIVATES A CONFIG");
        model.setConfiguration(((StateActivated) m).getConfiguration());
      }
      actualStep = model.getMicroStepNumber();
```

```
                model.setMacroStepSimulation(true);
                model.setStatus("MicroStep completed");
            }

                model.completeComputation();
                return true;
            }
        }
    }
```

```
        String msg = "Step " + (actualStep + 1)
                + "/" + model.getMacroStep().getMicroSteps().size()
                + " " + m;
        model.getSimulatorLogFile().log(LogFile.INFO, msg);
        model.setStatus(msg);

        model.showSimStep();
    } else {
        model.showSimStep();
```

F.9.5. AddState

F.9.6. PriorityComparator

```
package kiel.browser.controller;

import java.util.Comparator;

import kiel.dataStructure.Edge;
import kiel.dataStructure.Transition;

/**
 * <p>Title: Kiel Browser Controller Class.</p>
 *
 * <p>Description: This is a comparator for edges for sorting them according
 *   their priority.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.1 £ last modified £Date: 2006/01/09 14:04:14 £
 */
public class PriorityComparator implements Comparator {
  /**
   * Compares its two arguments for order.
   *
   * @param o1 the first object to be compared.
   * @param o2 the second object to be compared.
   * @return a negative integer, zero, or a positive integer as the first
   *   argument is less than, equal to, or greater than the second.
   */
  public final int compare(final Object o1, final Object o2) {
    Edge e1 = (Edge) o1;
    Edge e2 = (Edge) o2;
    if (e1 instanceof Transition
        && e2 instanceof Transition) {
      int p1 = ((Transition) e1).getPriority().getValue();
      int p2 = ((Transition) e2).getPriority().getValue();
      if (p1 == 0 && p2 > 0) {
        return 1;
      }
      if (p2 == 0 && p1 > 0) {
        return -1;
      }
      if (p1 < p2) {
        return -1;
      }
      if (p1 > p2) {
        return 1;
      }
      return 0;
    } else if (e1 instanceof Transition
        && !(e2 instanceof Transition)) {
      return -1;
    } else if (e2 instanceof Transition
        && !(e1 instanceof Transition)) {
      return 1;
    }
    return 0;
  }
}
```

F.9.7. TracePlayback

```
package kiel.browser.controller;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.reflect.InvocationTargetException;

import javax.swing.SwingUtilities;

import kiel.browser.BrowserProperties;
import kiel.browser.model.BrowserModel;
import kiel.browser.view.BrowserToolbar;
import kiel.simulationTrace.TraceStep;
import kiel.preferences.LogFile;
import java.util.Iterator;

/**
 * <p>Title: Kiel Browser Controller class.</p>
 *
 * <p>Description: This Class is a thread that shows a complete trace.
 *    For showing the trace it calls for every step the doMacroStep method.
 *    This is a port from an earlier version from Adrian Posor, that doesn't
 *    worked any more with the new BrowserModel.
 * </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.6 £ last modified £Date: 2007/02/08 13:55:19 £
 */
public class TracePlayback extends Thread implements ActionListener {
  /**
   * Model to react on.
   */
  private BrowserModel model;
  /**
   * Flag if doing playback at the moment.
   * Must be an object because of synchronized statements.
   * Needed for stopping.
   */
  private Boolean doPlayBack = Boolean.valueOf(true);

  /**
   * Connects to BrowserModel and starts thread.
   * @param aModel BrowserModel
   */
  public TracePlayback(final BrowserModel aModel) {
    model = aModel;
    start();
  }

  /**
   * Runs the waiting and stepping thread.
   */
  public final synchronized void run() {
    final MacroStepAction myAction = new MacroStepAction(model);
    while (true) {
      try {
        wait();
      } catch (InterruptedException ex) {
        System.err.println("Wait returned with an exception " + ex);
      }
      BrowserProperties.setStepThrough(false);
      while (model.getTrace().hasNextStep() && doPlayBack.booleanValue()) {
        try {
          // gui stuff must run inside awt dispatching thread
          // most toolkits are not thread safe so just to be sure
          SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
              TraceStep step = model.getTrace().getNextStep();
              if (step.containsReset()) {
                model.resetSimulator(false);
                if (step.getStepDataAsCollection().size() > 1) {
                  model.getBrowserLogFile().log(LogFile.ERROR,
                                                "Reset tracestep contains data");
                  Iterator iter = step.getStepDataAsCollection().iterator();
                  while (iter.hasNext()) {
                    model.getBrowserLogFile().log(LogFile.ERROR,
                                                  "Step: " + iter.next());
                  }
                }
                model.showStaticView();
              } else {
                model.setInputTables(step);
                myAction.doMacroStep(step.getStepDataAsCollection());
              }
            }
          });
          sleep(BrowserProperties.getAnimationDuration());
        } catch (InvocationTargetException ex1) {
          ex1.printStackTrace();
        } catch (InterruptedException ex1) {
          ex1.printStackTrace();
        }
      }
      BrowserProperties.setStepThrough(true);
      model.quitSimStep();
    }
  }

  /**
   * Invoked when an action occurs.
   *
   * @param e ActionEvent
   */
  public final void actionPerformed(final ActionEvent e) {
    if (e.getActionCommand().equals(BrowserToolbar.TRACEPLAY)) {
      synchronized (doPlayBack) {   // lock playback flag
        doPlayBack = doPlayBack.valueOf(true);
        synchronized (this) {       // get monitor before notify
          notify();
```

```
                }
            }
                120     }
        }
    } else { // TRACESTOP          // lock playback flag
      synchronized (doPlayBack) {
        doPlayBack = doPlayBack.valueOf(false);
    }
  }
```

F.9.8. LayoutController

```
package kiel.browser.controller;

import java.awt.event.ActionEvent;

import javax.swing.AbstractAction;

import kiel.browser.Browser;
import kiel.preferences.LogFile;
import kiel.util.main.KielLayouter;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.8 £ last modified £Date: 2007/02/08 13:55:18 £
 */
public class LayoutController
    extends AbstractAction {
  /**
   * The layouter name as defined by the layouter package, provided by the
   * Handler.
   */
  private KielLayouter layouter;
  /**
   * Name of layouter.
   */
  private String name;
  /**
   * Logfile got from static Browser.class method.
   */
  private LogFile l = Browser.getModel().getBrowserLogFile();

  /**
   * Creates an object with an layout.gif image.
   * @param aLayouterName .
   */
  public LayoutController(final String aLayouterName) {
    super(aLayouterName);
    name = aLayouterName;
    layouter = Browser.getKielFrame().getKielLayouterByName(aLayouterName);
  }

  /**
   * Invoked when an action occurs.
   *
   * @param event ActionEvent
   */
  public final void actionPerformed(final ActionEvent event) {
    String source = event.getActionCommand();
    if (Browser.getModel().getStateChart() != null) {
      if (layouter == null) {
        l.log(LogFile.ERROR, "Getting layouter " + source + " failed!");
        return;
      }
      if (!name.equals("OriginalLayout")) {
        l.log(LogFile.INFO, "Switching to " + layouter);
        try {
          layouter.setStatechart(Browser.getModel().getStateChart());
        } catch (Exception ex) {
          l.log(LogFile.ERROR, "Layouter throws Exception: " + ex);
//        Browser.getModel().setLayouter(Handler.instance().getPseudoLayouter());
          return;
        }
//        layouter.layoutAllViews(getModel().getConfigs());
//        layouter.layoutView(layouter.getStaticView());
       } else {
        l.log(LogFile.INFO, "Switching to PseudoLayouter");
       }
       if (Browser.getModel().isSimulating()) {
         l.log(LogFile.INFO, "Setting view from configuration");
         layouter.layoutView(layouter.getConfigurationView(
           Browser.getModel().getConfiguration()));
         Browser.getModel().setView(layouter.getConfigurationView(
           Browser.getModel().getConfiguration()));
       } else if (layouter.getStaticView() != null) {
         l.log(LogFile.INFO, "Setting static view");
         layouter.layoutView(layouter.getStaticView());
         Browser.getModel().setView(layouter.getStaticView());
       }
       Browser.getModel().setLayouter(layouter, name);
     }
   }
}
```

F.10. kiel.browser.controller.interpreter

F.10.1. ActionInterpreter

```
package kiel.browser.controller.interpreter;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PushbackReader;
import java.util.Collection;
import java.util.Iterator;

import javax.swing.JComponent;
import javax.swing.KeyStroke;

import kiel.browser.BrowserProperties;
import kiel.browser.controller.interpreter.lexer.Lexer;
import kiel.browser.controller.interpreter.lexer.LexerException;
import kiel.browser.controller.interpreter.node.AAction;
import kiel.browser.controller.interpreter.node.AActionkey;
import kiel.browser.controller.interpreter.node.AActionname;
import kiel.browser.controller.interpreter.node.AKeypropertyKeystroke;
import kiel.browser.controller.interpreter.node.AKeystringKeystroke;
import kiel.browser.controller.interpreter.node.Start;
import kiel.browser.controller.interpreter.parser.Parser;
import kiel.browser.controller.interpreter.parser.ParserException;
import kiel.browser.model.BrowserModel;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Main action interpreter class. Reads the resource. </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £ last modified £Date: 2006/05/21 21:39:33 £
 */
public class ActionInterpreter {
  /**
   *  Buffer for Reader.
   */
  private static final int BUFFER = 1024;
  /**
   * AST 2 Swing Menu.
   */
  private Translation trans;
  /**
   * Model to change.
   */
  private BrowserModel model;

  /**
   * @param aActionComponent JComponent
   * @param aModel BrowserModel
   */
  public ActionInterpreter(final JComponent aActionComponent,
                           final BrowserModel aModel) {
    model = aModel;
    try {
      parseResource();
    } catch (ParserException ex) {
      ex.printStackTrace();
    } catch (LexerException ex) {
      ex.printStackTrace();
    } catch (IOException ex) {
      ex.printStackTrace();
    }
    createInputAndActionMap();
  }

  /**
   * Creates the keybindings.
   */
  private void createInputAndActionMap() {
    Iterator iter = trans.getItemlessActions().iterator();
    AAction action;
    while (iter.hasNext()) {
      action = (AAction) iter.next();
      KeyStroke key = null;
      if (((AActionkey) action.getActionkey()).getKeystroke().getClass()
          == AKeystringKeystroke.class) {
        key = KeyStroke.getKeyStroke(Translation.removeQuotes(
          ((AKeystringKeystroke) ((AActionkey) action.getActionkey()).
           getKeystroke()).
          getString().getText()));
      } else {
        key = BrowserProperties.getKeyStroke(
          ((AKeypropertyKeystroke) ((AActionkey) action.getActionkey()).
           getKeystroke()).getBrowserproperty().getText()
          + ((AKeypropertyKeystroke) ((AActionkey) action.getActionkey()).
           getKeystroke()).getIdentifier().getText());
      }
//      System.out.println("Must register keystroke: " + key + " with action: "
//                         + ((AActionname) action.getActionname()).getString().
//                         getText());
    }
  }

  /**
   * The resource containing the action rules.
   */
  private static final String RESOURCE =
    "/kiel/browser/resources/EditorActions.properties";

  /**
```

```
     *
     * @throws IOException 1
     * @throws LexerException 2
     * @throws ParserException 3
     */
    private void parseResource()
      throws IOException,
             LexerException,
             ParserException {
      Lexer lexer = null;
      Parser parser;
      if (this.getClass().getResourceAsStream(RESOURCE) != null) {
        lexer = new Lexer(new PushbackReader(new InputStreamReader(
          this.getClass().getResourceAsStream(RESOURCE)), BUFFER));
        parser = new Parser(lexer);
        Start tree = parser.parse();
        trans = new Translation(model);
        tree.apply(trans);
      }
    }

    /**
     *
     * @return Collection
     */
    public final Collection getMenus() {
      return trans.getMenus();
    }

    /**
     * reset editing.
     */
    public final void resetEdit() {
      InterpretingAction.resetNodeNumber();
    }
}
```

F.10.2. Translation

```
package kiel.browser.controller.interpreter;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Hashtable;
import java.util.Iterator;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;

import kiel.browser.controller.interpreter.analysis.DepthFirstAdapter;
import kiel.browser.controller.interpreter.node.AAction;
import kiel.browser.controller.interpreter.node.AActionIdentMenuitem;
import kiel.browser.controller.interpreter.node.AActionfile;
import kiel.browser.controller.interpreter.node.AActionid;
import kiel.browser.controller.interpreter.node.AActionkey;
import kiel.browser.controller.interpreter.node.AActionname;
import kiel.browser.controller.interpreter.node.AKeystringKeystroke;
import kiel.browser.controller.interpreter.node.AMenu;
import kiel.browser.controller.interpreter.node.AMenuargument;
import kiel.browser.controller.interpreter.node.ASeparatorMenuitem;
import kiel.browser.model.BrowserModel;
import kiel.browser.controller.interpreter.node.AKeypropertyKeystroke;
import kiel.browser.BrowserProperties;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: Translates the AST to a swing menu.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version $Revision: 1.4 $ last modified $Date: 2006/02/08 07:07:48 $
 */
public class Translation extends DepthFirstAdapter {
  /**
   *  List of actions.
   */
  private Hashtable actions = new Hashtable();
  /**
   *  List of generated menus.
   */
  private ArrayList jmenus = new ArrayList();
  /**
   *  List of actions without menuitem.
   */
  private ArrayList itemlessActions = new ArrayList();
  /**
   * Table of menuitems.
   */
  private Hashtable jmenuitems = new Hashtable();
  /**
   * The action interpreter.
   */
  private InterpretingAction interAction;

  /**
   * @param aModel BrowserModel
   */
  public Translation(final BrowserModel aModel) {
    super();
    interAction = new InterpretingAction(actions, aModel);
  }

  /**
   * @param node AActionid
   */
  public final void inAActionid(final AActionid node) {
    actions.put(node.getIdentifier().getText(), node.parent());
  }

  /**
   * Actual translated menu.
   */
  private JMenu actualMenu = null;

  /**
   * @param node AMenu
   */
  public final void inAMenu(final AMenu node) {
    actualMenu = new JMenu();

    actualMenu.setText(removeQuotes(((AMenuargument) node.getMenuargument()).
                                    getString().getText()));
    jmenus.add(actualMenu);
  }

  /**
   * @param node AActionIdentMenuitem
   */
  public final void outAActionIdentMenuitem(final AActionIdentMenuitem node) {
    JMenuItem item = new JMenuItem();
    AAction action = (AAction) actions.get(node.getIdentifier().getText());
    item.setActionCommand(node.getIdentifier().getText());
    item.setText(removeQuotes(((AActionname) action.getActionname()).
                              getString().getText()));
    if (((AActionkey) action.getActionkey()).getKeystroke().getClass()
        == AKeystringKeystroke.class) {
      item.setAccelerator(KeyStroke.getKeyStroke(removeQuotes(
        ((AKeystringKeystroke) ((AActionkey) action.getActionkey()).
         getKeystroke()).
        getString().getText())));
    } else {
      item.setAccelerator(BrowserProperties.getKeyStroke(
        ((AKeypropertyKeystroke) ((AActionkey) action.getActionkey()).
         getKeystroke()).getBrowserproperty().getText()
        + ((AKeypropertyKeystroke) ((AActionkey) action.getActionkey()).
         getKeystroke()).getIdentifier().getText()));
```

```
        }
        item.addActionListener(interAction);
        actualMenu.add(item);
        jmenuitems.put(item.getActionCommand(), item);
      }

      /**
       * @param node ASeparatorMenuitem
       */
      public final void outASeparatorMenuitem(final ASeparatorMenuitem node) {
        actualMenu.addSeparator();
      }

      /**
       * @param node AMenu
       */
      public final void outAMenu(final AMenu node) {
        actualMenu = null;
      }

      /**
       * @param node AActionfile
       */
      public final void outAActionfile(final AActionfile node) {
        Iterator iter = actions.values().iterator();
        JMenuItem item;
        AAction action;
        while (iter.hasNext()) {
          action = (AAction) iter.next();
          item = (JMenuItem) jmenuitems.get(((AActionid) action.getActionid()).
                                            getIdentifier().getText());
          if (item == null) {
      //        System.out.println("Action: " + ((AActionid)action.getActionid()).
      //                           getIdentifier().getText() + " without menu item");
            itemlessActions.add(action);
          }
        }
      }

      /**
       * @return Collection
       */
      public final Collection getMenus() {
        return jmenus;
      }

      /**
       * @return Collection Actions with an menuItem
       */
      public final Collection getItemlessActions() {
        return itemlessActions;
      }

      /**
       *
       * @param s String
       * @return String
       */
      public static String removeQuotes(final String s) {
        if (s.length() > 2) {
          return s.substring(1, s.length() - 1).trim();
        }
        return "";
      }

      /**
       *
       * @return Hashtable
       */
      public final Hashtable getActionHash() {
        return actions;
      }
    }
```

F.10.3. InterpretingAction

```
package kiel.browser.controller.interpreter;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.ConditionalTransition;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.ForkConnector;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Join;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.State;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.SynchState;
import kiel.dataStructure.Transition;
import kiel.dataStructure.WeakAbortion;
import kiel.browser.Browser;
import kiel.browser.controller.interpreter.node.AAction;
import kiel.browser.controller.interpreter.node.AActionname;
import kiel.browser.controller.interpreter.node.AActionoptions;
import kiel.browser.controller.interpreter.node.AActionrule;
import kiel.browser.controller.interpreter.node.AAndTypes;
import kiel.browser.controller.interpreter.node.AAnotherselectedtransition;
import kiel.browser.controller.interpreter.node.AAnotherstate;
import kiel.browser.controller.interpreter.node.ACompoTypes;
import kiel.browser.controller.interpreter.node.ACompositeState;
import kiel.browser.controller.interpreter.node.AConditionalTtypes;
import kiel.browser.controller.interpreter.node.ACopyallStateortransition;
import kiel.browser.controller.interpreter.node.AFinalTypes;
import kiel.browser.controller.interpreter.node.AFinalandTypes;
import kiel.browser.controller.interpreter.node.AFinalorTypes;
import kiel.browser.controller.interpreter.node.AForallchildsModifier;
import kiel.browser.controller.interpreter.node.AInitialTtypes;
import kiel.browser.controller.interpreter.node.AModifiers;
import kiel.browser.controller.interpreter.node.ANormalTtypes;
import kiel.browser.controller.interpreter.node.AOrTypes;
import kiel.browser.controller.interpreter.node.AParentModifier;
import kiel.browser.controller.interpreter.node.APchoiceTypes;
import kiel.browser.controller.interpreter.node.APdeephistoryTypes;
import kiel.browser.controller.interpreter.node.APdynamicTypes;
import kiel.browser.controller.interpreter.node.APforkTypes;
import kiel.browser.controller.interpreter.node.APhistoryTypes;
import kiel.browser.controller.interpreter.node.APinitialTypes;
import kiel.browser.controller.interpreter.node.APjoinTypes;
import kiel.browser.controller.interpreter.node.APjunctionTypes;
import kiel.browser.controller.interpreter.node.APsuspendTypes;
import kiel.browser.controller.interpreter.node.APsyncTypes;
import kiel.browser.controller.interpreter.node.ARegionStateortransition;
import kiel.browser.controller.interpreter.node.ARegionTypes;
import kiel.browser.controller.interpreter.node.ASelectedident;
import kiel.browser.controller.interpreter.node.ASelectedstate;
import kiel.browser.controller.interpreter.node.ASelectedtransition;
import kiel.browser.controller.interpreter.node.ASimpleState;
import kiel.browser.controller.interpreter.node.ASimpleTypes;
import kiel.browser.controller.interpreter.node.AStateStateortransition;
import kiel.browser.controller.interpreter.node.AStateandtransLvalue;
import kiel.browser.controller.interpreter.node.AStateattributes;
import kiel.browser.controller.interpreter.node.AStrongTtypes;
import kiel.browser.controller.interpreter.node.ATarg;
import kiel.browser.controller.interpreter.node.ATattributes;
import kiel.browser.controller.interpreter.node.ATransition;
import kiel.browser.controller.interpreter.node.ATransitionStateortransition;
import kiel.browser.controller.interpreter.node.ATransonlyLvalue;
import kiel.browser.controller.interpreter.node.ATtypeargument;
import kiel.browser.controller.interpreter.node.ATypeStateattrib;
import kiel.browser.controller.interpreter.node.ATypeTransattrib;
import kiel.browser.controller.interpreter.node.ATypeargument;
import kiel.browser.controller.interpreter.node.AWeakTtypes;
import kiel.browser.controller.interpreter.node.PLvalue;
import kiel.browser.controller.interpreter.node.PModifier;
import kiel.browser.controller.interpreter.node.PState;
import kiel.browser.controller.interpreter.node.PStateattrib;
import kiel.browser.controller.interpreter.node.PStateortransition;
import kiel.browser.controller.interpreter.node.PTransattrib;
import kiel.browser.controller.interpreter.node.PTtypes;
import kiel.browser.controller.interpreter.node.PTypes;
import kiel.browser.model.BrowserModel;
import kiel.browser.model.ModelHelper;
import kiel.util.main.KielLayouter;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
```

```
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.13 £ last modified £Date: 2006/05/21 21:39:33 £
 */
public class InterpretingAction implements ActionListener {
  /**
   *
   */
  private static final int TRANSITION = 1;
  /**
   *
   */
  private static final int WEAK_ABORT = 2;
  /**
   *
   */
  private static final int STRONG_ABORT = 3;
  /**
   *
   */
  private static final int NORMAL_TERM = 4;
  /**
   *
   */
  private static final int CONDITIONAL = 6;
  /**
   *
   */
  private static final int HISTORY = 0;
  /**
   *
   */
  private static final int DEEPHISTORY = 1;
  /**
   *
   */
  private static final int INITIAL = 5;
  /**
   *
   */
  private static final int FORK = 3;
  /**
   *
   */
  private static final int JOIN = 4;
  /**
   *
   */
  private static final int JUNCTION = 2;
  /**
   *
   */
  private static final int SYNC = 6;
  /**
   *
   */
  private static final int CHOICE = 7;
  /**
   *
   */
  private static final int DYNCHOICE = 8;
  /**
   *
   */
  private static final int SUSPEND = 9;
  /**
   *
   */
  private static final int FINAL = 10;
  /**
   *
   */
  private static final int SIMPLE = 11;
  /**
   *
   */
  private static final int COMPOSITE = 12;
  /**
   *
   */
  private static final int ORSTATE = 13;
  /**
   *
   */
  private static final int ANDSTATE = 14;
  /**
   *
   */
  private static final int REGION = 15;
  /**
   *
   */
  private static final int FINALOR = 16;
  /**
   *
   */
  private static final int FINALAND = 17;
  /**
   *
   */
  private BrowserModel model;
  /**
   *
   */
  private Hashtable table;
  /**
   *
   */
  private String n1Id = null;
  /**
   *
   */
  private boolean foundN1 = false;
  /**
   *
   */
  private String n2Id = null;
  /**
   *
   */
  private boolean foundN2 = false;
```

```
    model = aModel;
  }

  /**
   * Invoked when an action occurs.
   *
   * @param ev ActionEvent
   */
  public final void actionPerformed(final ActionEvent ev) {
//   System.out.println("Calling Action: " + ev.getActionCommand());

    n[0] = null;
    n11d = null;

    n[1] = null;
    n21d = null;

  e = null;
  e11d = null;

    AAction action = (AAction) table.get(ev.getActionCommand());
    if (action != null && model.getStateChart() != null
      && (model.getSelectedStates()[0] != null
          || model.getSelectedStates()[1] != null
          || model.getSelectedTransition() != null)) {
      if (action.getActionoptions() != null) {
        options = Translation.removeQuotes(((AActionoptions) action.
                                            getActionoptions()).getString().
                                           getText());
      } else {
        options = "";
      }
//    System.out.println("Analysing Action rules: " + action);
      Iterator iter = action.getActionrule().iterator();
      while (iter.hasNext()) {
        status = "";
        if (consumedRule((AActionrule) iter.next())) {
          model.setChangedObjects(removed.values(),
                                  added.values(),
                                  changed.values(),
                                  this.getClass().getName());
          if (!(options.indexOf("nolayout") > -1)) {
            KielLayouter layouter = Browser.getKielFrame().
                                    getKielLayouterByName(
                                      Browser.getKielFrame().
                                      getDefaultLayouterName());
            try {
              layouter.setStatechart(model.getStateChart());
              layouter.layoutView(layouter.getStaticView());
              model.setLayouter(layouter, Browser.getKielFrame().
                                  getDefaultLayouterName());
              model.showStaticView();
              model.setFocus(null, null, null);
//            System.out.println("Setting new focus!");
              boolean isState1 = true;
              Node[] nodes = new Node[2];
              Edge edge = null;
              for (int i = 0; i < fokus.length; i++) {
//              System.out.println("Setting: " + fokus[i]);
                if (fokus[i] instanceof Node && isState1) {
```

```
  /**
   *
   */
  private String e11d = null;
  /**
   *
   */
  private boolean foundE1 = false;
  /**
   *
   */
  private Node[] n = new Node[2];
  /**
   *
   */
  private Edge e = null;
  /**
   * Size of focus.
   */
  private static final int FOCUS_SIZE = 3;
  /**
   * Actual selected focus.
   */
  private GraphicalObject[] fokus = new GraphicalObject[FOCUS_SIZE];
  /**
   * Position in focus.
   */
  private int fpos = 0;
  /**
   *
   */
  private Hashtable gos = new Hashtable();
  /**
   *
   */
  private Hashtable removed = new Hashtable();
  /**
   *
   */
  private Hashtable added = new Hashtable();
  /**
   *
   */
  private Hashtable changed = new Hashtable();
  /**
   *
   */
  private String status = "";

  /**
   *
   * @param aTable Hashtable
   * @param aModel BrowserModel
   */
  public InterpretingAction(final Hashtable aTable, final BrowserModel aModel) {
    table = aTable;
```

```
                    nodes[0] = (Node) fokus[i];
                    isState1 = !isState1;
                  } else if (fokus[i] instanceof Node && !isState1) {
                    nodes[1] = (Node) fokus[i];
                    isState1 = !isState1;
                  } else if (fokus[i] instanceof Edge) {
                    edge = (Edge) fokus[i];
                  }
                }
                model.setFocus(nodes[0], nodes[1], edge);
              } catch (Exception ex) {
                ex.printStackTrace();
              }
            }
            model.setStatus("Applied action: "
                            + ((AActionname) action.getActionname()).getString());
            break;
          }
          model.setStatus(status);
        }
      }
    }

    /**
     * @param aActionrule AActionrule
     * @return boolean
     */
    private boolean consumedRule(final AActionrule aActionrule) {
      status = "";
      if (testLValue(aActionrule.getLvalue())) {
//      System.out.println("Applying Rule: " + aActionrule);
        ArrayList values = new ArrayList(aActionrule.getSecond().size() + 1);
        values.add(aActionrule.getFirst());
        values.addAll(aActionrule.getSecond());
        return applyRValue(values);
      }
      return false;
    }

    /**
     * @param values ArrayList
     * @return boolean
     */
    private boolean applyRValue(final ArrayList values) {
      boolean retval = true;

//    System.out.println("Changing Nodes " + n[0] + " and " + n[1]
//                       + " according to rule");
//    System.out.println("Changing Transition " + e);
//    System.out.println("--");
      gos.clear();
      added.clear();
      removed.clear();
      changed.clear();
      fpos = 0;
      fokus[0] = null;
      fokus[1] = null;
      fokus[2] = null;
      foundN1 = false;
      foundN2 = false;
      foundE1 = false;
      if (n[0] != null) {
        // in the action rules n[0] is identified by n1Id
        gos.put(n1Id, n[0]);
      }
      if (n[1] != null) {
        // in the action rules n[1] is identified by n2Id
        gos.put(n2Id, n[1]);
      }
      if (e != null) {
        // in the action rules e is identified by e
        gos.put(e1Id, e);
        String[] answers = e1Id.split("->");
        if (answers.length == 2) {
          gos.put(answers[0].trim(), e.getSource());
          gos.put(answers[1].trim(), e.getTarget());
        } else if (answers.length == 1) {
          gos.put(answers[0].trim(), e.getTarget());
        } else {
          throw new RuntimeException("Expected different answers in reg exp");
        }
      }

//    System.out.println("Changing to:");

      CompositeState parent = null;
      if (n[0] != null) {
        parent = n[0].getParent();
      }
      if (n[1] != null && n[1].getParent() != null
          && !n[1].getParent().getID().equals(parent.getID())) {
        status = "Selected states are not in same level";
        return false;
      }
      if (e != null && parent == null) {
        parent = e.getSource().getParent();
      } else if (e != null && parent != null
                 && !e.getSource().getParent().equals(parent.getID())) {
        status = "States and transitions are not in same level";
        return false;
      }
      for (int i = 0; i < values.size(); i++) {
//      System.out.println(values.get(i));
        retval = retval
                 && applyStateOrTransition((PStateortransition) values.get(i),
                                           parent, false);
      }
      if (n[0] != null && !foundN1) {
        removeNode(n[0]);
      }
      if (n[1] != null && !foundN2) {
        removeNode(n[1]);
      }
      if (e != null && !foundE1) {
        removeEdge(e);
      }
//    }
      return retval;
    }
```

```
  /**
   * @param node PStateortransition
   * @param parent CompositeState
   * @param addForAllChilds boolean
   * @return boolean
   */
  private boolean applyStateOrTransition(final PStateortransition node,
                                         final CompositeState parent,
                                         final boolean addForAllChilds) {
    if (node.getClass() == AStateStateortransition.class) {
      PState state = ((AStateStateortransition) node).getState();
      if (state.getClass() == ACompositeState.class) {
        applyACompositeState((ACompositeState) state, parent);
      } else { // simple state
        applyASimpleState((ASimpleState) state, parent, addForAllChilds);
      }
    } else if (node.getClass() == ATransitionStateortransition.class) {
      ATransition trans = (ATransition)
                          ((ATransitionStateortransition) node).getTransition();
      applyATransition(trans, parent);
    }
    return true;
  }

  /**
   *
   * @param state ASimpleState
   * @param parent CompositeState
   * @param forAllChilds boolean
   */
  private void applyASimpleState(final ASimpleState state,
                                 final CompositeState parent,
                                 final boolean forAllChilds) {
    Node node = (Node) gos.get(state.getIdentifier().getText());
    if (node != null) {
      // in this case the node is declared before
      if (node == n[0]) {
        foundN1 = true;
      }
      if (node == n[1]) {
        foundN2 = true;
      }
      Node test = createNodeFromType(getType(state), false, false);
      if (test.getClass() != node.getClass()) {
        // different types so remove add the node
        removed.put(node.getID(), node);
        test.setName(node.getName());
        test.setIDManual(node.getID());
        parent.addSubnode(test);
        Iterator iter = node.getIncomingTransitions().iterator();
        while (iter.hasNext()) {
          Transition t = (Transition) iter.next();
          t.setTarget(test);
        }
        iter = node.getOutgoingTransitions().iterator();
        while (iter.hasNext()) {
          Transition t = (Transition) iter.next();
          t.setSource(test);
        }
        added.put(test.getID(), test);
        gos.remove(state.getIdentifier().getText());
        gos.put(state.getIdentifier().getText(), test);
        node = test;
      }
    } else {
      // this node is not declared before
      int type = getType(state);
      if (forAllChilds) {
        Iterator iter = parent.getSubnodes().iterator();
        int i = 1;
        while (iter.hasNext()) {
          Node sub = (Node) iter.next();
          if (sub instanceof CompositeState) {
            node = createNodeFromType(type, false, false);
            node.setName("" + state.getIdentifier().getText() + i);
            i++;
            ((CompositeState) sub).addSubnode(node);
            added.put(node.getID(), node);
          }
        }
      } else {
        node = createNodeFromType(type, false, false);
        nameNode(node);
//        node.setName(state.getIdentifier().getText());
        parent.addSubnode(node);
        added.put(node.getID(), node);
        gos.put(state.getIdentifier().getText(), node);
      }
    }
    if (state.getSelect() != null) {
      fokus[fpos] = node;
      fpos = (fpos + 1) % FOCUS_SIZE;
    }
  }

  /**
   * @param state ACompositeState
   * @param parent the parent state
   */
  private void applyACompositeState(final ACompositeState state,
                                    final CompositeState parent) {
    Node node = (Node) gos.get(state.getIdentifier().getText());
    CompositeState localParent = null;
    Region reg = null;
    boolean forAllChilds = false;
    if (node != null) {
      // in this case the node is declared before
      if (node == n[0]) {
        foundN1 = true;
      }
      if (node == n[1]) {
        foundN2 = true;
      }
      Node test = createNodeFromType(getType(state), true, hasRegions(state));
      if (test.getClass() != node.getClass()) {
        // different types so remove add the node
//        if (node instanceof CompositeState) {
//          ModelHelper.addSubTree((CompositeState) node, changed);
//        }
        removed.put(node.getID(), node);
```

```
        test.setName(node.getName());
        test.setIDManual(node.getID());
        parent.addSubnode(test);
        Iterator iter = node.getIncomingTransitions().iterator();
        while (iter.hasNext()) {
          Transition t = (Transition) iter.next();
          t.setTarget(test);
        }
        iter = node.getOutgoingTransitions().iterator();
        while (iter.hasNext()) {
          Transition t = (Transition) iter.next();
          t.setSource(test);
        }
        added.put(test.getID(), test);
        if (test instanceof State && node instanceof State) {
          ((State) test).addConstants(((State) node).getConstants());
          ((State) test).addVariables(((State) node).getVariables());
          ((State) test).addLocalEvents(((State) node).getLocalEvents());
          ((State) test).setBindAction(((State) node).getBindAction());
          ((State) test).setEntry(((State) node).getEntry());
          ((State) test).setDoActivity(((State) node).getDoActivity());
          ((State) test).setExit(((State) node).getExit());
          ((State) test).setInternalTransition(
            ((State) node).getInternalTransition());
          ((State) test).setExit(((State) node).getExit());
        }

        gos.remove(state.getIdentifier().getText());
        gos.put(state.getIdentifier().getText(), test);
        if (node instanceof CompositeState
            && containsCopyChilds(state)) {
          // okay childs should be copied
          if (test instanceof ANDState
              && node instanceof ORState) {
            reg = new Region();
//            reg.setName("COPIED REG");
            Iterator iter2 = ((ORState) node).getSubnodes().iterator();
            Node sub = null;
            while (iter2.hasNext()) {
              sub = (Node) iter2.next();
              reg.addSubnode(sub);
            }
            ((CompositeState) test).addSubnode(reg);
            added.put(reg.getID(), reg);
            ModelHelper.addSubTree(reg, changed);
          } else if (test instanceof ORState
                     && node instanceof ANDState) {
            Iterator iter2 = ((ANDState) node).getSubnodes().iterator();
            Iterator iter3 = null;
            Region r = null;
            Node sub = null;
            while (iter2.hasNext()) {
              r = (Region) iter2.next();
              iter3 = ((ANDState) node).getSubnodes().iterator();
              while (iter3.hasNext()) {
                sub = (Node) iter3.next();
                ((ORState) test).addSubnode(sub);
              }
            }
            ModelHelper.addSubTree((CompositeState) test, changed);
          } else if ((test instanceof ANDState
                      && node instanceof ANDState)
                     || (test instanceof ORState
                         && node instanceof ORState)) {
            Iterator iter2 = ((CompositeState) node).getSubnodes().iterator();
            Node sub = null;
            while (iter2.hasNext()) {
              sub = (Node) iter2.next();
              ((CompositeState) test).addSubnode(sub);
            }
            ModelHelper.addSubTree((CompositeState) test, changed);
          }
          node = test;
        }
      } else {
        // this node is not declared before
        int type = getType(state);
        node = createNodeFromType(type, true, hasRegions(state));
        nameNode(node);
//        node.setName(state.getIdentifier().getText());
        parent.addSubnode(node);
        gos.put(state.getIdentifier().getText(), node);
        added.put(node.getID(), node);
      }
      if (state.getSelect() != null) {
        fokus[fpos] = node;
        fpos = (fpos + 1) % FOCUS_SIZE;
      }
      localParent = (CompositeState) node;
      if (state.getModifiers() != null) {
        ArrayList modifiers = new ArrayList();
        modifiers.add(((AModifiers) state.getModifiers()).getFirst());
        modifiers.addAll(((AModifiers) state.getModifiers()).getSec());
        Iterator iter = modifiers.iterator();
        while (iter.hasNext()) {
          PModifier modi = (PModifier) iter.next();
          if (modi.getClass() == AParentModifier.class) {
            localParent = localParent.getParent();
          } else if (modi.getClass() == AForallchildsModifier.class) {
            forAllChilds = true;
          }
        }
      }
      if (forAllChilds) {
        Iterator iter = state.getStateortransition().iterator();
        while (iter.hasNext()) {
          PStateortransition sor = (PStateortransition) iter.next();
          applyStateOrTransition(sor,
                                 localParent,
                                 true);
        }
      } else {
        if (node instanceof ANDState) {
          localParent = reg;
        }
        Iterator iter = state.getStateortransition().iterator();
        while (iter.hasNext()) {
          PStateortransition sor = (PStateortransition) iter.next();
          if (sor.getClass() == ARegionStateortransition.class) {
```

```
            if (parent.getSubnodes().size() > 0 && localParent != reg) {
              added.put(localParent.getID(), localParent);
              ((CompositeState) node).addSubnode(localParent);
            }
            ((ANDState) node).addDelimiterLine(new DelimiterLine());
            localParent = new Region();
//            localParent.setName("NEW REG");
          } else {
            applyStateOrTransition(sor,
                                   localParent,
                                   false);
          }
        }
        if (node instanceof ANDState && localParent.getSubnodes().size() > 0) {
          added.put(node.getID(), node);
          ((CompositeState) node).addSubnode(localParent);
        }
      }
    }

    /**
     * Resets the numbered nodes starting again with "S1".
     */
    public  static final void resetNodeNumber() {
      nodeNumber = 1;
    }

    /**
     * number of node for name.
     */
    private static int nodeNumber = 1;

    /**
     * @param node Node
     */
    private void nameNode(final Node node) {
      node.setName("S" + nodeNumber);
      nodeNumber++;
    }

    /**
     *
     * @param state ACompositeState
     * @return boolean
     */
    private boolean containsCopyChilds(final ACompositeState state) {
      Iterator iter = state.getStateortransition().iterator();
      while (iter.hasNext()) {
        if (iter.next().getClass() == ACopyallStateortransition.class) {
          return true;
        }
      }
      return false;
    }

    /**
     *
     * @param state ACompositeState
     * @return boolean
     */
    private static boolean hasRegions(final ACompositeState state) {
      Iterator iter = state.getStateortransition().iterator();
      while (iter.hasNext()) {
        PStateortransition node = (PStateortransition) iter.next();
        if (node.getClass() == ARegionStateortransition.class) {
          return true;
        }
      }
      return false;
    }

    /**
     * @param type int
     * @param isComposite boolean
     * @param hasRegions boolean
     * @return Node
     */
    private static Node createNodeFromType(final int type,
                                           final boolean isComposite,
                                           final boolean hasRegions) {
      Node node = null;
      if (type == FINAL && isComposite && hasRegions) {
        node = new FinalANDState();
      } else if (type == FINAL && isComposite && !hasRegions) {
        node = new FinalORState();
      } else if (type == FINAL && !isComposite && !hasRegions) {
        node = new FinalSimpleState();
      } else {
        switch (type) {
        case HISTORY:
          node = new History();
          break;
        case DEEPHISTORY:
          node = new DeepHistory();
          break;
        case INITIAL:
          node = new InitialState();
          break;
        case FORK:
          node = new ForkConnector();
          break;
        case JOIN:
          node = new Join();
          break;
        case JUNCTION:
          node = new Junction();
          break;
        case SYNC:
          node = new SynchState();
          break;
        case CHOICE:
          node = new Choice();
          break;
        case DYNCHOICE:
          node = new DynamicChoice();
          break;
        case SUSPEND:
          node = new Suspend();
          break;
        case ORSTATE:
```

```
        node = new ORState();
        break;
      case ANDSTATE:
        node = new ANDState();
        break;
      case REGION:
        node = new Region();
        break;
      case FINALOR:
        node = new FinalORState();
        break;
      case FINALAND:
        node = new FinalANDState();
        break;
      default:
        if (type != FINAL && isComposite && !hasRegions) {
          node = new ORState();
        } else if (type != FINAL && isComposite && hasRegions) {
          node = new ANDState();
        } else {
          node = new SimpleState();
        }
      }
    }
    return node;
  }

  /**
   * @param trans ATransition
   * @param parent CompositeState
   */
  private void applyATransition(final ATransition trans,
                                final CompositeState parent) {
    Transition t = (Transition) gos.get(getId(trans));
    if (t == null) {
      // this is a new transition, so must check the source and target node
      Node source = null;
      if (trans.getSource() != null) {
        // okay this is not an initialarc
        source = (Node) gos.get(((ASelectedident) trans.getSource()).
                                getIdentifier().getText());
        if (source == null) {
          // the source node is also unkown so a new node must be created.
          source = new SimpleState();
          parent.addSubnode(source);
          nameNode(source);
          gos.put(((ASelectedident) trans.getSource()).getIdentifier().
                  getText(), source);
          added.put(source.getID(), source);
        } else if (source == n[0]) {
          foundN1 = true;
        } else if (source == n[1]) {
          foundN2 = true;
        }
      }
      Node target = (Node) gos.get(((ASelectedident) trans.getTarget()).
                                   getIdentifier().getText());
      if (target == null) {
        // the target node is also unkown so a new node must be created.
        target = new SimpleState();
        parent.addSubnode(target);
        nameNode(target);
//        target.setName(((ASelectedident) trans.getTarget()).
//                       getIdentifier().getText());
        gos.put(((ASelectedident) trans.getTarget()).getIdentifier().getText(),
                target);
        added.put(target.getID(), target);
      } else if (target == n[1]) {
        foundN2 = true;
      } else if (target == n[0]) {
        foundN1 = true;
      }
      // since we now have a source and a target node we now
      // can create the transiton
      if (source == null) {
        t = new InitialArc();
        if (model.getStateChart().getModelSource().equals("Esterel Studio")) {
          t.setLabel(new CompoundLabel());
        }
        InitialState init = new InitialState();
        parent.addSubnode(init);
        t.setSource(init);
        t.setTarget(target);
      } else {
        int type = STRONG_ABORT;
        type = getType(trans);
        t = createTransitionFromType(type);
        t.setSource(source);
        t.setTarget(target);
      }
      gos.put(getId(trans), t);
      added.put(t.getID(), t);
    } else {
      // the edge was created before so test if it was selected
      if (e == t) {
        foundE1 = true;
      }
      // test if the type changed
      int type = TRANSITION;
      if (t.getClass() == WeakAbortion.class) {
        type = WEAK_ABORT;
      } else if (t.getClass() == StrongAbortion.class) {
        type = STRONG_ABORT;
      } else if (t.getClass() == NormalTermination.class) {
        type = NORMAL_TERM;
      }
      int newType = getType(trans);
      if (newType != type) {
        // type changed so we remove the transiton and create a new one
        Transition tnew = createTransitionFromType(newType);
        tnew.setLabel(t.getLabel());
        tnew.setPriority(t.getPriority());
        tnew.setSource(t.getSource());
        tnew.setTarget(t.getTarget());
        tnew.setIDManual(t.getID());
        removeEdge(t);
        added.put(tnew.getID(), tnew);
        gos.remove(getId(trans));
        gos.put(getId(trans), tnew);
        t = tnew;
```

```
      }
    }
    assignPriority(t);
    if (trans.getSelect() != null) {
      fokus[fpos] = t;
      fpos = (fpos + 1) % FOCUS_SIZE;
    }
    if (trans.getSource() != null
        && ((ASelectedident) trans.getSource()).getSelect() != null) {
      fokus[fpos] = t.getSource();
      fpos = (fpos + 1) % FOCUS_SIZE;
    }
    if (((ASelectedident) trans.getTarget()).getSelect() != null) {
      fokus[fpos] = t.getTarget();
      fpos = (fpos + 1) % FOCUS_SIZE;
    }
  }

  /**
   * assignPriority
   *
   * @param t Transition
   */
  private static void assignPriority(Transition t) {
    Iterator iter = t.getSource().getOutgoingTransitions().iterator();
    int maxPrio = -1;
    Transition tn = null;
    while (iter.hasNext()) {
      tn = (Transition) iter.next();
      if (tn.getPriority().getValue() > maxPrio) {
        maxPrio = tn.getPriority().getValue();
      }
    }
    t.setPriority(maxPrio+1);
  }

  /**
   * @param type int
   * @return Transition
   */
  private static Transition createTransitionFromType(final int type) {
    Transition t;
    switch (type) {
    case WEAK_ABORT:
      t = new WeakAbortion();
      break;
    case NORMAL_TERM:
      t = new NormalTermination();
      break;
    case STRONG_ABORT:
      t = new StrongAbortion();
      break;
    case INITIAL:
      t = new InitialArc();
      break;
    case CONDITIONAL:
      t = new ConditionalTransition();
      break;
    default:
      // we set strong abortion to default
      t = new StrongAbortion();
    }
    if (Browser.getModel().getStateChart().getModelSource().equals(
                                                        "Esterel Studio")) {
      t.setLabel(new CompoundLabel());
    }
    return t;
  }

  /**
   * @param state PState
   * @return int
   */
  private static int getType(final PState state) {
    ATypeStateattrib types = null;
    int type = SIMPLE;
    Iterator iter = null;
    if (state.getClass() == ACompositeState.class
        && (((ACompositeState) state).getStateattributes()) != null) {
      iter = ((AStateattributes) ((ACompositeState) state).
                getStateattributes()).getStateattrib().iterator();
    } else if (state.getClass() == ASimpleState.class
              && (((ASimpleState) state).getStateattributes()) != null) {
      iter = ((AStateattributes) ((ASimpleState) state).
              getStateattributes()).getStateattrib().iterator();
    }
    if (iter != null) {
      while (iter.hasNext()) {
        PStateattrib attrib = (PStateattrib) iter.next();
        if (attrib.getClass() == ATypeStateattrib.class) {
          types = (ATypeStateattrib) attrib;
        }
      }
      if (types != null) {
        PTypes typ = ((ATypeargument) types.getTypeargument()).getTypes();
        if (typ.getClass() == AAndTypes.class) {
          type = ANDSTATE;
        } else if (typ.getClass() == ACompoTypes.class) {
          type = COMPOSITE;
        } else if (typ.getClass() == AFinalTypes.class) {
          type = FINAL;
        } else if (typ.getClass() == AOrTypes.class) {
          type = ORSTATE;
        } else if (typ.getClass() == APchoiceTypes.class) {
          type = CHOICE;
        } else if (typ.getClass() == APdeephistoryTypes.class) {
          type = DEEPHISTORY;
        } else if (typ.getClass() == APdynamicTypes.class) {
          type = DYNCHOICE;
        } else if (typ.getClass() == APforkTypes.class) {
          type = FORK;
        } else if (typ.getClass() == APhistoryTypes.class) {
          type = HISTORY;
        } else if (typ.getClass() == APinitialTypes.class) {
          type = INITIAL;
        } else if (typ.getClass() == APjoinTypes.class) {
          type = JOIN;
        } else if (typ.getClass() == APjunctionTypes.class) {
          type = JUNCTION;
        } else if (typ.getClass() == APsuspendTypes.class) {
```

```
            type = SUSPEND;
          } else if (typ.getClass() == APsyncTypes.class) {
            type = SYNC;
          } else if (typ.getClass() == ARegionTypes.class) {
            type = REGION;
          } else if (typ.getClass() == AFinalorTypes.class) {
            type = FINALOR;
          } else if (typ.getClass() == AFinalandTypes.class) {
            type = FINALAND;
          }
        }
      }
      return type;
    }

    /**
     * @param trans ATransition
     * @return int
     */
    private int getType(final ATransition trans) {
      int type = TRANSITION;
      if (trans.getTattributes() != null) {
        Iterator iter = ((ATattributes) trans.getTattributes()).
                        getTransattrib().
                        iterator();
        while (iter.hasNext()) {
          PTransattrib att = (PTransattrib) iter.next();
          if (att.getClass() == ATypeTransattrib.class) {
            PTtypes typ = ((ATtypeargument)
                           ((ATypeTransattrib) att).getTtypeargument()).
                          getTtypes();
            if (typ.getClass() == AWeakTtypes.class) {
              type = WEAK_ABORT;
            } else if (typ.getClass() == ANormalTtypes.class) {
              type = NORMAL_TERM;
            } else if (typ.getClass() == AStrongTtypes.class) {
              type = STRONG_ABORT;
            } else if (typ.getClass() == AInitialTtypes.class) {
              type = INITIAL;
            } else if (typ.getClass() == AConditionalTtypes.class) {
              type = CONDITIONAL;
            }
          }
        }
      }
      return type;
    }

    /**
     * @param ed Edge
     */
    private void removeEdge(final Edge ed) {
      removed.put(ed.getID(), ed);
    }

    /**
     * @param node Node
     */
    private void removeNode(final Node node) {
      if (node instanceof CompositeState) {
        ModelHelper.addSubTree((CompositeState) node, removed);
      }
      if (!node.getID().equals(
        model.getStateChart().getRootNode().getID())) {
        // root state could not be removed
        removed.put(node.getID(), node);
      }
      // edges must coming to and from the node must also be removed
      Iterator iter = node.getIncomingTransitions().iterator();
      Edge ed = null;
      while (iter.hasNext()) {
        ed = (Edge) iter.next();
        removed.put(ed.getID(), ed);
      }
      iter = node.getOutgoingTransitions().iterator();
      ed = null;
      while (iter.hasNext()) {
        ed = (Edge) iter.next();
        removed.put(ed.getID(), ed);
      }
    }

    /**
     *
     * @param pLvalue PLvalue
     * @return boolean
     */
    private boolean testLValue(final PLvalue pLvalue) {
      int i = 0;
      boolean correct = true;
      status = "";

      if (pLvalue.getClass() == AStateandtransLvalue.class) {
        AStateandtransLvalue lvalue = (AStateandtransLvalue) pLvalue;
        n = model.getSelectedStates();
        if (model.getSelectedTransition() != null) {
          e = model.getSelectedTransition();
        }
        i = 0;
        if (n[0] == null && n[1] != null) {
          // swap array because we assume the first state in 0
          n[0] = n[1];
          n[1] = null;
        }
        if (lvalue.getSelectedstate() != null && n[i] != null) {
          n1Id = ((ASelectedstate) lvalue.getSelectedstate()).
                 getIdentifier().getText();
          if (!isTypeEqual(((ASelectedstate) lvalue.getSelectedstate()), n[i])) {
            correct = false;
          }
        } else if (lvalue.getSelectedstate() != null && n[i] == null) {
          status = "You need to select a state for this action (1)";
          correct = false;
        }
        i = (i + 1) % 2;
        if (lvalue.getAnotherstate() != null && n[i] != null) {
          n2Id = ((ASelectedstate) ((AAnotherstate) lvalue.getAnotherstate()).
                  getSelectedstate()).getIdentifier().getText();
          if (!isTypeEqual((ASelectedstate) ((AAnotherstate) lvalue.
                                             getAnotherstate()).
```

```
                        getSelectedstate(), n[i])) {
          correct = false;
        }
      } else if (lvalue.getAnotherstate() != null && n[i] == null) {
        status = "You need to select a state for this action (2)";
        correct = false;
      } else if (lvalue.getAnotherstate() == null && n[i] != null) {
        status = "You selected two states , but this action need only one";
        correct = false;
      }

      if (lvalue.getAnotherselectedtransition() != null && e != null) {
        ASelectedtransition trans = (ASelectedtransition)
                                    ((AAnotherselectedtransition)
                                     lvalue.getAnotherselectedtransition()).
                                    getSelectedtransition();
        e1Id = "";
        if (trans.getSource() != null) {
          e1Id += trans.getSource().getText() + "->";
        }
        e1Id += trans.getTarget().getText();
      } else if (lvalue.getAnotherselectedtransition() != null && e == null) {
        correct = false;
      } else if (lvalue.getAnotherselectedtransition() == null && e != null) {
        e = null;
      }

    } else if (pLvalue.getClass() == ATransonlyLvalue.class) {
      ATransonlyLvalue lvalue = (ATransonlyLvalue) pLvalue;
      e = model.getSelectedTransition();
      if (lvalue.getSelectedtransition() != null && e != null) {
        ASelectedtransition trans = (ASelectedtransition)
                                    lvalue.getSelectedtransition();
        e1Id = "";
        if (trans.getSource() != null) {
          e1Id += trans.getSource().getText() + "->";
        }
        e1Id += trans.getTarget().getText();
        correct = true;
      } else if (lvalue.getSelectedtransition() != null && e == null) {
        status = "You need to select a transtion for the action";
        correct = false;
      }
    }

    return correct;
  }

  /**
   * @param node ASelectedstate
   * @param nd Node
   * @return boolean
   */
  private static boolean isTypeEqual(final ASelectedstate node,
                                     final Node nd) {
    if (node.getTarg() != null) {
      ATypeargument type = (ATypeargument) ((ATarg) node.getTarg()).
                           getTypeargument();
      if ((type.getTypes().getClass() == AAndTypes.class
           && nd instanceof ANDState)
          || (type.getTypes().getClass() == ACompoTypes.class
              && nd instanceof CompositeState)
          || (type.getTypes().getClass() == ARegionTypes.class
              && nd instanceof Region)
          || (type.getTypes().getClass() == AFinalTypes.class
              && (nd instanceof FinalState
                  || nd instanceof FinalSimpleState))
          || (type.getTypes().getClass() == AFinalorTypes.class
              && nd instanceof FinalORState)
          || (type.getTypes().getClass() == AFinalandTypes.class
              && nd instanceof FinalANDState)
          || (type.getTypes().getClass() == AOrTypes.class
              && nd instanceof ORState)
          || (type.getTypes().getClass() == APchoiceTypes.class
              && nd instanceof Choice)
          || (type.getTypes().getClass() == APdeephistoryTypes.class
              && nd instanceof DeepHistory)
          || (type.getTypes().getClass() == APdynamicTypes.class
              && nd instanceof DynamicChoice)
          || (type.getTypes().getClass() == APforkTypes.class
              && nd instanceof ForkConnector)
          || (type.getTypes().getClass() == APhistoryTypes.class
              && nd instanceof History)
          || (type.getTypes().getClass() == APinitialTypes.class
              && nd instanceof InitialState)
          || (type.getTypes().getClass() == APjoinTypes.class
              && nd instanceof Join)
          || (type.getTypes().getClass() == APjunctionTypes.class
              && nd instanceof Junction)
          || (type.getTypes().getClass() == APsuspendTypes.class
              && nd instanceof Suspend)
          || (type.getTypes().getClass() == APsuspendTypes.class
              && nd instanceof Suspend)
          || (type.getTypes().getClass() == APsyncTypes.class
              && nd instanceof SynchState)
          || (type.getTypes().getClass() == ASimpleTypes.class
              && nd instanceof SimpleState)) {
        return true;
      } else {
        return false;
      }
    } else {
      return true;
    }
  }

  /**
   * @param trans ATransition
   * @return String
   */
  private static String getId(final ATransition trans) {
    String eid = "";
    if (trans.getSource() != null) {
      eid = ((ASelectedident) trans.getSource()).getIdentifier().getText();
    }
    eid += "->"
      + ((ASelectedident) trans.getTarget()).getIdentifier().getText();
    return eid;
  }

}
```

F.11. kiel.fileInterface.kit

F.11.1. EdgeGenerator

```
package kiel.fileInterface.kit;

import java.io.IOException;
import java.util.ArrayList;

import kiel.dataStructure.Edge;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Node;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Transition;
import kiel.dataStructure.WeakAbortion;
import kiel.graphicalInformations.CurvetoPath;
import kiel.graphicalInformations.EdgeLayoutInformation;
import kiel.graphicalInformations.LabelLayoutInformation;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.MovetoPath;
import kiel.graphicalInformations.PathElement;
import kiel.graphicalInformations.Point;
import kiel.graphicalInformations.QuadraticCurvetoPath;
import kiel.util.EdgeWorker;
import kiel.util.WorkException;
import kiel.dataStructure.Suspension;
import kiel.dataStructure.ConditionalTransition;

/**
 * <p>Kit file interface plugin.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: </p>
 *
 * @author miwi
 * @version 1.0
 */
public class EdgeGenerator implements EdgeWorker {
  /**
   * This is a singleton.
   */
  private static EdgeGenerator instance = null;

  /**
   * not for direct use.
   */
  private EdgeGenerator() {
  }

  /**
   * Singleton.
   * @return EdgeGenerator
   */
  public static EdgeGenerator getInstance() {
    if (instance == null) {
      instance = new EdgeGenerator();
    }
    return instance;
  }

  /**
   *
   * @param e Edge
   * @return boolean
   */
  public static boolean hasAttributtes(final Edge e) {
    return (e instanceof Transition
            && (((Transition) e).getPriority().getValue() >= 1
                || e.getClass() == WeakAbortion.class
                || e.getClass() == StrongAbortion.class
                || e.getClass() == NormalTermination.class
                || e.getClass() == Suspension.class
                || e.getClass() == ConditionalTransition.class
                || KitGenerator.shouldWriteLayoutInformation()
                || !((Transition) e).getLabel().toString().equals("")));
  }

  /**
   * work.
   *
   * @param e Edge
   * @param n Node
   * @throws WorkException ex
   */
  public final void work(final Edge e, final Node n) throws WorkException {
    if (!(e.getSource().equals(n) || e.getTarget().equals(n))) {
      throw new WorkException("Wrong scope of Edge " + e
                              + " is not src/target of " + n);
    }
    StringBuffer buf = createEdge(e);

    try {
      KitGenerator.getWriter().write(buf.toString());
    } catch (IOException ex) {
      throw new WorkException(ex.getMessage());
    }
  }

  public static StringBuffer createEdge(Edge e) {
    StringBuffer buf = new StringBuffer();
    buf.append(NodeGenerator.getSpaces());
    if (e.getClass() == InitialArc.class
        || e.getSource().getClass() == InitialState.class) {
      if (!NodeGenerator.isCreatedThroughEdge(e.getSource())) {
        buf.append(KitGenerator.getReference(e.getSource()));
```

```
      }
      buf.append("->");
      buf.append(KitGenerator.getReference(e.getTarget()));
    } else {
      buf.append(KitGenerator.getReference(e.getSource()));
      buf.append("->");
      buf.append(KitGenerator.getReference(e.getTarget()));
    }
    if (hasAttributtes(e)) {
      buf.append('[');
      if (e instanceof Transition) {
        if (((Transition) e).getPriority().getValue() >= 1) {
          buf.append("priority=");
          buf.append('"');
          buf.append(((Transition) e).getPriority().getValue());
          buf.append('"');
          buf.append(';');
        }
        if (e.getClass() == WeakAbortion.class) {
          buf.append("type=wa;");
        } else if (e.getClass() == StrongAbortion.class) {
          buf.append("type=sa;");
        } else if (e.getClass() == NormalTermination.class) {
          buf.append("type=nt;");
        } else if (e.getClass() == Suspension.class) {
          buf.append("type=sp;");
        } else if (e.getClass() == ConditionalTransition.class) {
          buf.append("type=co;");
        }
        if (!((Transition) e).getLabel().toString().equals("")) {
          buf.append("label=");
          buf.append('"');
          buf.append(((Transition) e).getLabel().toString().trim().replaceAll("\n
 ",
                                                                              "\\\\n"))
 ;
          buf.append('"');
          buf.append(';');
        }
      }
      if (KitGenerator.shouldWriteLayoutInformation()) {
        createLayoutInf(e, buf);
      }
      buf.append(']');
    }
    buf.append(';');
    buf.append('\n');
    return buf;
  }

  /**
   *
   * @param e Edge
   * @param buf StringBuffer
   */
  private static void createLayoutInf(final Edge e, final StringBuffer buf) {
    if (KitGenerator.getActualView() != null
        && KitGenerator.getActualView().getLayoutInformation(e) != null) {
      EdgeLayoutInformation layoutInf
        = KitGenerator.getActualView().getLayoutInformation(e);
      String s = createDotPos(layoutInf.getAllPathElements());
      if (!s.equals("")) {
        buf.append("pos=\"");
        buf.append(s);
        buf.append('"');
        buf.append(';');
      }
      if (e instanceof Transition) {
        LabelLayoutInformation labelLayoutInf
          = KitGenerator.getActualView().getLayoutInformation(
            ((Transition) e).getPriority());
        if (labelLayoutInf != null) {
          buf.append("pp=\"");
          buf.append(labelLayoutInf.getUpperLeft().getX());
          buf.append(',');
          buf.append(labelLayoutInf.getUpperLeft().getY());
          buf.append('"');
          buf.append(';');
        }
        labelLayoutInf
          = KitGenerator.getActualView().getLayoutInformation(
            ((Transition) e).getLabel());
        if (labelLayoutInf != null
            && !((Transition) e).getLabel().toString().equals("")) {
          buf.append("lp=\"");
          buf.append(labelLayoutInf.getUpperLeft().getX());
          buf.append(',');
          buf.append(labelLayoutInf.getUpperLeft().getY());
          buf.append('"');
          buf.append(';');
        }
      }
    }
  }

  /**
   *
   * @param p Point
   * @return String
   */
  private static String dotPoint(final Point p) {
    return "" + p.getX() + "," + p.getY() + "␣";
  }

  /**
   * createDotPos
   *
   * @param arrayList ArrayList
   * @return string in annotated dot format
   */
  private static String createDotPos(final ArrayList arrayList) {
    if (arrayList.size() < 2) {
      System.out.println("Path␣to␣small");
      return "";
    }
    PathElement endPoint = (PathElement) arrayList.get(arrayList.size() - 1);
    StringBuffer buf = new StringBuffer();
    buf.append("e,");
    buf.append(dotPoint(endPoint.getTargetPoint()));
    if (endPoint.getClass() == LinetoPath.class) {
```

```
      arrayList.remove(arrayList.size() - 1);
    }
    PathElement pe = (PathElement) arrayList.get(0);
    if (pe.getClass() != MovetoPath.class) {
      System.out.println("Path must beginn with MoveTo");
      return "";
    }
    buf.append(dotPoint(pe.getTargetPoint()));
    PathElement pe2;
    for (int i = 1; i < arrayList.size(); i++) {
      pe2 = (PathElement) arrayList.get(i);
      if (pe2.getClass() == CurvetoPath.class) {
        buf.append(dotPoint(pe2.getControlPoint1()));
        buf.append(dotPoint(pe2.getControlPoint2()));
        buf.append(dotPoint(pe2.getTargetPoint()));
      } else if (pe2.getClass() == QuadraticCurvetoPath.class) {
        buf.append(dotPoint(pe2.getControlPoint1()));
        buf.append(dotPoint(pe2.getControlPoint1()));
        buf.append(dotPoint(pe2.getTargetPoint()));
      } else if (pe2.getClass() == LinetoPath.class) {
        buf.append(dotPoint(pe2.getTargetPoint()));
        buf.append(dotPoint(pe.getTargetPoint()));
        buf.append(dotPoint(pe2.getTargetPoint()));
      } else {
        System.out.println("Unsupported PathElement");
        return "";
      }
      pe = pe2;
    }
    return buf.toString().trim();
  }
}
```

F.11.2. Kit

```
// £Id: Kit.java,v 1.7 2006/02/08 08:32:00 miwi Exp £
package kiel.fileInterface.kit;

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>Kit Language FileInterface.</p>
 *
 * <p>Kit FileInterface main class</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: CAU Kiel</p>
 *
 * @author <a mailto="miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.7 £
 */
public class Kit extends FileInterface {
  /**
   * Does nothing.
   */
  public Kit() {
  }

  /**
   *
   * @return true if plugin can read charts
   */
  public final boolean canRead() {
    return true;
  }

  /**
   *
   * @return true if plugin can write charts
   */
  public final boolean canWrite() {
    return true;
  }

  /**
   * @return .
   */
  public final String getName() {
    return "KIT Language Plugin";
  }

  /**
   * returns the view from the read file.
   *
   * @return the read view
   * @throws FileInterfaceException some charts don't contain views
   */
  public final View getReadView() throws FileInterfaceException {
    //  throw new FileInterfaceException("Contains only topology");
    return null;
  }

  /**
   * FileFilter that accepts right files.
   *
   * @return the file filter
   */
  public final FileFilter getStateChartFileFilter() {
    return KitFileFilter.getInstance();
  }

  /**
   * reads a statechart from the given file.
   *
   * @param f the file
   * @return the statechart the file conatins
   * @throws FileInterfaceException if the file could not be read
   */
  public final StateChart readStateChartDocument(final File f) throws
    FileInterfaceException {
    return KitParser.parseStateChart(f);
  }

  /**
   * Write the statechart with this view in the given file.
   *
   * @param s the chart
   * @param v the view
   * @param f file to be written
   */
  public final File writeStateChartDocument(final StateChart s,
                                            final View v,
                                            final File f) {
    return KitGenerator.generateChart(s, v, f, false);
  }
}
```

F.11.3. KitFileFilter

```
package kiel.fileInterface.kit;

import java.io.File;
import java.io.IOException;

/**
 * <p>File Interface for kit. </p>
 *
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: </p>
 *
 * @author miwi
 * @version 1.0
 */
public class KitFileFilter extends javax.swing.filechooser.FileFilter {
    /**
     * <code>KielFileFilter</code> is implemented as a singleton.
     */
    private static KitFileFilter instance = null;

    /**
     * @param pathname file to be tested
     * @return true if accepted by this plugin
     */
    public final boolean accept(final File pathname) {
        if (pathname.isDirectory()) {
            return true;
        }
        try {
          pathname.createNewFile();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
        return (pathname.canRead()
                && pathname.getPath().toLowerCase().endsWith(".kit"));
    }

    /**
     * @return Name of file filer -> "Kiel Statecharts"
     */
    public final String getDescription() {
        return "Kit Language";
    }

    /**
     * @return the static instance for this filter
     */
    public static KitFileFilter getInstance() {
        if (instance == null) {
            instance = new KitFileFilter();
        }
        return instance;
    }
}
```

F.11.4. KitGenerator

```
// £Id: KitGenerator.java,v 1.10 2006/04/02 21:38:40 miwi Exp £
package kiel.fileInterface.kit;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Writer;
import java.util.Hashtable;
import java.util.Iterator;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Node;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.graphicalInformations.View;
import kiel.util.TreeHelper;
import kiel.util.WorkException;
import java.util.ArrayList;

/**
 * <p>Kit Language FileInterface.</p>
 *
 * <p>Kit Generator main lcass</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: CAU Kiel</p>
 *
 * @author <a mailto="miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.10 £
 */
public final class KitGenerator {
  /**
   * not for use.
   */
  private KitGenerator() {
  }

  /**
   * writer to write kit to.
   */
  private static Writer writer = new PrintWriter(System.out);
  /**
   * write xkit or kit.
   */
  private static boolean writeViewAnnotations = false;

  /**
   * @return Writer
   */
  public static Writer getWriter() {
    return writer;
  }

  /**
   * @return View
   */
  public static View getActualView() {
    return actualView;
  }

  /**
   * Table mapping nodes to names.
   */
  private static Hashtable nodeToName;
  /**
   * View.
   */
  private static View actualView;
  /**
   * Number.
   */
  private static int nr = 1;

  /**
   * @param start Node
   */
  private static void createNodeToNameTable(final Node start) {
    ArrayList childs = getAllNodes((CompositeState) start);
    childs.add(start);
    Iterator iter = childs.iterator();
    while (iter.hasNext()) {
      Node node = (Node) iter.next();
//      System.err.println("Child: " + node.getName() + " ID: " + node.getID());
      if (node.getName() != null && !node.getName().equals("")) {
        boolean found = false;
        Iterator iter2 = nodeToName.values().iterator();
        if (!isKeyword(node.getName()) && node.getName().indexOf('␣') == -1) {
          while (iter2.hasNext()) {
            String val = (String) iter2.next();
            if (val.equals(node.getName())) {
              found = true;
              break;
            }
          }
        } else {
          found = true;
        }
        if (found) {
          if (node.getID().equals(node.getName())) {
            // we got a problem
            nodeToName.put(node.getID(), node.getID() + "#" + nr);
            nr++;
          } else {
            nodeToName.put(node.getID(), node.getID());
          }
        } else {
          nodeToName.put(node.getID(), node.getName());
        }
      } else {
```

```
        if (nodeToName.containsKey(node.getID())) {
          nodeToName.put(node.getID(), node.getID() + "#" + nr);
          nr++;
        } else {
          nodeToName.put(node.getID(), node.getID());
        }
      }
    }
  }

  private static ArrayList getAllNodes(CompositeState start) {
    ArrayList childs = new ArrayList();
    childs.addAll(start.getSubnodes());
    Iterator iter = start.getSubnodes().iterator();
    while (iter.hasNext()) {
      Node node = (Node) iter.next();
      if (node instanceof CompositeState) {
        childs.addAll(getAllNodes((CompositeState) node));
      }
    }
    return childs;
  }


  /**
   * Test wether string is a keyword.
   *
   * @param string String to test
   * @return boolean true if string is a keyword
   */
  private static boolean isKeyword(final String string) {
    return string.equals("statechart")
      || string.equals("model")
      || string.equals("version")
      || string.equals("input")
      || string.equals("output")
      || string.equals("var")
      || string.equals("label")
      || string.equals("type")
      || string.equals("history")
      || string.equals("hi")
      || string.equals("deephistory")
      || string.equals("dh")
      || string.equals("initial")
      || string.equals("in")
      || string.equals("fork")
      || string.equals("fk")
      || string.equals("join")
      || string.equals("jn")
      || string.equals("junction")
      || string.equals("jc")
      || string.equals("sync")
      || string.equals("sy")
      || string.equals("choice")
      || string.equals("choice")
      || string.equals("dynamicchoice")
      || string.equals("dc")
      || string.equals("suspend")
      || string.equals("sd")
      || string.equals("final")
      || string.equals("doActivity")
      || string.equals("do")
      || string.equals("entryActivity")
      || string.equals("entry")
      || string.equals("exitActivity")
      || string.equals("exit")
      || string.equals("bindAction")
      || string.equals("bind")
      || string.equals("internalTransition")
      || string.equals("localEvent")
      || string.equals("localVariable")
      || string.equals("priority")
      || string.equals("weakAbortion")
      || string.equals("wa")
      || string.equals("strongAbortion")
      || string.equals("sa")
      || string.equals("normaltermination")
      || string.equals("nt")
      || string.equals("integer")
      || string.equals("float")
      || string.equals("double")
      || string.equals("boolean")
      || string.equals("combine")
      || string.equals("with");
  }

  /**
   * @param n Node
   * @return String
   */
  public static String getReference(final Node n) {
    return (String) nodeToName.get(n.getID());
  }

  /**
   * @param chart StateChart
   * @return String
   */
  public static String createStateChartDeclaration(final StateChart chart) {
    StringBuffer buf = new StringBuffer();
    buf.append("statechart ");
    buf.append(getReference(chart.getRootNode()));
    buf.append('[');
    buf.append("model=");
    buf.append('"');
    buf.append(chart.getModelSource());
    buf.append('"');
    buf.append(';');
    buf.append("version=");
    buf.append('"');
    buf.append(chart.getModelVersion());
    buf.append('"');
    buf.append(';');
    buf.append(']');
    buf.append('{');
    buf.append('\n');
    Iterator iter = chart.getOutputEvents().iterator();
    while (iter.hasNext()) {
      buf.append(" output ");
      buf.append(((Event) iter.next()).toDeclaration());
```

```
    buf.append(';');
    buf.append('\n');
  }
  iter = chart.getInputEvents().iterator();
  while (iter.hasNext()) {
    buf.append("␣input␣");
    buf.append(((Event) iter.next()).toDeclaration());
    buf.append(';');
    buf.append('\n');
  }
  iter = chart.getAllVariables().iterator();
  while (iter.hasNext()) {
    buf.append("␣var␣");
    buf.append(((Variable) iter.next()).toDeclaration());
    buf.append(';');
    buf.append('\n');
  }
  return buf.toString();
}

/**
 * @return boolean
 */
public static boolean shouldWriteLayoutInformation() {
  return writeViewAnnotations;
}

/**
 *
 * @param chart StateChart to generate kit file for
 * @param f File to create
 * @param v View to generate
 * @param writeLayout writes v if this is true
 * @return File that was actual created (could be different from f
 *                                       because of different suffix)
 */
public static File generateChart(final StateChart chart,
                                 final View v,
                                 final File f,
                                 final boolean writeLayout) {

  File file = null;
  actualView = v;
  if (v != null && writeLayout) {
    writeViewAnnotations = true;
  } else {
    writeViewAnnotations = false;
  }
  if (f != null) {
    String fName = f.getPath();
    int last = fName.lastIndexOf('.');
    if (last > 1) {
      fName = fName.substring(0, last);
    }
    if (writeViewAnnotations) {
      fName += ".xkit";
    } else {
      fName += ".kit";
    }
    try {
      file = new File(fName);
      writer = new FileWriter(file);
    } catch (IOException ex2) {
      ex2.printStackTrace();
    }
  }
  nr = 1;
  nodeToName = new Hashtable();
  createNodeToNameTable(chart.getRootNode());
  try {
    writer.write(createStateChartDeclaration(chart));
    TreeHelper.walkStatechart(chart, NodeGenerator.getInstance(),
                              EdgeGenerator.getInstance());
    writer.write("};\n");
  } catch (WorkException ex) {
    nodeToName = null;
    actualView = null;
  } catch (IOException ex2) {
    ex2.printStackTrace();
  }
  try {
    writer.flush();
    writer.close();
  } catch (IOException ex1) {
    nodeToName = null;
    actualView = null;
  }
  actualView = null;
  nodeToName = null;
  System.out.println("[Kit␣Plugin]␣Wrote␣file␣:␣" + file);
  return file;
 }
}
```

F.11.5. NodeGenerator

```
package kiel.fileInterface.kit;

import java.io.IOException;
import java.util.Iterator;

import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.ForkConnector;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Join;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.PseudoState;
import kiel.dataStructure.Region;
import kiel.dataStructure.State;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.SynchState;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.graphicalInformations.CompositeStateLayoutInformation;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.util.NodeWorker;
import kiel.util.WorkException;

/**
 * <p>File Interface for kit.</p>
 *
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: </p>
 *
 * @author miwi
 * @version 1.0
 */
public class NodeGenerator implements NodeWorker {
  /**
   * buffer.
   */
  private static StringBuffer sp = new StringBuffer();
  /**
   * instance of this class.
   */
  private static NodeGenerator instance = null;
  /**
   * @return NodeGenerator
   */
  public static NodeGenerator getInstance() {
    if (instance == null) {
      instance = new NodeGenerator();
    }
    return instance;
  }

  /**
   * not for use.
   */
  private NodeGenerator() {
    sp.append('␣');
  }

  /**
   * @return String containing spaces for recursion
   */
  public static String getSpaces() {
    return sp.toString();
  }

  /**
   * @param n Node
   * @return boolean
   */
  public static boolean hasAttributtes(final Node n) {
    return ((!KitGenerator.getReference(n).equals(n.getName())
             && n.getClass() != Region.class)
            || n instanceof PseudoState
            || n.getClass() == FinalState.class
            || n.getClass() == FinalSimpleState.class
            || n.getClass() == FinalANDState.class
            || n.getClass() == FinalORState.class
            || KitGenerator.shouldWriteLayoutInformation()
            || (n instanceof State
                && (!((State) n).getDoActivity().toString().equals("")
                    || !((State) n).getEntry().toString().equals("")
                    || !((State) n).getExit().toString().equals("")
                    || !((State) n).getInternalTransition().getLabel().
                          toString().equals("")
                    || ((State) n).getLocalEvents().size() > 0
                    || ((State) n).getVariables().size() > 0
                    || !((State) n).getBindAction().toString().equals(""))
            ));
  }

  /**
   * @param n Node
   * @return boolean
   */
  public static boolean isCreatedThroughEdge(final Node n) {
    if (n instanceof CompositeState) {
      return false;
    }
    if (n.getClass() == InitialState.class
        && n.getOutgoingTransitions().size() == 1) {
      Iterator iter = n.getParent().getSubnodes().iterator();
      int initNumber = 0;
```

```
      while (iter.hasNext()) {
        if (iter.next().getClass() == InitialState.class) {
          initNumber++;
        }
      }
      if (initNumber > 1) {
        return false;
      }
      return true;
    }
    if (!hasAttributtes(n) && (n.getOutgoingTransitions().size() > 0
                               || n.getIncomingTransitions().size() > 0)) {
      return true;
    }
    return false;
  }

  /**
   * @param n Node
   * @throws WorkException ex
   */
  public final void startNode(final Node n) throws WorkException {
    if (!isCreatedThroughEdge(n)) {
      StringBuffer buf = new StringBuffer();
      buf.append(sp.toString());
      if (n.getClass() == Region.class) {
        if (hasAttributtes(n)) {
          buf.append('<');
          buf.append(KitGenerator.getReference(n));
          buf.append('>');
        }
      } else if (n.getParent() != null) {
        buf.append(KitGenerator.getReference(n));
      }

      // test if there are attributes to show
      if (hasAttributtes(n)) {
        buf.append('[');
        if (!KitGenerator.getReference(n).equals(n.getName())) {
          buf.append("label=");
          buf.append('\"');
          buf.append(n.getName());
          buf.append("\";");
        }
        if (n instanceof PseudoState) {
          buf.append("type=");
          if (n.getClass() == History.class) {
            buf.append("history");
          } else if (n.getClass() == DeepHistory.class) {
            buf.append("deephistory");
          } else if (n.getClass() == InitialState.class) {
            buf.append("initial");
          } else if (n.getClass() == ForkConnector.class) {
            buf.append("fork");
          } else if (n.getClass() == Join.class) {
            buf.append("join");
          } else if (n.getClass() == Junction.class) {
            buf.append("junction");
          } else if (n.getClass() == SynchState.class) {
            buf.append("sync");
          } else if (n.getClass() == Choice.class) {
            buf.append("choice");
          } else if (n.getClass() == DynamicChoice.class) {
            buf.append("dynamicchoice");
          } else if (n.getClass() == Suspend.class) {
            buf.append("suspend");
          } else {
            throw new WorkException("Found new pseudostate " + n);
          }
          buf.append(';');
        } else if (n.getClass() == FinalState.class
                   || n.getClass() == FinalSimpleState.class
                   || n.getClass() == FinalANDState.class
                   || n.getClass() == FinalORState.class) {
          buf.append("type=final;");
        }
        if (n instanceof State) {
          if (!((State) n).getDoActivity().toString().equals("")) {
            buf.append("doActivity=");
            buf.append('\"');
            buf.append(((State) n).getDoActivity().toString().trim());
            buf.append("\";");
          }
          if (!((State) n).getEntry().toString().equals("")) {
            buf.append("entryActivity=");
            buf.append('\"');
            buf.append(((State) n).getEntry().toString().trim());
            buf.append("\";");
          }
          if (!((State) n).getExit().toString().equals("")) {
            buf.append("exitActivity=");
            buf.append('\"');
            buf.append(((State) n).getExit().toString().trim());
            buf.append("\";");
          }
          if (!((State) n).getBindAction().toString().equals("")) {
            buf.append("bindAction=");
            buf.append('\"');
            buf.append(((State) n).getBindAction().toString().trim());
            buf.append("\";");
          }
          if (!((State) n).getInternalTransition().getLabel().toString().
              equals("")) {
            buf.append("internalTransition=");
            buf.append('\"');
            buf.append((((State) n).getInternalTransition()).getLabel().
                       toString().trim());
            buf.append("\";");
          }
          if (((State) n).getLocalEvents().size() > 0) {
            Iterator iter = ((State) n).getLocalEvents().iterator();
            while (iter.hasNext()) {
              buf.append("localEvent=");
              buf.append('\"');
              buf.append(((Event) iter.next()).toDeclaration());
              buf.append("\";");
            }
          }
          if (((State) n).getVariables().size() > 0) {
            Iterator iter = ((State) n).getVariables().iterator();
```

```
            while (iter.hasNext()) {
              buf.append("localVariable=");
              buf.append('\"');
              String decl = ((Variable) iter.next()).toDeclaration();
              buf.append(decl);
              buf.append("\";");
            }
          }
        }
        if (KitGenerator.shouldWriteLayoutInformation()) {
          createLayoutInf(n, buf);
        }
        buf.append(']');
      }
      if (n instanceof CompositeState) {
        sp.append('␣');
        if (n.getClass() != Region.class) {
          buf.append('{');
        }
      } else {
        buf.append(';');
      }
      buf.append('\n');
      try {
        if (!buf.toString().trim().equals("")) {
          KitGenerator.getWriter().write(buf.toString());
        }
      } catch (IOException ex) {
        throw new WorkException(ex.getMessage());
      }
    }
  }

  private void createLayoutInf(Node n, StringBuffer buf) {
    if (KitGenerator.getActualView() != null
        && KitGenerator.getActualView().getLayoutInformation(n) != null) {
      NodeLayoutInformation layoutInf
        = KitGenerator.getActualView().getLayoutInformation(n);
      buf.append("pos=\"");
      buf.append(layoutInf.getXPos());
      buf.append(',');
      buf.append(layoutInf.getYPos());
      buf.append('"');
      buf.append(';');
      buf.append("width=\"");
      buf.append(layoutInf.getWidth());
      buf.append('"');
      buf.append(';');
      buf.append("height=\"");
      buf.append(layoutInf.getHeight());
      buf.append('"');
      buf.append(';');
      if (n instanceof CompositeState &&
          layoutInf.getClass() == CompositeStateLayoutInformation.class
          && ((CompositeStateLayoutInformation) layoutInf).isCollapsed()) {
        buf.append("collapsed=true;");
      }
    }
  }

  /**
   * @param n Node
   * @throws WorkException
   */
  public void endNode(Node n) throws WorkException {
    if (!isCreatedThroughEdge(n)) {
      if (n instanceof CompositeState) {
        sp.deleteCharAt(0);
        StringBuffer buf = new StringBuffer();
        buf.append(sp.toString());
        if (n instanceof CompositeState
            && n.getClass() != Region.class) {
          buf.append('}');
          buf.append(';');
          buf.append('\n');
        } else if (n.getClass() == Region.class) {
          if (n.getParent().getSubnodes().lastIndexOf(n)
              != n.getParent().getSubnodes().size() - 1) {
            buf.append("||");
            buf.append('\n');
          } else {
            // write nothing if last region ending
            return;
          }
        }
        try {
          KitGenerator.getWriter().write(buf.toString());
        } catch (IOException ex) {
          throw new WorkException(ex.getMessage());
        }
      }
    }
  }
}
```

F.11.6. KitProperties

```
// £Id: KitProperties.java,v 1.10 2007/03/08 18:02:41 khe Exp £
package kiel.fileInterface.kit;

import java.io.FileInputStream;
import java.io.IOException;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import kiel.fileInterface.kit.xmlconf.Configuration;
import kiel.preferences.LogFile;
import kiel.preferences.XMLPreferences;

/**
 *
 * <p>Title:.  </p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 *
 * @version £Revision: 1.10 £ last modified £Date: 2007/03/08 18:02:41 £
 */
public final class KitProperties {

  /**
   * The Configuration object, unmarshalled from XML.
   */
  private static Configuration conf;

  /**
   * jaxb context.
   */
  private static JAXBContext jc;

  /**
   * reload preferences.
   * @return success
   */
  public static boolean reload() {
      String userPrefFileName = "NOTFOUND";
      try {
          userPrefFileName = xmlprefs.getXMLPreferencesURL().getPath();
          // create an Unmarshaller
          Unmarshaller u = jc.createUnmarshaller();

          conf = (Configuration) u.unmarshal(
                  new FileInputStream(userPrefFileName));
          return true;
      } catch (JAXBException je) {
          logger.log(LogFile.ERROR, "Error␣parsing␣" + userPrefFileName
                  + ":␣" + je.getMessage());
          return false;
      } catch (IOException ioe) {
          logger.log(LogFile.ERROR, "Error␣opening␣" + userPrefFileName
                  + ":␣" + ioe.getMessage());
          return false;
      }

  }

  /**
   * Store references to xml-related information.
   */
  private static XMLPreferences xmlprefs;

  /**
   * The <code>kiel.util.LogFile</code> object for layouter properties.
   */
  private static LogFile logger;

  static {
      xmlprefs = XMLPreferences.createInstance("kitFileInterface",
              KitProperties.class);
      logger = xmlprefs.getLogger();

      /**
       * Listener to reload the properties.
       * @author khe
       */
      class PreferencesListener implements XMLPreferences.Listener {

          /**
           * reload properties.
           */
          public void preferencesChanged() {
              KitProperties.reload();
          }
      }

      XMLPreferences.addListener(new PreferencesListener());
      try {
        jc = JAXBContext.newInstance("kiel.layouter.xmlconf",
                                     KitProperties.class.getClassLoader());
      } catch (JAXBException je) {
          je.printStackTrace();
      }

      if (!reload()) {
          logger.log(LogFile.ERROR, "User␣Preferences␣for␣module␣layouter␣"
                  + "could␣not␣be␣initialized.␣Aborting!");
          System.exit(1);
      }
  }


  /**
```

```
    * Not for use.
    */
    private KitProperties() {
    }

        // Preferences getter/setter go here.

    120 }
```

F.11.7. Translation

```
package kiel.fileInterface.kit;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.Edge;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.ForkConnector;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Join;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.StringLabel;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.SynchState;
import kiel.dataStructure.Transition;
import kiel.dataStructure.Variable;
import kiel.dataStructure.WeakAbortion;
import kiel.dataStructure.action.ActionsStringLabel;
import kiel.dataStructure.boolexp.BooleanVariable;
import kiel.dataStructure.doubleexp.DoubleConstant;
import kiel.dataStructure.doubleexp.DoubleVariable;
import kiel.dataStructure.eventexp.CombineWithAdd;
import kiel.dataStructure.eventexp.CombineWithMult;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.eventexp.Signal;
import kiel.dataStructure.floatexp.FloatConstant;
import kiel.dataStructure.floatexp.FloatVariable;
import kiel.dataStructure.intexp.IntegerConstant;
import kiel.dataStructure.intexp.IntegerVariable;
import kiel.fileInterface.FileInterfaceModel;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;
import kiel.util.EdgeWorker;
import kiel.preferences.LogFile;
import kiel.util.NodeWorker;
import kiel.util.TreeHelper;
import kiel.util.WorkException;
import kiel.util.declaration.DeclarationException;
import kiel.util.kit.analysis.DepthFirstAdapter;
import kiel.util.kit.node.ABooleanVarEventType;
import kiel.util.kit.node.AChart;
import kiel.util.kit.node.ACompositeState;
import kiel.util.kit.node.ACstate;
import kiel.util.kit.node.ADeclaration;
import kiel.util.kit.node.ADoActionStateargument;
import kiel.util.kit.node.ADoubleVarEventType;
import kiel.util.kit.node.AEntryActionStateargument;
import kiel.util.kit.node.AEvent;
import kiel.util.kit.node.AExitActionStateargument;
import kiel.util.kit.node.AFinalPseudo;
import kiel.util.kit.node.AFirstrarguments;
import kiel.util.kit.node.AFirstsarguments;
import kiel.util.kit.node.AFloatVarEventType;
import kiel.util.kit.node.AInitialTransition;
import kiel.util.kit.node.AInitialvalue;
import kiel.util.kit.node.AInputDeclTyp;
import kiel.util.kit.node.AIntegerAddVarEventType;
import kiel.util.kit.node.AIntegerMultVarEventType;
import kiel.util.kit.node.AIntegerVarEventType;
import kiel.util.kit.node.AInternalTransitionTranstype;
import kiel.util.kit.node.AIodeclaration;
import kiel.util.kit.node.ALabelRegionargument;
import kiel.util.kit.node.ALabelStateargument;
import kiel.util.kit.node.ALabelTransargument;
import kiel.util.kit.node.ALastrargument;
import kiel.util.kit.node.ALastsargument;
import kiel.util.kit.node.AModelChartargument;
import kiel.util.kit.node.ANewRegionElement;
import kiel.util.kit.node.ANormalTranstype;
import kiel.util.kit.node.AOtherTransition;
import kiel.util.kit.node.AOutputDeclTyp;
import kiel.util.kit.node.APchoicePseudo;
import kiel.util.kit.node.APdeephistoryPseudo;
import kiel.util.kit.node.APdynamicPseudo;
import kiel.util.kit.node.APforkPseudo;
import kiel.util.kit.node.APhistoryPseudo;
import kiel.util.kit.node.APinitialPseudo;
import kiel.util.kit.node.APjoinPseudo;
import kiel.util.kit.node.APjunctionPseudo;
import kiel.util.kit.node.APriorityTransargument;
import kiel.util.kit.node.APsuspendPseudo;
import kiel.util.kit.node.APsyncPseudo;
import kiel.util.kit.node.ARargument;
import kiel.util.kit.node.ARegion;
import kiel.util.kit.node.ASargument;
import kiel.util.kit.node.ASimpleState;
import kiel.util.kit.node.AStateElement;
import kiel.util.kit.node.AStrongTranstype;
import kiel.util.kit.node.ATransitionElement;
import kiel.util.kit.node.ATypeStateargument;
import kiel.util.kit.node.AVarDeclTyp;
```

```
import kiel.util.kit.node.AVariable;
import kiel.util.kit.node.AVersionChartargument;
import kiel.util.kit.node.AVtype;
import kiel.util.kit.node.AWeakTranstype;
import kiel.util.kit.node.PDeclTyp;
import kiel.util.kit.node.PVarEventType;
import kiel.util.kit.node.ASuspensionTranstype;
import kiel.util.kit.node.AConditionalTranstype;
import kiel.dataStructure.Suspension;
import kiel.dataStructure.ConditionalTransition;

/**
 * <p>Kiel Browser: FileInterface for kit.</p>
 *
 * <p>Description: AST to kiel translator.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.36 £ last modified £Date: 2007/02/08 13:55:28 £
 */
public class Translation extends DepthFirstAdapter implements EdgeWorker,
  NodeWorker {
  /**
   * Constant.
   */
  private static final int INPUT = 0;
  /**
   * Constant.
   */
  private static final int OUTPUT = 1;
  /**
   * Constant.
   */
  private static final int VARIABLE = 2;
  /**
   * Constant.
   */
  private static final int NO_VALUE = 0;
  /**
   * Constant.
   */
  private static final int INTEGER = 1;
  /**
   * Constant.
   */
  private static final int FLOAT = 2;
  /**
   * Constant.
   */
  private static final int DOUBLE = 3;
  /**
   * Constant.
   */
  private static final int BOOLEAN = 4;
  /**
   * Constant.
   */
  private static final int COMBINE_ADD = 5;
  /**
   * Constant.
   */
  private static final int COMBINE_MULT = 6;
  /**
   * Constant.
   */
  private static final int SIMPLE = 0;
  /**
   * Constant.
   */
  private static final int CHOICE = 1;
  /**
   * Constant.
   */
  private static final int DYNCHOICE = 2;
  /**
   * Constant.
   */
  private static final int FORK = 3;
  /**
   * Constant.
   */
  private static final int HISTORY = 4;
  /**
   * Constant.
   */
  /**
   * Constant.
   */
  private static final int DEEPHISTORY = 5;
  /**
   * Constant.
   */
  private static final int INITIAL = 6;
  /**
   * Constant.
   */
  private static final int JOIN = 7;
  /**
   * Constant.
   */
  private static final int JUNCTION = 8;
  /**
   * Constant.
   */
  private static final int SUSPEND = 9;
  /**
   * Constant.
   */
  private static final int SYNC = 10;
  /**
   * default state type.
   */
  private static int stateType = SIMPLE;
  /**
   * Constant.
   */
  private static final int TRANS = 0;
```

```
    /**
     * Constant.
     */
    private static final int WEAK = 1;
    /**
     * Constant.
     */
240 private static final int STRONG = 2;
    /**
     * Constant.
     */
    private static final int NORMAL = 3;
    /**
     * Constant.
     */
    private static final int INTERNAL = 4;
    /**
250  * Constant.
     */
    private static final int SUSPENSION = 5;
    /**
     * Constant.
     */
    private static final int CONDITIONAL = 6;
    /**
     * Transition type.
     */
260 private static int transitionType = TRANS;
    /**
     *  Label.
     */
    private static String nodeLabel = null;
    /**
     *  Label.
     */
    private static String regionLabel = null;
    /**
270  *  Label.
     */
    private static String edgeLabel = null;
    /**
     *  Flag.
     */
    private static boolean hasRegions = false;
    /**
     *  List of region.
     */
280 private static ArrayList regions;
    /**
     *  Flag.
     */
    private static boolean isFinal = false;
    /**
     *  Chart.
     */
    private static StateChart chart;
    /**
290  *  Flag.
     */
    private static boolean isConverted = false;
    /**
     *  Name to nodes table.
     */
    private static Hashtable states;
    /**
     *  Name to edges table.
     */
300 private static Hashtable transitions;
    /**
     *  id to go table.
     */
    private static ArrayList idtable;
    /**
     * Label.
     */
    private static ActionsStringLabel entryActions = new ActionsStringLabel("");
    /**
310  * List.
     */
    private static ArrayList allEntryAction = new ArrayList();
    /**
     * Label.
     */
    private static ActionsStringLabel exitActions = new ActionsStringLabel("");
    /**
     * List.
     */
320 private static ArrayList allExitAction = new ArrayList();
    /**
     * Label.
     */
    private static ActionsStringLabel doActions = new ActionsStringLabel("");
    /**
     * List.
     */
    private static ArrayList allDoAction = new ArrayList();
    /**
330  * List.
     */
    private static ArrayList isInternal = new ArrayList();
    /**
     * counter.
     */
    private static int regionCounter = 1;
    /**
     * counter.
     */
340 private static int initalCounter = 1;
    /**
     * counter.
     */
    private static int initalArcCounter = 1;
    /**
     * counter.
     */
    private static int transitionCounter = 1;

350 /**
     * inAChart.
     *
```

```
     * @param node AChart
     */
    public final void inAChart(final AChart node) {
      chart = new StateChart();
      isConverted = false;
      states = new Hashtable();
      transitions = new Hashtable();
      idtable = new ArrayList();
      regions = new ArrayList();
      isInternal = new ArrayList();
      transitionType = TRANS;
      stateType = SIMPLE;
      nodeLabel = null;
      edgeLabel = null;
    }

    /**
     * outAChart.
     *
     * @param node AChart
     */
    public final void outAChart(final AChart node) {
      chart.setRootNode((kiel.dataStructure.CompositeState) states.get(
        node.getIdentifier().getText()));
      if (chart.getRootNode() != null) {
        isConverted = true;
        if (!chart.getModelSource().equals("Matlab/Stateflow")) {
          try {
            TreeHelper.walkStatechart(chart, this, this);
          } catch (WorkException ex) {
            ex.printStackTrace();
          }
        }
      }
    }

    /**
     * outAIodeclaration.
     *
     * @param node AIodeclaration
     */
    public final void outAIodeclaration(final AIodeclaration node) {
      int decl = 0;
      int varEv = NO_VALUE;
      double initValue = 0.0;
      boolean hasInit = false;
      PDeclTyp type = node.getDeclTyp();
      if (type != null) {
        if (type.getClass() == AInputDeclTyp.class) {
          decl = INPUT;
        } else if (type.getClass() == AOutputDeclTyp.class) {
          decl = OUTPUT;
        } else if (type.getClass() == AVarDeclTyp.class) {
          decl = VARIABLE;
        }
      }
      if (((ADeclaration) node.getDeclaration()).getVtype() != null) {
        PVarEventType varEvType = ((AVtype)
                                   ((ADeclaration) node.getDeclaration()).
                                   getVtype()).getVarEventType();
        if (varEvType.getClass() == ABooleanVarEventType.class) {
          varEv = BOOLEAN;
        } else if (varEvType.getClass() == ADoubleVarEventType.class) {
          varEv = DOUBLE;
        } else if (varEvType.getClass() == AFloatVarEventType.class) {
          varEv = FLOAT;
        } else if (varEvType.getClass() == AIntegerVarEventType.class) {
          varEv = INTEGER;
        } else if (varEvType.getClass() == AIntegerAddVarEventType.class) {
          varEv = COMBINE_ADD;
        } else if (varEvType.getClass() == AIntegerMultVarEventType.class) {
          varEv = COMBINE_MULT;
        }
      }
      if (((ADeclaration) node.getDeclaration()).getInitialvalue() != null) {
        initValue = Double.parseDouble(((AInitialvalue)
                                        ((ADeclaration) node.getDeclaration()).
                                        getInitialvalue()).getNumber().getText());
        hasInit = true;
      }
      Event ev = null;
      if (decl == INPUT || decl == OUTPUT) {
        switch (varEv) {
        default:
        case NO_VALUE:
          ev = new Signal(((ADeclaration) node.getDeclaration()).
                          getIdentifier().getText());
          break;
        case INTEGER:
          if (hasInit) {
            ev = new IntegerSignal(((ADeclaration) node.getDeclaration()).
                                   getIdentifier().getText(),
                                   new IntegerConstant((int) initValue));
          } else {
            ev = new IntegerSignal(((ADeclaration) node.getDeclaration()).
                                   getIdentifier().getText());
          }
          break;
        case COMBINE_ADD:
          if (hasInit) {
            ev = new CombineWithAdd(((ADeclaration) node.getDeclaration()).
                                    getIdentifier().getText(),
                                    new IntegerConstant((int) initValue));
          } else {
            ev = new CombineWithAdd(((ADeclaration) node.getDeclaration()).
                                    getIdentifier().getText());
          }
          break;
        case COMBINE_MULT:
          if (hasInit) {
            ev = new CombineWithMult(((ADeclaration) node.getDeclaration()).
                                     getIdentifier().getText(),
                                     new IntegerConstant((int) initValue));
          } else {
            ev = new CombineWithMult(((ADeclaration) node.getDeclaration()).
                                     getIdentifier().getText());
          }
          break;
        }
      }
```

```
    Variable v = null;
    switch (decl) {
    case INPUT:
      chart.getInputEvents().add(ev);
      break;
    case OUTPUT:
      chart.getOutputEvents().add(ev);
      break;
    case VARIABLE:
      switch (varEv) {
      default:
      case NO_VALUE:
      case INTEGER:
        if (hasInit) {
          v = new IntegerVariable(((ADeclaration) node.getDeclaration()).
                                  getIdentifier().getText(),
                                  new IntegerConstant((int) initValue));
        } else {
          v = new IntegerVariable(((ADeclaration) node.getDeclaration()).
                                  getIdentifier().getText());
        }
        break;
      case BOOLEAN:
        v = new BooleanVariable(((ADeclaration) node.getDeclaration()).
                                getIdentifier().getText());
        break;
      case DOUBLE:
        if (hasInit) {
          v = new DoubleVariable(((ADeclaration) node.getDeclaration()).
                                 getIdentifier().getText(),
                                 new DoubleConstant((int) initValue));
        } else {
          v = new DoubleVariable(((ADeclaration) node.getDeclaration()).
                                 getIdentifier().getText());
        }
        break;
      case FLOAT:
        if (hasInit) {
          v = new FloatVariable(((ADeclaration) node.getDeclaration()).
                                getIdentifier().getText(),
                                new FloatConstant((int) initValue));
        } else {
          v = new FloatVariable(((ADeclaration) node.getDeclaration()).
                                getIdentifier().getText());
        }
        break;
      }
      chart.addVariable(v);
      break;
    }
  }

  /**
   * localEvents.
   */
  private static ArrayList localEvents = new ArrayList();
  /**
   * allLocals.
   */
  private static ArrayList allLocals = new ArrayList();
  /**
   * localVariables.
   */
  private static ArrayList localVariables = new ArrayList();
  /**
   * allVars.
   */
  private static ArrayList allVars = new ArrayList();

  /**
   * outAEvent.
   *
   * @param node AEvent
   */
  public final void outAEvent(final AEvent node) {
    try {
      localEvents.addAll(kiel.util.declaration.DeclarationParser.parseEvents(
        removeQuotes(node.getString().getText())));
    } catch (DeclarationException ex) {
      localEvents = new ArrayList();
    }
  }

  /**
   * outAVariable.
   *
   * @param node AVariable
   */
  public final void outAVariable(final AVariable node) {
    try {
      localVariables.addAll(
        kiel.util.declaration.DeclarationParser.parseVariables(
          removeQuotes(node.getString().getText())));
    } catch (DeclarationException ex) {
      localVariables = new ArrayList();
    }
  }

  /**
   * outAEntryActionStateargument.
   *
   * @param node AEntryActionStateargument
   */
  public final void outAEntryActionStateargument(
    final AEntryActionStateargument node) {
    entryActions.addAction(removeQuotes(node.getString().getText()));
  }

  /**
   * outAExitActionStateargument.
   *
   * @param node AExitActionStateargument
   */
  public final void outAExitActionStateargument(final AExitActionStateargument
                                                  node) {
    exitActions.addAction(removeQuotes(node.getString().getText()));
  }

  /**
```

```
     * outADoActionStateargument.
     *
     * @param node ADoActionStateargument
     */
    public final void outADoActionStateargument(
      final ADoActionStateargument node) {
      doActions.addAction(removeQuotes(node.getString().getText()));
    }

    /**
     * getStateChart.
     *
     * @return StateChart
     */
    public final StateChart getStateChart() {
      if (isConverted) {
        return chart;
      }
      return null;
    }

    /**
     * removeQuotes.
     *
     * @param s String
     * @return String
     */
    private static String removeQuotes(final String s) {
      if (s.length() > 2) {
        return s.substring(1, s.length() - 1).trim();
      }
      return "";
    }

    /**
     * outALabelStateargument.
     *
     * @param node ALabelStateargument
     */
    public final void outALabelStateargument(final ALabelStateargument node) {
      nodeLabel = removeQuotes(node.getString().getText());
    }

    /**
     * outALabelTransargument.
     *
     * @param node ALabelTransargument
     */
    public final void outALabelTransargument(final ALabelTransargument node) {
      edgeLabel = removeQuotes(node.getString().getText()).replaceAll("\\\\n","\n")
        ;
    }

    /**
     * inASimpleState.
     *
     * @param node ASimpleState
     */
    public final void inASimpleState(final ASimpleState node) {
      stateType = SIMPLE;
      nodeLabel = null;
      isFinal = false;
      newState();
    //   allEntryAction.add(entryActions);
    //   entryActions = new ActionsStringLabel("");
    //   allExitAction.add(exitActions);
    //   exitActions = new ActionsStringLabel("");
    //   allDoAction.add(doActions);
    //   doActions = new ActionsStringLabel("");
    }

    /**
     * outASimpleState.
     *
     * @param node ASimpleState
     */
    public final void outASimpleState(final ASimpleState node) {
      kiel.dataStructure.Node myNode = null;
      switch (stateType) {
      case SIMPLE:
        if (isFinal) {
          myNode = new FinalSimpleState();
        } else {
          myNode = new SimpleState();
        }
        break;
      case CHOICE:
        myNode = new Choice();
        break;
      case DYNCHOICE:
        myNode = new DynamicChoice();
        break;
      case FORK:
        myNode = new ForkConnector();
        break;
      case HISTORY:
        myNode = new History();
        break;
      case DEEPHISTORY:
        myNode = new DeepHistory();
        break;
      case INITIAL:
        myNode = createInitalState();
        break;
      case JOIN:
        myNode = new Join();
        break;
      case JUNCTION:
        myNode = new Junction();
        break;
      case SUSPEND:
        myNode = new Suspend();
        break;
      case SYNC:
        myNode = new SynchState();
        break;
      default:
        myNode = new SimpleState();
      }
      if (nodeLabel != null) {
```

```
      myNode.setName(nodeLabel);
    } else {
      myNode.setName(node.getIdentifier().getText());
    }
    if (localEvents.size() > 0 && myNode instanceof State) {
      ((State) myNode).addLocalEvents(localEvents);
    }
    if (allLocals.size() > 0) {
      localEvents = (ArrayList) allLocals.remove(allLocals.size() - 1);
    }
    if (localVariables.size() > 0 && myNode instanceof State) {
      ((State) myNode).addVariables(localVariables);
    }
    if (allVars.size() > 0) {
      localVariables = (ArrayList) allVars.remove(allVars.size() - 1);
    }
    if (!entryActions.toString().equals("")
        && myNode instanceof State) {
      ((State) myNode).setEntry(entryActions);
    }
    if (allEntryAction.size() > 0) {
      entryActions = (ActionsStringLabel) allEntryAction.remove(
        allEntryAction.size() - 1);
    }
    if (!exitActions.toString().equals("")
        && myNode instanceof State) {
      ((State) myNode).setExit(exitActions);
    }
    if (allExitAction.size() > 0) {
      exitActions = (ActionsStringLabel) allExitAction.remove(
        allExitAction.size() - 1);
    }
    if (!doActions.toString().equals("")
        && myNode instanceof State) {
      ((State) myNode).setDoActivity(doActions);
    }
    if (allDoAction.size() > 0) {
      doActions = (ActionsStringLabel) allDoAction.remove(
        allDoAction.size() - 1);
    }
    addState(node, myNode);
  }

  /**
   * outAPchoicePseudo.
   *
   * @param node APchoicePseudo
   */
  public final void outAPchoicePseudo(final APchoicePseudo node) {
    stateType = CHOICE;
  }

  /**
   * outAPdeephistoryPseudo.
   *
   * @param node APdeephistoryPseudo
   */
  public final void outAPdeephistoryPseudo(final APdeephistoryPseudo node) {
    stateType = DEEPHISTORY;
  }

  /**
   * outAPdynamicPseudo.
   *
   * @param node APdynamicPseudo
   */
  public final void outAPdynamicPseudo(final APdynamicPseudo node) {
    stateType = DYNCHOICE;
  }

  /**
   * outAPforkPseudo.
   *.
   * @param node APforkPseudo
   */
  public final void outAPforkPseudo(final APforkPseudo node) {
    stateType = FORK;
  }

  /**
   * outAPhistoryPseudo.
   *
   * @param node APhistoryPseudo
   */
  public final void outAPhistoryPseudo(final APhistoryPseudo node) {
    stateType = HISTORY;
  }

  /**
   * outAPinitialPseudo.
   *
   * @param node APinitialPseudo
   */
  public final void outAPinitialPseudo(final APinitialPseudo node) {
    stateType = INITIAL;
  }

  /**
   * outAPjoinPseudo.
   *
   * @param node APjoinPseudo
   */
  public final void outAPjoinPseudo(final APjoinPseudo node) {
    stateType = JOIN;
  }

  /**
   * outAPjunctionPseudo.
   *
   * @param node APjunctionPseudo
   */
  public final void outAPjunctionPseudo(final APjunctionPseudo node) {
    stateType = JUNCTION;
  }

  /**
   * outAPsuspendPseudo.
   *
   * @param node APsuspendPseudo
   */
```

```
  public final void outAPsuspendPseudo(final APsuspendPseudo node) {
    stateType = SUSPEND;
  }

  /**
   * outAPsyncPseudo.
   *
   * @param node APsyncPseudo
   */
  public final void outAPsyncPseudo(final APsyncPseudo node) {
    stateType = SYNC;
  }

  /**
   * outAFinalPseudo.
   *
   * @param node AFinalPseudo
   */
  public final void outAFinalPseudo(final AFinalPseudo node) {
    isFinal = true;
  }

  /**
   * inACstate.
   *
   * @param node ACstate
   */
  public final void inACstate(final ACstate node) {
    // push old values
    regions.add(new Boolean(hasRegions));
    hasRegions = false;
    newState();
  }

  /**
   * newCompositestate.
   */
  private void newState() {
    allLocals.add(localEvents);
    localEvents = new ArrayList();
    allVars.add(localVariables);
    localVariables = new ArrayList();
    allEntryAction.add(entryActions);
    entryActions = new ActionsStringLabel("");
    allExitAction.add(exitActions);
    exitActions = new ActionsStringLabel("");
    allDoAction.add(doActions);
    doActions = new ActionsStringLabel("");
  }

  /**
   * createRegion.
   *
   * @return Region
   */
  private static Region createRegion() {
    Region region = new Region();
    region.setIDManual("#" + regionCounter + "#region");
    regionCounter++;
    return region;
  }

  /**
   * createInitalState.
   *
   * @return InitialState
   */
  private static InitialState createInitalState() {
    InitialState init = new InitialState();
    init.setID("initialState#" + initalCounter);
    initalCounter++;
    return init;
  }

  /**
   * outACstate.
   *
   * @param node ACstate
   */
  public final void outACstate(final ACstate node) {
    CompositeState compi = null;
    Region region = null;
    if (hasRegions) {
      if (isFinal(node)) {
        compi = new FinalANDState();
      } else {
        compi = new ANDState();
      }
      region = createRegion();
      region.setParent(compi);
      if (node.getRegion() != null) {
        nameRegion((ARegion) node.getRegion(), region);
      }
      compi.addSubnode(region);
    } else {
      if (isFinal(node)) {
        compi = new FinalORState();
      } else {
        compi = new ORState();
      }
    }
    if (localEvents.size() > 0) {
      compi.addLocalEvents(localEvents);
    }
    if (allLocals.size() > 0) {
      localEvents = (ArrayList) allLocals.remove(allLocals.size() - 1);
    }
    if (localVariables.size() > 0) {
      compi.addVariables(localVariables);
    }
    if (allVars.size() > 0) {
      localVariables = (ArrayList) allVars.remove(allVars.size() - 1);
    }
    if (!entryActions.toString().equals("")) {
      compi.setEntry(entryActions);
    }
    if (allEntryAction.size() > 0) {
      entryActions = (ActionsStringLabel) allEntryAction.remove(allEntryAction.
        size() - 1);
    }
```

```
    if (!exitActions.toString().equals("")) {
      compi.setEntry(exitActions);
    }
    if (allExitAction.size() > 0) {
      exitActions = (ActionsStringLabel) allExitAction.remove(allExitAction.
        size() - 1);
    }
    if (!doActions.toString().equals("")) {
      compi.setEntry(doActions);
    }
    if (allDoAction.size() > 0) {
      doActions = (ActionsStringLabel) allDoAction.remove(allDoAction.size()
        - 1);
    }
    for (int i = 0; i < node.getElement().size(); i++) {
      Object obj = node.getElement().get(i);
      if (obj.getClass() == AStateElement.class) {
        handleStates(compi, region, (AStateElement) obj);
      } else if (obj.getClass() == ATransitionElement.class) {
        handleTransition(compi, region, (ATransitionElement) obj);
      } else if (obj.getClass() == ANewRegionElement.class) {
        region = createRegion();
        region.setParent(compi);
        compi.addSubnode(region);
        // This is for the layouter only
        ((ANDState) compi).addDelimiterLine(new DelimiterLine());
        if (((ANewRegionElement) obj).getRegion() != null) {
          nameRegion((ARegion) ((ANewRegionElement) obj).getRegion(), region);
        }
      }
    }
    nameCompositeState(node, compi);
    addState(node, compi);
    hasRegions = ((Boolean) regions.remove(regions.size() - 1)).booleanValue();
  }

  /**
   * handleTransition.
   *
   * @param compi CompositeState
   * @param region Region
   * @param node ATransitionElement
   */
  private void handleTransition(
    final CompositeState compi,
    final Region region,
    final ATransitionElement node) {

    kiel.dataStructure.Edge e = null;
    if (node.getTransition().getClass()
        == AInitialTransition.class) {
      // initial transition must be handled different because initial states
      // are created implicit
      AInitialTransition enode = (AInitialTransition) node.getTransition();
      e = (kiel.dataStructure.Edge) transitions.get("" + enode.hashCode());
      kiel.dataStructure.Node target;
      // search if the target state is already created
      if (states.containsKey(enode.getTargetState().getText())) {
        target = (kiel.dataStructure.Node) states.get(enode.getTargetState().
          getText());
        if (target.getParent() != null
            && ((compi instanceof ANDState
                 && !target.getParent().getID().equals(region.getID()))
                || (compi instanceof ORState
                    && !target.getParent().getID().equals(compi.getID())))) {
          target = new SimpleState();
          target.setName(enode.getTargetState().getText());
          addState(enode.getTargetState().getText(), target);
        }
      } else {
        // if not create new one as simple
        target = new SimpleState();
        target.setName(enode.getTargetState().getText());
        addState(enode.getTargetState().getText(), target);
      }
      e.setTarget(target);
      e.getSource().setIDManual("#init#" + target.getID());
      e.setIDManual("#initArc#" + target.getID());
      if (compi instanceof ANDState && region != null) {
        addState("#region#" + target.getID(), region);
      }
    } else if (node.getTransition().getClass()
               == AOtherTransition.class) {
      // other transitions are handelt as follows
      AOtherTransition enode = (AOtherTransition) node.getTransition();
      e = (kiel.dataStructure.Edge) transitions.get("" + enode.hashCode());
      kiel.dataStructure.Node src;
      kiel.dataStructure.Node target;

      // test if the states are already created
      // (must be if different than simplestate)
      if (states.containsKey(enode.getTargetState().getText())) {
        target = (kiel.dataStructure.Node) states.get(enode.getTargetState().
          getText());
        if (target.getParent() != null
            && ((compi instanceof ANDState
                 && !target.getParent().getID().equals(region.getID()))
                || (compi instanceof ORState
                    && !target.getParent().getID().equals(compi.getID())))) {
          target = new SimpleState();
          target.setName(enode.getTargetState().getText());
          addState(enode.getTargetState().getText(), target);
        }
      } else {
        target = new SimpleState();
        target.setName(enode.getTargetState().getText());
        addState(enode.getTargetState().getText(), target);
      }
      if (states.containsKey(enode.getSourceState().getText())) {
        src = (kiel.dataStructure.Node) states.get(enode.getSourceState().
          getText());
        if (src.getParent() != null
            && ((compi instanceof ANDState
                 && !src.getParent().getID().equals(region.getID()))
                || (compi instanceof ORState
                    && !src.getParent().getID().equals(compi.getID())))) {
          src = new SimpleState();
          src.setName(enode.getTargetState().getText());
          addState(enode.getTargetState().getText(), src);
        }
```

```
      } else {
        src = new SimpleState();
        src.setName(enode.getSourceState().getText());
        addState(enode.getSourceState().getText(), src);
      }
      if (isInternal.contains(e)) {
        if (src != target) {
          System.err.println("Source and target from internal transition does "
                             + "no match, using source (" + node + ")");
        }
        if (src instanceof State) {
          ((State) src).setInternalTransition((Transition) e);
        } else {
          System.err.println("Only States can have internal transitions ("
                             + node + ")");
        }
      } else {
        e.setTarget(target);
        e.setSource(src);
      }
      String id = src.getID() + "#" + target.getID() + "#";
      if (idtable.contains(id)) {
        id += "" + transitionCounter;
        transitionCounter++;
      }
      e.setIDManual(id);
      idtable.add(id);
    }

    addSubNode(compi, region, e.getTarget());
    addSubNode(compi, region, e.getSource());
  }

  /**
   * handleStates.
   *
   * @param compi CompositeState
   * @param region Region
   * @param node AStateElement
   */
  private void handleStates(final CompositeState compi,
                            final Region region,
                            final AStateElement node) {
    kiel.dataStructure.Node n = null;
    if (node.getState().getClass() == ASimpleState.class) {
      n = (kiel.dataStructure.Node) states.get(((ASimpleState)
                                                node.getState()).
                                               getIdentifier().getText());
    } else if (node.getState().getClass()
               == ACompositeState.class) {
      n = (kiel.dataStructure.Node) states.get(((ACompositeState) node.
                                                getState()).
                                               getIdentifier().getText());
    }
    addSubNode(compi, region, n);
  }

  /**
   * nameCompositeState.
   *
   * @param node ACstate
   * @param compi CompositeState
   */
  private void nameCompositeState(final ACstate node,
                                  final CompositeState compi) {
    String myLabel = null;
    if (node.getSargument() != null) {
      Iterator iter = ((ASargument) node.getSargument()).getFirstsarguments().
                      iterator();
      while (iter.hasNext()) {
        Object obj = ((AFirstsarguments) iter.next()).getStateargument();
        if (obj.getClass() == ALabelStateargument.class) {
          myLabel = removeQuotes(((ALabelStateargument) obj).getString().
                                 getText());
        }
      }
      if (((ASargument) node.getSargument()).getLastsargument() != null
          && ((ALastsargument) ((ASargument) node.getSargument()).
              getLastsargument()).getStateargument().getClass()
          == ALabelStateargument.class) {
        myLabel = removeQuotes(((ALabelStateargument) ((ALastsargument)
          ((ASargument) node.getSargument()).getLastsargument()).
                               getStateargument()).getString().getText());
      }
    }
    if (myLabel != null) {
      compi.setName(myLabel);
    } else {
      if (node.parent().getClass() == ACompositeState.class) {
        compi.setName(((ACompositeState) node.parent()).
                      getIdentifier().getText());
      } else if (node.parent().getClass() == AChart.class) {
        compi.setName(((AChart) node.parent()).
                      getIdentifier().getText());
      }
    }
  }

  /**
   * nameRegion.
   *
   * @param node ARegion
   * @param region Region
   */
  private void nameRegion(final ARegion node, final Region region) {
    String myLabel = null;
    if (node.getRargument() != null) {
      Iterator iter = ((ARargument) node.getRargument()).getFirstrarguments().
                      iterator();
      while (iter.hasNext()) {
        Object obj = ((AFirstrarguments) iter.next()).getRegionargument();
        if (obj.getClass() == ALabelRegionargument.class) {
          myLabel = removeQuotes(((ALabelRegionargument) obj).getString().
                                 getText());
        }
      }
      if (((ARargument) node.getRargument()).getLastrargument() != null
          && ((ALastrargument) ((ARargument) node.getRargument()).
              getLastrargument()).getRegionargument().getClass()
          == ALabelRegionargument.class) {
```

```
        myLabel = removeQuotes(((ALabelRegionargument) ((ALastrargument)
          ((ARargument) node.getRargument()).getLastrargument()).
                              getRegionargument()).getString().getText());
      }
    }
    if (myLabel != null) {
      region.setName(myLabel);
    } else {
      region.setName(node.getIdentifier().getText());
    }
  }

  /**
   * isFinal.
   *
   * @param node ACstate
   * @return boolean
   */
  private boolean isFinal(final ACstate node) {
    boolean found = false;
    if (node.getSargument() != null) {
      Iterator iter = ((ASargument) node.getSargument()).getFirstsarguments().
                      iterator();
      while (iter.hasNext()) {
        Object obj = ((AFirstsarguments) iter.next()).getStateargument();
        if (obj.getClass() == ATypeStateargument.class
            && ((ATypeStateargument) obj).getPseudo().getClass()
            == AFinalPseudo.class) {
          found = true;
          break;
        }
      }
      if (((ASargument) node.getSargument()).getLastsargument() != null
          && ((ALastsargument) ((ASargument) node.getSargument()).
              getLastsargument()).getStateargument().getClass()
          == ATypeStateargument.class
          && ((ATypeStateargument) ((ALastsargument) ((ASargument)
        node.getSargument()).getLastsargument()).getStateargument())
          .getPseudo().getClass() == AFinalPseudo.class) {
        found = true;
      }
    }
    return found;
  }

  /**
   * addSubNode.
   *
   * @param compi CompositeState
   * @param region Region
   * @param n Node
   */
  private static void addSubNode(final CompositeState compi,
                                 final Region region,
                                 final kiel.dataStructure.Node n) {
    if (n != null) {
      if (compi instanceof ANDState) {
        n.setParent(region);
        if (!region.getSubnodes().contains(n)) {
          region.addSubnode(n);
        }
      } else {
        n.setParent(compi);
        if (!compi.getSubnodes().contains(n)) {
          compi.addSubnode(n);
        }
      }
    }
  }

  /**
   * addState.
   *
   * @param node ACstate
   * @param kNode Node
   */
  private static void addState(final ACstate node,
                               final kiel.dataStructure.Node kNode) {
    String name = null;
    if (node.parent().getClass() == ACompositeState.class) {
      name = ((ACompositeState) node.parent()).getIdentifier().getText();
    } else if (node.parent().getClass() == AChart.class) {
      name = ((AChart) node.parent()).getIdentifier().getText();
    } else {
      System.err.println(
        "Warning /!\\ found Compositestate with wrong parent (assuming AChart"
        + " or ACompositeState): " + node.parent().getClass());
      return;
    }
    addState(name, kNode);
  }

  /**
   * addState.
   *
   * @param node ASimpleState
   * @param kNode Node
   */
  private static void addState(final ASimpleState node,
                               final kiel.dataStructure.Node kNode) {
    addState(node.getIdentifier().getText(), kNode);
  }

  /**
   * addState.
   *
   * @param identifier String
   * @param kNode Node
   */
  private static void addState(final String identifier,
                               final kiel.dataStructure.Node kNode) {
    if (states.containsKey(identifier)) {
      System.err.println(
        "Warning /!\\ found state with same name in hashtable " + identifier
        + " " + kNode.getClass() + " " + states.get(identifier).getClass());
      kNode.setID(identifier);
    } else if (identifier.matches("#\\d*#.*")) {
      kNode.setIDManual(identifier);
    } else {
      //      kNode.setID(identifier);
```

```
      kNode.setIDManual(identifier);
    }
    states.put(kNode.getID(), kNode);
  }

  /**
   * outANewRegionElement.
   *
   * @param node ANewRegionElement
   */
  public final void outANewRegionElement(final ANewRegionElement node) {
    hasRegions = true;
  }


  /**
   * priority.
   */
  private static int priority = 0;

  /**
   * outAPriorityTransargument.
   *
   * @param node APriorityTransargument
   */
  public final void outAPriorityTransargument(
    final APriorityTransargument node) {
    priority = Integer.parseInt(removeQuotes(node.getInt().getText()));
  }

  /**
   * outAInitialTransition.
   *
   * @param node AInitialTransition
   */
  public final void outAInitialTransition(final AInitialTransition node) {
    transitionType = TRANS;
    InitialArc initArc = new InitialArc();
    initArc.setIDManual("#" + initalArcCounter + "#initArc");
    initalArcCounter++;
    initArc.setSource(createInitalState());
    addState(initArc.getSource().getID(), initArc.getSource());
    if (transitions.containsKey("" + node.hashCode())) {
      throw new RuntimeException("You declared same transitions");
    }
    transitions.put("" + node.hashCode(), initArc);

    if (priority > 0) {
      initArc.getPriority().setValue(priority);
    }
    priority = 0;
    if (edgeLabel != null) {
      initArc.setLabel(new StringLabel(edgeLabel));
    }
    edgeLabel = null;
  }

  /**
   * outAOtherTransition.
   *
   * @param node AOtherTransition
   */
  public final void outAOtherTransition(final AOtherTransition node) {
    Transition trans;
    switch (transitionType) {
    case WEAK:
      trans = new WeakAbortion();
      break;
    case STRONG:
      trans = new StrongAbortion();
      break;
    case NORMAL:
      trans = new NormalTermination();
      break;
    case SUSPENSION:
      trans = new Suspension();
      break;
    case CONDITIONAL:
      trans = new ConditionalTransition();
      break;
    case INTERNAL:
      trans = new Transition();
      isInternal.add(trans);
      break;
    default:
      trans = new Transition();
      break;
    }
    if (edgeLabel != null) {
      trans.setLabel(new StringLabel(edgeLabel));
    }
    if (transitions.containsKey("" + node.hashCode())) {
      throw new RuntimeException("You declared same transitions: " + node
                                 + " @" + node.hashCode());
    }
    transitions.put("" + node.hashCode(), trans);
    transitionType = TRANS;
    edgeLabel = null;
    if (priority > 0) {
      trans.getPriority().setValue(priority);
    }
    priority = 0;
  }

  /**
   * outAWeakTranstype.
   *
   * @param node AWeakTranstype
   */
  public final void outAWeakTranstype(final AWeakTranstype node) {
    transitionType = WEAK;
  }

  /**
   * outAStrongTranstype.
   *
   * @param node AStrongTranstype
   */
  public final void outAStrongTranstype(final AStrongTranstype node) {
    transitionType = STRONG;
```

```
    }

    /**
     * outANormalTranstype.
     *
     * @param node ANormalTranstype
     */
    public final void outANormalTranstype(final ANormalTranstype node) {
      transitionType = NORMAL;
    }

    /**
     *
     * @param node ASuspensionTranstype
     */
    public final void outASuspensionTranstype(final ASuspensionTranstype node) {
      transitionType = SUSPENSION;
    }

    /**
     *
     * @param node AConditionalTranstype
     */
    public final void outAConditionalTranstype(final AConditionalTranstype node) {
      transitionType = CONDITIONAL;
    }

    /**
     * outAInternalTransitionTranstype.
     *
     * @param node AInternalTransitionTranstype
     */
    public final void outAInternalTransitionTranstype(final
      AInternalTransitionTranstype node) {
      transitionType = INTERNAL;
    }


    /**
     * outAModelChartargument.
     *
     * @param node AModelChartargument
     */
    public final void outAModelChartargument(final AModelChartargument node) {
      chart.setModelSource(removeQuotes(node.getString().getText()));
    }

    /**
     * outAVersionChartargument.
     *
     * @param node AVersionChartargument
     */
    public final void outAVersionChartargument(final AVersionChartargument node) {
      chart.setModelVersion(removeQuotes(node.getString().getText()));
    }

    /**
     * work.
     *
     * @param e Edge
     * @param n Node
     * @throws WorkException ex
     */
    public final void work(final Edge e, final Node n) throws WorkException {
      if (e instanceof Transition) {
        Transition trans = (Transition) e;
        if (trans.getLabel() != null
            && trans.getLabel().getClass() == StringLabel.class) {
          convertStringLabel(trans);
        }
      }
    }

    /**
     * convertStringLabel.
     *
     * @param trans Transition
     * @throws WorkExceptions
     */
    private final void convertStringLabel(final Transition trans)
        throws  WorkException {
      String label = ((StringLabel) trans.getLabel()).toString().trim();
      if (!label.equals("")) {
//      System.err.println("Converting label: " + label);
        boolean hasTrigger = false;
        boolean hasAction = false;
        boolean hasCondition = false;
        boolean isImmediate = false;
        if (label.matches("[^\\s\\[/]+.*(\\[.*\\])?(/.*)?")) {
          hasTrigger = true;
        }
        if (label.matches(".*/.*")) {
          hasAction = true;
        }
        if (label.matches(".*\\[.*\\].*")) {
          hasCondition = true;
        }
        if (label.matches("\\s*#.*(\\[.*\\])?(/.*)?")) {
          isImmediate = true;
        }
        String[] answers = label.split("[#/\\[\\]]");
//      System.err.println("Splitting label " + label);
//      System.err.println("Which has " + ((hasTrigger) ? "Trigger, " : "")
//                                      + ((hasCondition) ? "Condition, " : "")
//                                      + ((hasAction) ? "Action " : "")
//                                      + "and is " + ((isImmediate) ? "" : "not ")
//                                      + "immediate");
        if (isImmediate && answers.length > 1 &&
            !answers[0].trim().equals("")) {
          throw new WorkException("Label " + label
                                  + " cannot be converted to CompoundLabel");
        }
        if (isImmediate && !hasTrigger) {
          throw new WorkException("Label " + label
                                  + " is immediate but has no trigger");
        }
        if (answers.length >= 5) {
          throw new WorkException("Label " + label
                                  + " has wrong format:"
                                  + " (trigger)? ([condition])? (/effect)?");
```

```
      }
      CompoundLabel aComLabel = new CompoundLabel();
      CompoundLabelParser.setStateChart(chart);
      CompoundLabelParser.setCompositeState(parent);
      ArrayList parts = new ArrayList();
      for (int i = 0; i < answers.length; i++) {
        if (!answers[i].trim().equals("")) {
          parts.add(answers[i].trim());
        }
      }
      try {
        if (hasTrigger) {
          aComLabel.setTrigger(
            CompoundLabelParser.parseDelayExpression(
              ((String) parts.get(0)).trim(), isImmediate));
          parts.remove(0);
        }
        if (hasCondition) {
          aComLabel.setCondition(
            CompoundLabelParser.parseBooleanExpression(
              ((String) parts.get(0)).trim()));
          parts.remove(0);
        }
        if (hasAction) {
          aComLabel.setEffect(
            CompoundLabelParser.parseActions(
              ((String) parts.get(0)).trim()));
        }
        trans.setLabel(aComLabel);
      } catch (CompoundLabelException ex) {
        if (parts.size() > 0) {
          FileInterfaceModel.getFileInterfaceLog().log(LogFile.ERROR,
            "KitFile Translation Edge Label Parsing ("
            + ((String) parts.get(0)).trim() + "): " + ex);
        }
      } catch (Exception ex) {
        if (parts.size() > 0) {
          FileInterfaceModel.getFileInterfaceLog().log(LogFile.ERROR,
            "KitFile Translation Edge Label Parsing ("
            + ((String) parts.get(0)).trim() + "): " + ex);
        }
        ex.printStackTrace();
      }
    } else {
     trans.setLabel(new CompoundLabel());
    }
  }

  /**
   *
   */
  private CompositeState parent;

  /**
   *
   * @param n Node
   * @throws WorkException
   */
  public final void startNode(final Node n) throws WorkException {
    if (n instanceof State) {
      State s = (State) n;
      CompoundLabelParser.setStateChart(chart);
      CompoundLabelParser.setCompositeState(parent);
      if (s.getEntry() != null
          && s.getEntry().getClass() == ActionsStringLabel.class) {
        try {
          s.setEntry(CompoundLabelParser.parseActions(s.getEntry().toString()));
        } catch (CompoundLabelException ex) {
          FileInterfaceModel.getFileInterfaceLog().log(LogFile.ERROR,
            "KitFile Translation On Entry Action Parsing: " + ex.toString());
        }
      }
      if (s.getExit() != null
          && s.getExit().getClass() == ActionsStringLabel.class) {
        try {
          s.setExit(CompoundLabelParser.parseActions(s.getExit().toString()));
        } catch (CompoundLabelException ex) {
          FileInterfaceModel.getFileInterfaceLog().log(LogFile.ERROR,
            "KitFile Translation On Exit Action Parsing: " + ex.toString());
        }
      }
      if (s.getDoActivity() != null
          && s.getDoActivity().getClass() == ActionsStringLabel.class) {
        try {
          s.setDoActivity(CompoundLabelParser.parseActions(s.getDoActivity().
            toString()));
        } catch (CompoundLabelException ex) {
          FileInterfaceModel.getFileInterfaceLog().log(LogFile.ERROR,
            "KitFile Translation Do Activity Parsing: " + ex.toString());
        }
      }
      if (s.getInternalTransition() != null
          && s.getInternalTransition().getLabel() != null
          && s.getInternalTransition().getLabel().getClass()
          == StringLabel.class) {
        convertStringLabel(s.getInternalTransition());
      }
    }
    if (n instanceof CompositeState) {
      parent = (CompositeState) n;
    }
  }

  /**
   *
   * @param n Node
   * @throws WorkException
   */
  public final void endNode(final Node n) throws WorkException {
  }

}
```

F.11.8. KitParser

```
package kiel.fileInterface.kit;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PushbackReader;

import kiel.dataStructure.StateChart;
import kiel.fileInterface.FileInterfaceException;
import kiel.util.kit.lexer.Lexer;
import kiel.util.kit.lexer.LexerException;
import kiel.util.kit.node.Start;
import kiel.util.kit.parser.Parser;
import kiel.util.kit.parser.ParserException;

/**
 * .
 * <p>Title: Kiel FileInterface - KIT Language.</p>
 *
 * <p>Description: Calls the automatic created parser </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.5 £ last modified £Date: 2006/02/08 08:32:00 £
 */
public final class KitParser {
  /**
   *  Buffer for Reader.
   */
  private static final int BUFFER = 1024;
  /**
   * Private Constructor.
   */
  private KitParser() {
  }

  /**
   *
   * @param f File
   * @return StateChart
   * @throws FileInterfaceException ex
   */
  public static StateChart parseStateChart(final File f) throws
      FileInterfaceException {
    Lexer lexer = null;
    Parser parser;
    Translation trans = new Translation();
    try {
      lexer = new Lexer(new PushbackReader(new FileReader(f), BUFFER));
    } catch (FileNotFoundException ex) {
      throw new FileInterfaceException(ex.getMessage());
    }
    parser = new Parser(lexer);
    try {
      Start tree = parser.parse();
      tree.apply(trans);
    } catch (IOException ex1) {
      ex1.printStackTrace();
      throw new FileInterfaceException(ex1.getMessage());
    } catch (LexerException ex1) {
      ex1.printStackTrace();
      throw new FileInterfaceException(ex1.getMessage());
    } catch (ParserException ex1) {
      ex1.printStackTrace();
      throw new FileInterfaceException(ex1.getMessage());
    }
    return trans.getStateChart();
  }
}
```

F.11.9. XKitFileFilter

```
package kiel.fileInterface.kit;

import java.io.File;
import java.io.*;

/**
 * <p>Xkit File Filter accepts only xkit-Files.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: </p>
 *
 * @author miwi
 * @version 1.0
 */
public class XKitFileFilter extends javax.swing.filechooser.FileFilter {
    /**
     * <code>KielFileFilter</code> is implemented as a singleton.
     */
    private static XKitFileFilter instance = null;

    /**
     * @param pathname file to be tested
     * @return true if accepted by this plugin
     */
    public final boolean accept(final File pathname) {
        if (pathname.isDirectory()) {
            return true;
        }
        try {
          pathname.createNewFile();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
        return (pathname.canRead()
                && pathname.getPath().toLowerCase().endsWith(".xkit"));
    }

    /**
     * @return Name of file filer -> "Kiel Statecharts"
     */
    public final String getDescription() {
        return "Annotated Kit Language";
    }

    /**
     * @return the static instance for this filter
     */
    public static XKitFileFilter getInstance() {
        if (instance == null) {
            instance = new XKitFileFilter();
        }
        return instance;
    }
}
```

F.11.10. XKit

```
// £Id: XKit.java,v 1.3 2006/02/08 08:32:00 miwi Exp £
package kiel.fileInterface.kit;

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>Kit Language FileInterface.</p>
 *
 * <p>Kit FileInterface xkit main class</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Organisation: CAU Kiel</p>
 *
 * @author <a mailto="miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £
 */
public class XKit extends FileInterface {
  /**
   *
   */
  public XKit() {
  }

  /**
   *
   * @return true if plugin can read charts
   */
  public final boolean canRead() {
    return false;
  }

  /**
   *
   * @return true if plugin can write charts
   */
  public final boolean canWrite() {
    return true;
  }

  /**
   * @return .
   */
  public final String getName() {
    return "Annotated KIT Language Plugin";
  }

  /**
   * returns the view from the read file.
   *
   * @return the read view
   * @throws FileInterfaceException some charts don't contain views
   */
  public final View getReadView() throws FileInterfaceException {
    return null;
  }

  /**
   * FileFilter that accepts right files.
   *
   * @return the file filter
   */
  public final FileFilter getStateChartFileFilter() {
    return XKitFileFilter.getInstance();
  }

  /**
   * reads a statechart from the given file.
   *
   * @param f the file
   * @return the statechart the file conatins
   * @throws FileInterfaceException if the file could not be read
   */
  public StateChart readStateChartDocument(File f) throws
    FileInterfaceException {
//        KitGenerator.generateChart(KitParser.parseStateChart(f), null, null);
    return KitParser.parseStateChart(f);
  }

  /**
   * Write the statechart with this view in the given file.
   *
   * @param s the chart
   * @param v the view
   * @param f file to be written
   */
  public File writeStateChartDocument(StateChart s, View v, File f) {
    return KitGenerator.generateChart(s, v, f, true);
  }
}
```

F.12. kiel.util.declaration

F.12.1. DeclarationParser

```
package kiel.util.declaration;

import java.io.IOException;
import java.io.PushbackReader;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Collection;

import kiel.dataStructure.Variable;
import kiel.dataStructure.boolexp.BooleanVariable;
import kiel.dataStructure.doubleexp.DoubleConstant;
import kiel.dataStructure.doubleexp.DoubleVariable;
import kiel.dataStructure.eventexp.CombineWithAdd;
import kiel.dataStructure.eventexp.CombineWithMult;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.eventexp.Signal;
import kiel.dataStructure.floatexp.FloatConstant;
import kiel.dataStructure.floatexp.FloatVariable;
import kiel.dataStructure.intexp.IntegerConstant;
import kiel.dataStructure.intexp.IntegerVariable;
import kiel.util.declaration.lexer.Lexer;
import kiel.util.declaration.lexer.LexerException;
import kiel.util.declaration.node.AAnotherDecl;
import kiel.util.declaration.node.ABooleanVarEventType;
import kiel.util.declaration.node.ADeclaration;
import kiel.util.declaration.node.ADeclarations;
import kiel.util.declaration.node.ADoubleVarEventType;
import kiel.util.declaration.node.AFloatVarEventType;
import kiel.util.declaration.node.AInitialvalue;
import kiel.util.declaration.node.AIntegerAddVarEventType;
import kiel.util.declaration.node.AIntegerMultVarEventType;
import kiel.util.declaration.node.AIntegerVarEventType;
import kiel.util.declaration.node.AVtype;
import kiel.util.declaration.node.PVarEventType;
import kiel.util.declaration.node.Start;
import kiel.util.declaration.parser.Parser;
import kiel.util.declaration.parser.ParserException;
import kiel.util.declaration.node.AInt16VarEventType;
import kiel.dataStructure.int16exp.Integer16Variable;
import kiel.dataStructure.int16exp.Integer16Expression;
import kiel.dataStructure.int16exp.Integer16Constant;
import kiel.dataStructure.int8exp.Integer8Variable;
import kiel.dataStructure.int8exp.Integer8Constant;
import kiel.dataStructure.uint32exp.UInt32Variable;
import kiel.dataStructure.uint32exp.UInt32Constant;
import kiel.dataStructure.uint16exp.UInt16Constant;
import kiel.dataStructure.uint16exp.UInt16Variable;
import kiel.dataStructure.uint8exp.UInt8Constant;
import kiel.dataStructure.uint8exp.UInt8Variable;
import kiel.util.declaration.node.AInt8VarEventType;
import kiel.util.declaration.node.AUint32VarEventType;
import kiel.util.declaration.node.AUint16VarEventType;
import kiel.util.declaration.node.AUint8VarEventType;

/**
 *
 * <p>Title: Kiel Utils.</p>
 *
 * <p>Description: A parser for declarations of events and variables.
 *                 See declaration.grammer for the grammar of declarations</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 *
 * @version £Revision: 1.5 £ last modified £Date: 2006/05/22 19:21:03 £
 */
public final class DeclarationParser {
  /**
   *  Buffer for Reader.
   */
  private static final int BUFFER = 1024;

  /**
   * Not a valued variable or event.
   */
  private static final int NO_VALUE = 0;
  /**
   * A integer variable or event.
   */
  private static final int INTEGER = 1;
  /**
   * A float variable or event.
   */
  private static final int FLOAT = 2;
  /**
   * A double variable or event.
   */
  private static final int DOUBLE = 3;
  /**
   * A boolean variable or event.
   */
  private static final int BOOLEAN = 4;
  /**
   * A int combine with + variable or event.
   */
  private static final int COMBINE_ADD = 5;
  /**
   * A int combine with * variable or event.
   */
  private static final int COMBINE_MULT = 6;
```

```
  /**
   * A variable type.
   */
  private static final int INT16 = 7;
  /**
   * A variable type.
   */
  private static final int INT8 = 8;
  /**
   * A variable type.
   */
  private static final int UINT32 = 9;
  /**
   * A variable type.
   */
  private static final int UINT16 = 10;
  /**
   * A variable type.
   */
  private static final int UINT8 = 11;

  /**
   * A private constructor because there are only static methods.
   */
  private DeclarationParser() {
    // nothing :-)
  }

  /**
   * @param node ADeclaration to convert
   * @return parsed Event
   */
  private static Event convertADeclarationToEvent(final ADeclaration node) {
    int varEvTyp = NO_VALUE;
    double initValue = 0.0;
    boolean hasInit = false;
    if (node.getVtype() != null) {
      PVarEventType varEvType = ((AVtype) node.getVtype()).getVarEventType();
      if (varEvType.getClass() == ABooleanVarEventType.class) {
        varEvTyp = BOOLEAN;
      } else if (varEvType.getClass() == ADoubleVarEventType.class) {
        varEvTyp = DOUBLE;
      } else if (varEvType.getClass() == AFloatVarEventType.class) {
        varEvTyp = FLOAT;
      } else if (varEvType.getClass() == AIntegerVarEventType.class) {
        varEvTyp = INTEGER;
      } else if (varEvType.getClass() == AIntegerAddVarEventType.class) {
        varEvTyp = COMBINE_ADD;
      } else if (varEvType.getClass() == AIntegerMultVarEventType.class) {
        varEvTyp = COMBINE_MULT;
      }
    }
    if (node.getInitialvalue() != null) {
      initValue = Double.parseDouble(((AInitialvalue) node.getInitialvalue()).
                                     getNumber().getText());
      hasInit = true;
    }
    Event ev = null;
    switch (varEvTyp) {
    default:
    case NO_VALUE:
      ev = new Signal(node.getIdentifier().
                      getText());
      break;
    case INTEGER:
      if (hasInit) {
        ev = new IntegerSignal(node.getIdentifier().getText(),
                               new IntegerConstant((int) initValue));
      } else {
        ev = new IntegerSignal(node.getIdentifier().getText());
      }
      break;
    case COMBINE_ADD:
      if (hasInit) {
        ev = new CombineWithAdd(node.getIdentifier().getText(),
                                new IntegerConstant((int) initValue));
      } else {
        ev = new CombineWithAdd(node.getIdentifier().getText());
      }
      break;
    case COMBINE_MULT:
      if (hasInit) {
        ev = new CombineWithMult(node.getIdentifier().getText(),
                                 new IntegerConstant((int) initValue));
      } else {
        ev = new CombineWithMult(node.getIdentifier().getText());
      }
      break;
    }
    return ev;
  }

  /**
   * @param node ADeclaration to convert
   * @return parsed Variable
   */
  private static Variable convertADeclarationToVar(final ADeclaration node) {
    int varEvTyp = NO_VALUE;
    double initValue = 0.0;
    boolean hasInit = false;
    if (node.getVtype() != null) {
      PVarEventType varEvType = ((AVtype) node.getVtype()).getVarEventType();
      if (varEvType.getClass() == ABooleanVarEventType.class) {
        varEvTyp = BOOLEAN;
      } else if (varEvType.getClass() == ADoubleVarEventType.class) {
        varEvTyp = DOUBLE;
      } else if (varEvType.getClass() == AFloatVarEventType.class) {
        varEvTyp = FLOAT;
      } else if (varEvType.getClass() == AIntegerVarEventType.class) {
        varEvTyp = INTEGER;
      } else if (varEvType.getClass() == AInt16VarEventType.class) {
        varEvTyp = INT16;
      } else if (varEvType.getClass() == AInt8VarEventType.class) {
        varEvTyp = INT8;
      } else if (varEvType.getClass() == AUint32VarEventType.class) {
        varEvTyp = UINT32;
      } else if (varEvType.getClass() == AUint16VarEventType.class) {
        varEvTyp = UINT16;
      } else if (varEvType.getClass() == AUint8VarEventType.class) {
        varEvTyp = UINT8;
```

```
      } else if (varEvType.getClass() == AIntegerAddVarEventType.class) {
        varEvTyp = COMBINE_ADD;
      } else if (varEvType.getClass() == AIntegerMultVarEventType.class) {
        varEvTyp = COMBINE_MULT;
      }
    }
    if (node.getInitialvalue() != null) {
      initValue = Double.parseDouble(((AInitialvalue) node.getInitialvalue()).
                                     getNumber().getText());
      hasInit = true;
    }
    Variable v = null;
    switch (varEvTyp) {
    default:
    case NO_VALUE:
    case INTEGER:
      if (hasInit) {
        v = new IntegerVariable(node.getIdentifier().getText(),
                                new IntegerConstant((int) initValue));
      } else {
        v = new IntegerVariable(node.getIdentifier().getText());
      }
      break;
    case INT16:
      if (hasInit) {
        v = new Integer16Variable(node.getIdentifier().getText(),
                                  new Integer16Constant((short) initValue));
      } else {
        v = new Integer16Variable(node.getIdentifier().getText());
      }
      break;
    case INT8:
      if (hasInit) {
        v = new Integer8Variable(node.getIdentifier().getText(),
                                 new Integer8Constant((byte) initValue));
      } else {
        v = new Integer8Variable(node.getIdentifier().getText());
      }
      break;
    case UINT32:
      if (hasInit) {
        v = new UInt32Variable(node.getIdentifier().getText(),
                               new UInt32Constant((int) initValue));
      } else {
        v = new UInt32Variable(node.getIdentifier().getText());
      }
      break;
    case UINT16:
      if (hasInit) {
        v = new UInt16Variable(node.getIdentifier().getText(),
                               new UInt16Constant((short) initValue));
      } else {
        v = new UInt16Variable(node.getIdentifier().getText());
      }
      break;
    case UINT8:
      if (hasInit) {
        v = new UInt8Variable(node.getIdentifier().getText(),
                              new UInt8Constant((byte) initValue));
      } else {
        v = new UInt8Variable(node.getIdentifier().getText());
      }
      break;
    case BOOLEAN:
      v = new BooleanVariable(node.getIdentifier().getText());
      break;
    case DOUBLE:
      if (hasInit) {
        v = new DoubleVariable(node.getIdentifier().getText(),
                               new DoubleConstant(initValue));
      } else {
        v = new DoubleVariable(node.getIdentifier().getText());
      }
      break;
    case FLOAT:
      if (hasInit) {
        v = new FloatVariable(node.getIdentifier().getText(),
                              new FloatConstant((float) initValue));
      } else {
        v = new FloatVariable(node.getIdentifier().getText());
      }
      break;
    }
    return v;
  }

  /**
   * @param s String with some variables declared
   * @return Collection of Variables
   * @throws DeclarationException if there is an lexer/parser/io exception
   */
  public static Collection parseVariables(final String s) throws
    DeclarationException {
    ArrayList vars = new ArrayList();
    if (!s.trim().equals("")) {
      Lexer lexer = null;
      Parser parser;
      lexer = new Lexer(new PushbackReader(new StringReader(s), BUFFER));
      parser = new Parser(lexer);
      try {
        Start tree = parser.parse();
        if (((ADeclarations) tree.getPDeclarations()).
            getDeclaration() != null) {
          vars.add(convertADeclarationToVar((ADeclaration)
                                            ((ADeclarations) tree.
                                             getPDeclarations()).
                                            getDeclaration()));
        }
        if (((ADeclarations) tree.getPDeclarations()).getAnotherDecl() != null
            && ((ADeclarations) tree.getPDeclarations()).
            getAnotherDecl().size() > 0) {
          ADeclaration decl = null;
          for (int i = 0; i < ((ADeclarations) tree.getPDeclarations()).
                       getAnotherDecl().size(); i++) {
            decl = (ADeclaration) ((AAnotherDecl) ((ADeclarations)
              tree.getPDeclarations()).getAnotherDecl().
                                   get(i)).getDeclaration();
            vars.add(convertADeclarationToVar(decl));
          }
        }
```

```
    } catch (IOException ex1) {
      ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    } catch (LexerException ex1) {
      ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    } catch (ParserException ex1) {
      ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    }
  }
  return vars;
}

/**
 * @param s String with some events declared
 * @return Collection of events
 * @throws DeclarationException if there is an lexer/parser/io exception
 */
public static Collection parseEvents(final String s) throws
    DeclarationException {
  ArrayList events = new ArrayList();
  if (!s.trim().equals("")) {
    Lexer lexer = null;
    Parser parser;
    lexer = new Lexer(new PushbackReader(new StringReader(s), BUFFER));
    parser = new Parser(lexer);
    try {
      Start tree = parser.parse();
      if (((ADeclarations) tree.getPDeclarations()).
          getDeclaration() != null) {
        events.add(convertADeclarationToEvent((ADeclaration)
                                              ((ADeclarations) tree.
                                                getPDeclarations()).
                                                getDeclaration()));
        ADeclaration decl = null;
        for (int i = 0; i < ((ADeclarations) tree.getPDeclarations()).
                            getAnotherDecl().size(); i++) {
          decl = (ADeclaration) ((AAnotherDecl) ((ADeclarations)
            tree.getPDeclarations()).getAnotherDecl().
                                       get(i)).getDeclaration();
          events.add(convertADeclarationToEvent(decl));
        }
      }
    } catch (IOException ex1) {
      // ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    } catch (LexerException ex1) {
      // ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    } catch (ParserException ex1) {
      // ex1.printStackTrace();
      throw new DeclarationException(ex1.getMessage());
    }
  }
  return events;
}
}
```

F.12.2. DeclarationException

```
package kiel.util.declaration;

/**
 *
 * <p>Title: Kiel util.</p>
 *
 * <p>Description: Exception for a wrong declaration.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/02/08 08:32:12 £
 */
public class DeclarationException extends Exception {
  /**
   * @param s String with an Exception cause.
   */
  public DeclarationException(final String s) {
    super(s);
  }

}
```

F.13. kiel.util.languages

F.13.1. UpdateThread

```
package kiel.util.languages;

import kiel.dataStructure.StateChart;
import javax.swing.text.Document;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.3 £ last modified £Date: 2006/03/06 09:59:03 £
 */

public interface UpdateThread {
  /**
   *
   * @param c StateChart
   * @param aDoc Document
   */
  void setStatechart(final StateChart c, final Document aDoc);
  /**
   *  Call this if a key was pressed.
   */
  void notifyKeyboardAction();
  /**
   *  Forces an update, use this to make update without waiting.
   */
  void forceUpdate();
}
```

F.13.2. ILineNumber

```
package kiel.util.languages;

import javax.swing.ImageIcon;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.2 £ last modified £Date: 2006/02/08 08:32:12 £
 */
public interface ILineNumber {
  /**
   *
   * @param line int
   * @param icon ImageIcon
   * @param message String
   */
  void addIconOnLine(int line, ImageIcon icon, String message);
  /**
   *
   * @param line int
   */
  void removeIconOnLine(int line);
  /**
   * remove icons.
   */
  void clearIcons();
}
```

F.13.3. StatechartLanguage

```
package kiel.util.languages;

import java.io.File;

import javax.swing.JTextPane;

import kiel.dataStructure.StateChart;

/**
 * <p>Title: Kiel Browser.</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version £Revision: 1.4 £ last modified £Date: 2006/04/04 23:28:57 £
 */
public interface StatechartLanguage {
  /**
   *
   * @param ut UpdateThread
   */
  void setUpdateThread(UpdateThread ut);
  /**
   *
   * @param ln ILineNumber
   */
  void setLineNumber(ILineNumber ln);
  /**
    *
    * @param ln ILineNumber
    */
   ILineNumber getLineNumber();
   /**
    *
    * @param p JTextPane
    */
  void setTextPane(JTextPane p);
  /**
   *
   * @return boolean
   */
  boolean isReadOnly();
  /**
   *
   * @param chart StateChart
   * @param chartFile File
   */
  void setStatechart(StateChart chart, File chartFile);
  /**
   *
   * @return String
   */
  String getTooltip();
  /**
   *
   * @return String
   */
  String getName();

  /**
   * Goes to last edited line.
   */
  void restoreCursor();
}
```