Christian-Albrechts-Universität zu Kiel

Studienarbeit

Ein Browser für die Visualisierung dynamischer Sichten von Statecharts

cand. inform. Mirko Wischer

28. Juni 2006

Institut für Informatik und Praktische Mathematik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:
Steffen H. Prochnow

Eidesstattliche Erklärung

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

Kiel,

_____

Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Browser zur Visualsierung von Statecharts und Steuern eines Simulator-Modul für das Projekt *KIEL* entwickelt. Der Browser unterstützt die Darstellung *Dynamischer Statecharts*, dabei werden inaktive Zustände wärend einer Simulation versteckt. Dies stellt eine mögliche Technik zur Darstellung großer Statecharts während einer Simulation dar. Dabei kommt ein im Rahmen einer anderen Arbeit entwickeltes Layout-Modul zur Generierung eines automatischen Layouts für Statecharts zum Einsatz. Evaluierungsphase verschiedener Toolkits zur Visualisierung stellte sich heraus, dass ein SVG-Toolkit die gewünschten Anforderungen am Besten unterstützt.

Schlüsselwörter Statechart, Dynamische Statecharts, *KIEL*

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Verzeichnis der Auflistungen

Verzeichnis der Abkürzungen

UML	*Unified Modeling Language*
KIEL	*Kiel Integrated Environment for Layout*
ZUI	*Zoomable User Interface*
HCI	*Human Computer Interaction*
SVG	*Scaleable Vector Graphics*
SMIL	*Synchronized Multimedia Integration Language*
XML	*Extensible Markup Language*
CSS	*Cascading Style Sheets*
MVC	*Model-View-Controller*

1. Einleitung

Die Anzahl der digital gesteuerten Geräte nimmt immer weiter zu, dabei sind gerade die Eingebetteten Systeme, dass heißt Computer, die nicht als solche erkannt werden, auf dem Vormarsch. Solche Systeme sind häufig Reaktive Systeme, stehen also in ständiger Interaktion mit der Umwelt. Beispiele für Reaktive Systeme sind *Airbag Controller* oder digitale Thermostate. Sie lassen sich mit Hilfe von Zustandsdiagrammen, auch Zustandsübergangsdiagramme genannt, beschreiben (engl: *Statecharts*). Für die Erstellung der Statecharts werden sogenannte Modellierungswerkzeuge eingesetzt. Die einzelnen Modellierungswerkzeuge verwenden häufig unterschiedliche Statechart-Dialekte. Mit Aufnahme der Statecharts in die ***U****nified* ***M****odeling* ***L****anguage* (kurz: UML) existiert ein Statechart-Dialekt, der nicht an einen Hersteller eines Modellierungswerkzeugs gebunden ist.

Je komplexer die beschriebenen Systeme sind, desto größer ist die benötige Fläche des Zustandsdiagramms. Bei der Erweiterung bestehender Statecharts muss der Entwickler zunächst Platz im Statecharts schaffen. Sind die Erweiterungen ergänzt worden, findet meist noch zusätzlich eine Umordnung der einzelnen Zustände statt, um das *Layout* der neuen Situation anzupassen. An dieser Stelle setzen Layoutverfahren an, die es ermöglichen, ein automatisiertes *Layout* zu erzeugen. Einerseits ist es möglich Layoutverfahren mit Hilfe einer Anbindung an ein Modellierungswerkzeug [12] umzusetzen, dabei ist man durch die Visualisierungsmöglichkeiten des Modellierungswerkzeugs beschränkt. Andererseits ist es möglich eigene Visualisierungsmechanismen umzusetzen, um beispielsweise dynamische Elemente in die Visualisierung einfließen zu lassen.

1.1. Aufgabenstellung im Rahmen des Projekts KIEL

Am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme wird ein neues Modellierungswerkzeug entwickelt, es trägt den Namen *KIEL*. Der Name steht für ***K****iel* ***I****ntegrated* ***E****nvironment for* ***L****ayout*. Aufgabe dieser Arbeit ist es, ein Browser für dieses Projekt zu entwickeln, unter einem Browser wird hier eine Visualisierungsumgebung für Statecharts verstanden, die sowohl Statecharts anzeigen als auch deren Simulation steuern kann. Der Browser interagiert dabei mit verschiedenen anderen Modulen aus dem Projekt *KIEL*, ein Überblick der Module befindet sich in Abbildung 1.1. Der Browser soll verschiedene Arten von Visualisierung der Statecharts vornehmen: statische Visualisierung (siehe Abschnitt 2.1.2) und dynamische Visualisierung (siehe Abschnitt 2.1.2). Im Vordergrund dieser Arbeit steht neben dem Prozess der Visualisierung von Statecharts die Steuerung eines Simulator-Moduls.

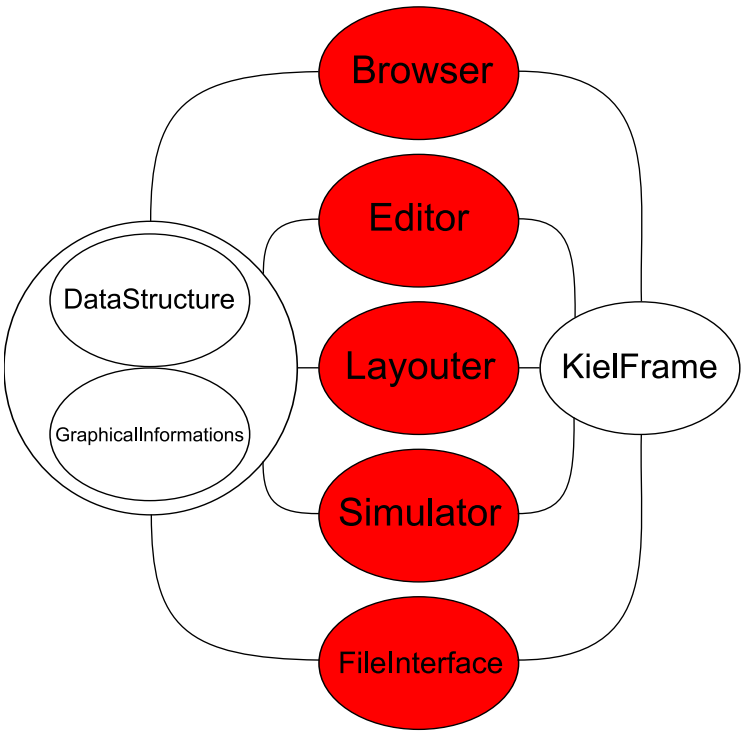


Abbildung 1.1.: *KIEL* Module im Überblick

Ein Modul für die Simulation von *Esterel Studio* Statecharts wurde im Rahmen einer Arbeit von André Ohloff [19] für das Projekt *KIEL* entwickelt. Ein weiteres Modul das im Rahmen des Projektes *KIEL* entwickelt wurde ist das Editor-Modul, mit dessen Hilfe man Statecharts erstellen kann. Es wurde im Rahmen einer Arbeit von Florian Lüpke [16] entwickelt. Wie das Akronym schon vermuten lässt ist ein Hauptmodul von *KIEL* das Layouter-Modul. Hiermit ist es möglich ein automatisches Layout auf Statecharts anzuwenden. In der Arbeit von Tobias Kloss [13] wurde ein Layout-Modul entwickelt, dass den Browser mit den, für die dynamische Visualisierung nötigen graphischen Informationen versorgt. Ein weiteres Modul von *KIEL* ist das FileInterface [27]. Es stellt ein *Interface* zur Verfügung um Statecharts aus Dateien einzulesen, in die *KIEL*-Datenstruktur zu konvertieren und um eine vorhandene *KIEL*-Datenstruktur in eine Datei abzulegen. Es können verschiedene Dateiformate in das FileInterface integriert werden, es existieren Möglichkeiten *Esterel Studio* Dateien und serialisierte *KIEL*-Datenstrukturen einzulesen. Ein weiteres Dateiformat ist im Rahmen der Arbeit von Lars Kühl [15] entstanden. Es ermöglicht, die synchrone Sprache *Esterel* Dateien in Statecharts umzuwandeln.

2. Grundlagen

In den folgenden Abschnitten werden die Grundlagen dieser Arbeit erläutert. Dazu gehören neben den theoretischen Grundlagen von Statecharts auch die Grundlagen der Implementierung. Dabei werden zwei Themengebiete besprochen: Existierende Werkzeuge zur Darstellung von Statecharts und die verschiedenen *Toolkits*, die zur Implementierung genutzt werden könnten und deren Evaluation.

2.1. Theoretische Grundlagen

In den theoretischen Grundlagen wird die Bedeutung und der Nutzen von Statecharts in der Software-Techologie erläutert. Dazu wird zum einen die Bedeutung und Funktionsweise von Statecharts und zum andern die Möglichkeiten einer Visualisierung von Statecharts erläutert. Bei der Visualisierung von Statecharts wird zuerst auf die statische Visualisierung von Statecharts und anschließend auf die dynamische Visualisierung eingegangen.

2.1.1. Modellierung reaktiver Systeme

Reaktive Systeme, also Computer-Systeme, die sich in ständiger Interaktion mit ihrer Umwelt befinden, lassen sich gut mit Hilfe von Statecharts beschreiben. Sie beschreiben das Verhalten des Systems in Abhängigkeit von Eingabewerten. Für die Modellierung von Statecharts-Dialekten gibt es zahlreiche Werkzeuge, die neben einer graphischen Darstellung meist eine Simulation oder Codesynthese aus den erzeugten Statecharts ermöglichen.

Von deterministischen endlichen Automaten zu Statecharts

Endliche Automaten sind bekannte Modelle die in der Automatentheorie [7] schon seit geraumer Zeit analysiert worden sind. Ein deterministischer endlicher Automat M (englisch: *deterministic finite automaton*, kurz: DFA) wird dort formal als ein 5-Tupel definiert:

$$M = (Z, \Sigma, \delta, z_0, E).$$

Mit Z bezeichnet man die Menge der Zustände und mit Σ das Eingabealphabet. Es gilt hier zusätzlich $Z \cap \Sigma = \emptyset$ außerdem müssen Z und Σ endliche Mengen sein. Dabei ist z_0 ist der Startzustand, $E \subseteq Z$ ist die Menge der Endzustände und $\delta : Z \times \Sigma \to Z$ ist die Überführungsfunktion.

Sei $M = (Z, \Sigma, \delta, z_0, E)$, mit

$$\begin{aligned} Z &= \{z_0, z_1, z_2, z_3\} \\ \Sigma &= \{a, b\} \\ E &= \{z_3\} \\ \delta(z_0, a) &= z_1 \\ \delta(z_0, b) &= z_3 \\ \delta(z_1, a) &= z_2 \\ \delta(z_1, b) &= z_0 \\ \delta(z_2, a) &= z_3 \\ \delta(z_2, b) &= z_1 \\ \delta(z_3, a) &= z_0 \\ \delta(z_3, b) &= z_2 \end{aligned}$$

Tabelle 2.1.: DFA in textueller Form

Ein Beispiel für die formale Definition eines Automaten befindet sich in Tabelle 2.1. Dieser Automat erkennt eine Sprache $L \subseteq \Sigma^*$. Welche Sprache dies ist, ist auf den ersten Blick in dieser Darstellung schwer zu erkennen. Durch einen Zustandsgraphen veranschaulicht, läßt sich die Sprache die ein Endlicher Automat erkennt einfacher erkennen. Ein Zustandsgraph ist ein gerichteter, beschrifteter Graph, dessen Knoten die Zustände darstellen. Der Knoten, der dem Startzustand entspricht, wird besonders markiert, ebenso alle Endzustände. Betrachtet man nun die graphische Darstellung in Abbildung 2.1 ist die Funktionsweise des Automaten einfacher zu durchschauen.

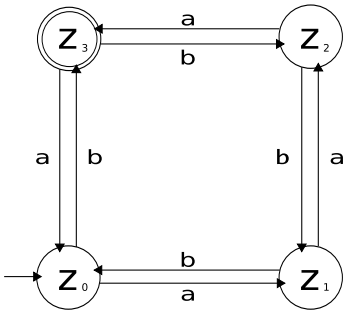


Abbildung 2.1.: Zustandsdiagramm eines DEA

Man erkennt, dass der Automat folgende Sprache akzeptiert:

$$L = \{x \in \Sigma^* | ((\text{Anzahl } a\text{'s in } x) - (\text{Anzahl } b\text{'s in } x)) \equiv 3(\mod 4)\}.$$

Statecharts sind eng mit den Endlichen Automaten verwandt. Der Begriff des Statecharts wurde von David Harel [11] geprägt. Die Statecharts erweitern das Prin-

zip der Automaten um einige Konzepte wie Hierarchie oder Parallelität. Durch die neuen Konzepte ist es möglich, ein System zu konstruieren, dass auf Eingaben reagiert. Solche Reaktiven Systeme finden vor allem bei Eingebetteten Systemen eine Anwendung. Vorteile solcher Systeme sind die besseren Analysemöglichkeiten durch formale Methoden, die Simulation des Systems und die Möglichkeiten einer Codesynthese.

2.1.2. Visualisierung von Statecharts

Modellierungswerkzeuge benutzen verschiedene Dialekte von Statecharts, die Dialekte unterscheidet sich in der Darstellung und der Semantik der Statecharts. Der zugrundeliegende Graph ist jedoch immer zu erkennen. In Bezug auf die Darstellung beziehen sich die Unterschiede auf Formen von Zuständen oder Transitionen. Während einer Simulation wird mit Hilfe von Farben gekennzeichnet, welcher Zustand aktiv oder welche Transition getestet oder ausgeführt wird.

Statische Visualisierung

Die Statische Visualisierung ist der klassische Ansatz der in allen Modellierungswerkzeuge eingesetzt wird. Dabei wird ein Statechart jeweils komplett gezeichnet und bei einer Simulation wird durch Einfärben der Ablauf der Simulation verdeutlicht. Es gibt nur eine Sicht auf jedes graphische Objekt.

Dynamische Visualisierung

Bei der dynamischen Visualisierung wird die statische Bindung zwischen den Objekten (Zustände, Transitionen usw.) und deren graphischen Repräsentationen aufgehoben. Dies hat zur Folge, dass ein Zustand zu zwei unterschiedlichen Zeitpunkten auch unterschiedlich aussehen kann. Damit wird es möglich, Teile des Statecharts, die im Moment nicht interessant sind, auszublenden [14]. Während einer Simulation werden auf diese Weise nur die Hierarchieebenen gezeigt, die gerade aktiv sind. Unter einer aktiven Hierarchieebene versteht man eine Ebene die einen aktiven Zustand enthält. Dies schafft Platz auf der Zeichenebene und erhöht die Übersichtlichkeit. Die Änderungen in Position und Größe finden jeweils gleitend statt, um die Übersichtlichkeit zu erhöhen und eine schnelle Zuordnung der einzelnen Objekte zu ermöglichen. In Abbildung 2.2 wird eine gleitende Veränderung an Hand eines Beispiels verdeutlicht. Diese Technik wird *Focus-and-context* genannt. Statecharts, die solche dynamische Sichten enthalten heißen *Dynamic-Statecharts*. Dieser Begriff wurde durch das Projekt *KIEL* geprägt [23].

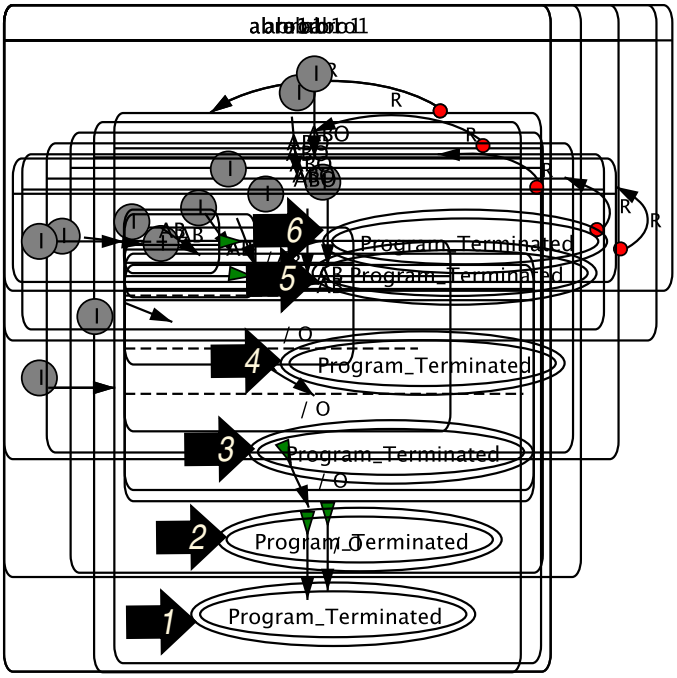


Abbildung 2.2.: Wechsel der Sicht in einem Statechart

2.2. Evaluierung existierender Statechart Visualisierungen

Die Modellierungswerkzeuge unterscheiden sich in Bezug auf die Visualisierung kaum, deshalb werden in dieser Arbeit, beispielhaft für alle vorhandenen Modellierungswerkzeuge, *Esterel Studio* [9] und *Matlab-Stateflow* [17] vorgestellt. Weitere bekannte Modellierungswerkzeuge sind:

- *ArgoUML* [3]
- *Statemate*
- *ARTiSAN RealTime Studio* [4]
- *ASCET-SD*
- *Autofocus*
- *Rhapsody*
- *Rose RealTime*

2.2.1. Esterel Studio

Statecharts in *Esterel Studio* [2] heißen ***S**afe **S**tate **M**achines* (kurz: SSM), sie sind eine komerzielle Variante der von Charles Andreé [1] entwickelten *SyncCharts*. In

Abbildung 2.3 wird ein Überblick über verwendete Statechart-Komponenten gegeben.

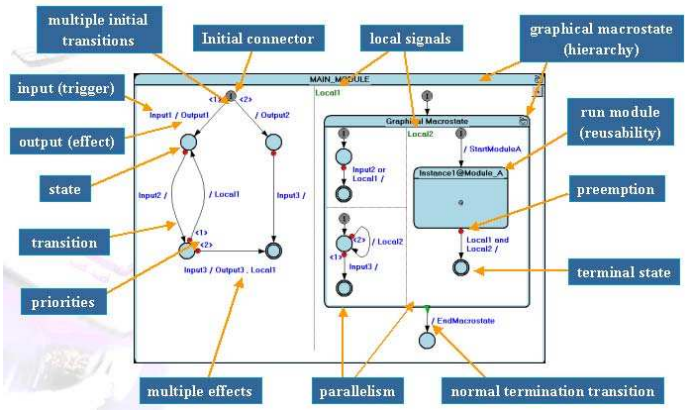


Abbildung 2.3.: Aufbau einer SSM[1]

2.2.2. Matlab-Stateflow

Statecharts in *Matlab-Stateflow* werden ***F**inite **S**tate **M**achines* (kurz: FSM) genannt, sie wurden mit *Stateflow* [17] in *Matlab 5.1* eingeführt. *Stateflow* ist dabei eine Ergänzung der *Matlab*-Erweiterung *Simulink* [18]. In Abbildung 2.3 wird ein Überblick über verwendete Statechart-Komponenten gegeben.

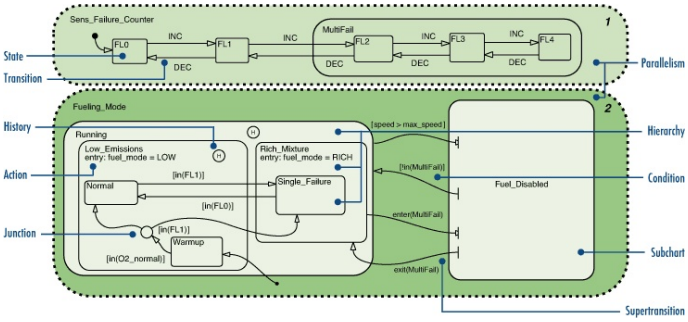


Abbildung 2.4.: Aufbau einer FSM[2]

2.2.3. Zusammenfassung

Betrachtet man die verschiedenen Darstellungsweisen der einzelnen Statechart- Dialekte bemerkt man Unterschiede und Gemeinsamkeiten. Bei der Entwicklung des

[1]Entnommen aus dem *Esterel Studio Manual*
[2]Entnommen von der *Matlab-Simlink* Homepage

Projekts *KIEL* wurde darauf geachtet verschiedene Statechart-Dialekte zu unterstützen, deshalb wurde bei der Entwicklung des Browsers darauf geachtet, eine möglichst leicht anzupassende Darstellungsweise zu erhalten.

2.3. Toolkits zur Visualisierung

In diesem Abschnitt werden mögliche Toolkits zur Visualisierung von Statecharts besprochen. Dafür werden die Anforderungen an eine solche Visualisierung aufgezeigt und die Möglichkeiten existierende Toolkits zur Visualisierung zu nutzen, erläutert. Abschließend werden die in Frage kommenden Toolkits nochmals in einer Übersichtsdarstellung aufgelistet und bewertet.

Anforderungen an die Toolkits

Es existieren zwei verschiedene Arten von Anforderungen an die Toolkits: Als erstes sind hier die technischen Anforderungen, die sich auf die unterstützten und benötigten *Features* beziehen zu nennen, und als nächstes projektspezifische Anforderungen, die sich auf die Art der Verfügbarkeit beziehen. Folgende technische Anforderungen sind gegeben:

Zeichenoperationen: Es sollen verschiedenste Zeichenoperationen möglich sein, da die Darstellungsweise der Komponenten eines Statecharts konfigurierbar sein soll. Das Aussehen soll im besten Fall nicht durch die Implementierung gegeben sein, sondern durch Konfigurationsdateien festgelegt werden können.

Hierarchie: Die in Statecharts vorhandene Hierarchie sollte übernommen werden können, um unnötige Berechnungen von Positionsangaben zu vermeiden. Das Bewegen eines Vaterzustandes hat zur Folge, dass sich die inneren Zustände mit bewegen, ohne dass eine Berechnung nötig ist.

Pfeile: Transitionen werden als Pfeile dargestellt. Die Pfeile können eine unterschiedliche Start- bzw. Endmarkierung besitzen. Solch eine Eigenschaft sollte leicht umsetzbar sein.

Animationen: Die Dynamik in den Statecharts soll über Animationen dargestellt werden, dabei ändern die Zustände sowohl Größe als auch die Position. Die verschiedenen Beschriftungen ändern ebenfalls ihre Position. Transitionen können ihren kompletten Verlauf ändern.

Zoom: Da die Statecharts über die vorhandene Zeichenfläche herausragen können, sollte es einen einfachen Weg geben, eine erzeugte Zeichnung in ihrer gesamten Größe zu ändern.

Auszeichnungen: Während einer Simulation ändern sich häufig die Farben der einzelnen Zustände oder Transitionen, daher sollten diese Eigenschaften leicht veränderbar sein.

Dazu kommen einige projektspezifische Anforderungen:

Verfügbarkeit: Sind die Toolkits kommerziell oder als *Open Source* entwickelt worden? Die Quellen sollten verfügbar sein, um dort eventuell Änderungen oder Anpassungen vornehmen zu können.

Aktualität: Wird das Toolkit noch entwickelt oder ist die letzte Änderung schon Jahre her? Wenn es sich um eine stabile Version handelt, ist die Aktualität nicht ausschlaggebend, ist es jedoch ein relativ neues Toolkit und die Entwicklung stockt schon zu Beginn des Projekts, müssen Fehler im Toolkit selbst behoben werden und das kostet unnötig Zeit.

Anwenderbasis: Nutzen andere Personen ebenfalls dieses Toolkit oder gibt es Referenzen die ein positives Licht auf das Toolkit werfen? Ist das der Fall, gibt es eine *Community*, auf die man bei Fragen zugehen kann.

2.3.1. Allgemeine Zeichentoolkits

Da der entwickelte Browser ein Teil des Projekts *KIEL* ist und dies in *Java* entwickelt wird, bietet sich die Möglichkeit, die mitgelieferten Zeichenoperationen zu nutzen. Hierfür stellt *Java* jedem Entwickler eine Reihe von Klassen zur Verfügung.

Java-2D

Mit Hilfe von *Java*-2D lassen sich alle Anforderungen umsetzten. Allerdings ist der Aufwand für alle Anforderungen hoch, da es keine speziellen Mechanismen gibt, die einem in den einzelnen Punkten etwas Arbeit abnehmen. Dafür ist man auch keinen Einschränkungen unterworfen. In Bezug auf die projektspezifischen Anforderungen steht *Java*-2D, was die Aktualität und die Anwenderbasis angeht, am Besten dar. Das liegt zum größten Teil daran, dass *Java*-2D Teil der *Java* Klassenbibliothek ist.

2.3.2. Zoomable-User-Interface-Toolkits

Ein weiterer Ansatz ist der Einsatz sogenannter ***Z**oomable-**U**ser-**I**nterface*-Toolkits (kurz: ZUI). Diese Toolkits sind nicht nur zur Visualisierung sondern auch für Interaktion mit den gezeichneten Objekten geeignet. Solche Toolkits kommen aus dem Bereich der *Human-Computer-Interaction* (kurz: HCI). Der Ansatz solcher Toolkits ist das semantische Zoomen, das bedeutet, dass die Darstellung der gezeichneten Objekte vom Zoom bzw. der Entfernung des Objektes zum Betrachter abhängt.

Pad++/Jazz/Piccolo

Entwickler der Toolkits [20] ist eine Arbeitsgruppe des *Human-Computer-Interaction-Lab* der Universität von Maryland. *Pad++* [6], das älteste Toolkit, basiert auf *C++*, danach wurde *Jazz*, ein *Java*-Toolkit entwickelt. Das aktuelle Toolkit ist *Piccolo*. Der

Funktionsumfang der Toolkits ist ähnlich sie unterschieden sich jedeoch in der Effizienz der Umsetzung und der Einfachheit der Benutzung für die Entwickler. *Piccolo* ist für zwei verschieden Sprachen verfügbar: *Java* und *C#*. Da alle Funktionen von *Jazz* auch mit *Piccolo* möglich sind, *Piccolo* jedoch einfacher zu nutzen und weitaus aktueller ist, bezieht sich die Evaluierung ausschließlich auf *Piccolo*. Diesem sind nur drei verschiedene Zeichenoperationen bekannt: Pfade, Text und Bilder. Es ist auch möglich zusätzliche Operationen zu implementieren. Hierbei kommt als zugrundeliegende *Rendering*-Einheit *Java*-2D zum Einsatz. Eine Dateischnittstelle existiert nicht. Eine Hierarchie ist einfach umzusetzen , da die Objekte, die *Piccolo* zeichnet, jeweils noch Kinder haben können. Um auf einfache Weise Pfeile darzustellen, kann man die Möglichkeit von *Piccolo* nutzen, die Zeichenoperationen zu erweitern. Jedes Objekt, was gezeichnet wird, kann animiert werden. Es existieren vorgefertigte Methoden, um die Position, die Rotation und den Skalierungsfaktor eines Objektes zu animieren. Die Skalierung ist jedoch ein optischer Zoom, um eine Größenänderung ohne optischen Zoom hervorzurufen, muss ein höherer Aufwand getrieben werden. Ein Zoom der kompletten Zeichenfläche ist ohne Probleme möglich. Auszeichnungen müssten über die erweiterten Zeichenoperationen erzeugt werden.

In Bezug auf die projektspezifischen Anforderungen schneidet *Piccolo* gut ab. Die Quellen sind per *CVS* oder als Archiv verfügbar. Das letzte *Release* ist vom 01. August 2004 und die neuste Beta Version vom 22. Dezember 2004. Die Homepage enthält *Tutorials* und Beispielprogramme, eine Mailingliste ist ebenfalls vorhanden. Es sind 20 Projekte auf der Homepage [20] verzeichnet, die *Piccolo* nutzen.

Zomit

Zomit ist ebenfalls ein ZUI-Toolkit. Es wurde von Stuart Pook [28] als Teil seiner Dissertation entwickelt. Die Quellen sind nicht direkt über das Internet verfügbar, aber die vorhandenen Demonstrationen legen ein ähnliches Verhalten wie *Piccolo* nahe. Neben den schon bei *Piccolo* vorgestellten *Features*, gibt es noch verschiedene Möglichkeiten zur Interaktion. Das Projekt, in dessen Rahmen dieses Toolkit entwickelt wurde, ist abgeschlossen und es scheint dafür keine weitere Verwendung zu geben. Zur Funktionsweise existieren nur eine Veröffentlichung [21] und die bereits erwähnten Beispielprogramme.

2.3.3. Scalable-Vector-Graphics Toolkits

SVG [26] ist seit 2001 eine W3C Empfehlung um zweidimensionale Vektorgraphiken zu beschreiben und mittels der **S**ynchronized-**M**ultimedia-**I**ntegration-**L**anguage (kurz: SMIL) zu animieren. Es basiert auf der e**X**tensible-**M**arkup-**L**anguage (kurz: XML). Es gibt verschiedene Implementierungen: Neben reinen Betrachtern für SVG-Dateien gibt es auch Toolkits, die zur Implementierung von eigenen Anwendungen, die SVG als *Rendering-Engine* nutzen, geeignet sind. Die offizielle W3C Homepage [26] enthält eine Liste von Implementierungen. Daneben gib es noch zahlreiche weitere, die nicht dort aufgeführt wurden. Die Implementierungen unterscheiden sich

zum Teil erheblich, sowohl im Umfang der Implementierung, als auch den zugrundeliegenden Bibliotheken bzw. Programmiersprachen.

Die SVG-Toolkit setzen folgende Anforderungen um: Zeichenoperationen, wie Rechteck, Ellipse, Pfad, Text, werden von SVG zur Verfügung gestellt. Man kann in allen Implementierungen SVG-Dateien laden und daraus neue Zeichnungen erzeugen. Hierarchie wird von SVG unterstützt, indem man Grafikprimitive gruppieren kann. Man kann dadurch komplexe Objekte verschieben, indem man die Position der Gruppe ändert. Pfeile lassen sich sehr einfach darstellen, da Pfade Start- und Endmarkierungen haben können. Diese Markierungen sind beliebige SVG Grafikprimitive, dies entspricht genau den Anforderungen. Animationen sind in SVG über SMIL vorgesehen es lässt sich damit jedes beliebige Attribut eines SVG-Elements animieren. Zum Ändern von Positionen gibt es SVG-Elemente. Auch das Verändern von Pfaden wird von SVG unterstützt. Damit setzen SVG-Toolkits alle Anforderungen bzgl. Animationen um. Zoom wird von den Toolkits umgesetzt. Da es sich um Vektorgrafiken handelt, kann man die Positionsangaben transformieren um die gesammte Zeichnung zu vergrößern oder zu verkleinern. Auszeichnungen lassen sich einfach setzen, da jedes SVG Objekt über **C**ascading-**S**tyle-**S**heets (kurz: CSS) veränderbar ist, mit ist diese Anforderung ebenfalls umgesetzt. Zusätzlich kann man alle Objekte mit einer eindeutigen *ID* versehen und damit direkt auf sie zugreifen, ohne die SVG-Datenstruktur komplett zu durchsuchen.

Apache Batik SVG

Ein häufig verwendetes Toolkit, ist ein *Java*-Toolkit, das aus dem *Apache* XML Projekt [5] stammt. Es bietet die beste Unterstützung bei den statischen SVG Elementen, Animationen können jedoch nur über *ECMAScript* (besser bekannt als *JavaScript*) realisiert werden. In Bezug auf die projektspezifischen Anforderungen ist *Batik* vergleichbar mit *Piccolo*. Die Quellen sind ebenfalls per *CVS* oder als Archiv verfügbar. Das letzte *Release* hat bereits die Version 1.6 und es gibt täglich neue Ausgaben der Beta Version. Die Homepage [5] enthält *Tutorials* und Beispielprogramme. Es sind 18 Projekte auf der Homepage verzeichnet, die *Batik* nutzen.

CSIRO SVG

Ein von der *Commonwealth Scientific and Industrial Research Organisation* (kurz: CSIRO) entwickeltes *Java*-Toolkit zur Darstellung von SVG-Dateien. Alle Grafikprimitive und Animationen werden unterstützt, bis auf das Verändern von Pfaden, die entsprechenden Klassen sind jedoch vorbereitet. Was die projektspezifischen Anforderungen angeht, sind die Quellen für das CSIRO-Toolkit auf der Homepage verfügbar, das Toolkit wird jedoch nicht mehr weiterentwickelt. Da es einen stabilen Status hat, kommt trotzdem in Frage. Es wird im *Open Source* Projekt *X-Smile* eingesetzt. Dieses Projekte wird aktiv entwickelt (letztes *Release*: 31. Januar 2005), somit könnte man sich bei Fragen an Entwickler des Projekts wenden.

Andere SVG Toolkits

Neben diesen *Java*-Toolkits gibt es noch zahlreiche weitere, die nicht in *Java* geschrieben sind, deren Umfang aber nicht an die bereits vorgestellten Toolkits heranreicht. Die beiden *Java*-Toolkits, insbesondere das CSIRO-SVG-Toolkit weisen nahezu alle benötigten *Features* auf. Neben den eben erwähnten Projekten gib es noch zwei interessante: *SVGL* [24] und *Cairo* [8]. Das sind zwei Grafikbibliotheken, die mit SVG umgehen können und besonders in Bezug auf die Darstellungsgeschwindigkeit und Darstellungsqualität interessant erscheinen.

2.3.4. Übersicht der Toolkits

Folgende Toolkits kommen in Frage:

1. *Java*-2D
2. *Piccolo*
3. Apache Batik-SVG-Toolkit
4. CSIRO-SVG-Toolkit

Zomit wird nicht weiter betrachtet, da kein Quellcode verfügbar ist und *Piccolo* im Vergleich vom Umfang und Entwicklungsstand die bessere Alternative darstellt. Zusammenfassend sind hier noch einmal die Eigenschaften der einzelnen Toolkits im Vergleich (Tabelle 2.2) aufgeführt. Ein „+" in der Tabelle bedeutet, dass die Eigenschaft leicht mit dem Toolkit umgesetzt werden kann. Ein „–" bedeutet, dass sie komplett selbst nachgebildet werden müsste und ein „+/–" bedeutet, dass sie zum Teil vom Toolkit übernommen wird.

Eigenschaft	*Java*-2D	*Piccolo*	*Batik*	CSIRO
Zeichenoperationen	+/–	+/–	+	+
Hierarchie	–	+	+	+
Pfeile	–	+/–	+	+
Animationen	–	+/–	–	+
Zoom	–	+	+	+
Auszeichnungen	–	+/–	+	+

Tabelle 2.2.: Toolkits im Technischen Vergleich

Eine entsprechende Darstellung für die projektspezifischen Anforderungen befindet sich in Tabelle 2.3. Zahlen entsprechen Schulnoten von 1 (sehr gut) bis 6 (ungenügend).

Auf Grund der leichten Umsetzungsmöglichkeiten mit dem CSIRO-SVG-Toolkit wurde, trotz der schlechteren Noten in Bezug auf die projektspezifischen Anforderungen, dieses für die Umsetzung des *KIEL*-Browsers ausgewählt. Es werden hier

Eigenschaft	*Java*-2D	*Piccolo*	*Batik*	CSIRO
Verfügbarkeit	im JDK	Open Source	Open Source	Open Source
Aktualität	1	2	2	5
Anwenderbasis	1	3	2	3-

Tabelle 2.3.: Toolkits im Projekt Vergleich

alle benötigten *Features*, bis auf die Veränderung von Pfaden, unterstützt. Die Wahl wäre auf *Batik* gefallen, wenn das Toolkit Animationen unterstützen würde. Da die Unterstützung jedoch auf unbestimmte Zeit nicht möglich sein wird, wurde das CSIRO-Toolkit wegen der guten Animationsfähikeiten eingesetzt. Eine Implementation von Animationen für das *Batik*-Toolkit wäre viel zu zeitaufwendig gewesen. *Piccolo* und *Java*-2D kommen auf Grund des im Vergleich zu SVG-Toolkits wesentlich höheren Implementierungsaufwandes nicht in Frage.

3. Beschreibung der Funktionen im KIEL-Browser

Nachdem im vorherigen Kapitel die Grundlagen der Arbeit erklärt worden sind, sollen in den folgenden Abschnitten die umgesetzten Funktionen des *KIEL*-Browsers erläutert werden. Nähere Erläuterungen wie man den Browser bedient und die entsprechenden Ausgaben des Browsers findet man in der Bedienungsanleitung im Anhang A.

3.1. Steuerung eines Simulators

Die Hauptaufgabe des Browsers ist die Steuerung des Simulators und die Visualisierung der Simulation. Hier gibt es zwei verschiedene Möglichkeiten:

1. Die Simulation ohne Layoutunterstützung, also mit einer statischen Sicht.
2. Die Simulation mit layoutgestützten dynamischen Sichten.

3.1.1. Simulation mit einer statischen Sicht

Wird ein Statechart geladen oder erstellt und die Layoutunterstützung nicht aktiviert, findet die Simulation in der dargestellten Sicht statt. Im Sprachgebrauch von *KIEL* ist das `OriginalLayout` ausgewählt. Dabei wird der Programmfluss durch Markierungen dargestellt. Ein Beispiel einer Simulation ohne Layoutunterstützung befindet sich in Abbildung 3.1.

3.1.2. Simulation mit layoutgestützten dynamischen Sichten

Wählt man aus dem *Layouter*-Menü einen geeigneten *Layouter* aus, findet die Simulation in dynamischen Sichten statt. Dabei legt der Layouter die Art der dynamischen Visualisierung fest, der Browser dagegen stellt nur Sichten dar. Das hat den Vorteil, dass der Browser kein Wissen über das Statechart benötigt. Dadurch ist eine stärkere Kapselung möglich, durch die eine bessere Trennung der Module und damit der Arbeiten gewährleistet werden kann. Damit die dabei stattfindenden Veränderungen des Layouts gleitend stattfinden, wird von einer Sicht zu einer anderen Sicht animiert. Gleitende Veränderungen erhöhen die Anschaulichkeit der Veränderungen, da auf diese Weise die Veränderungen nicht genau überprüft werden

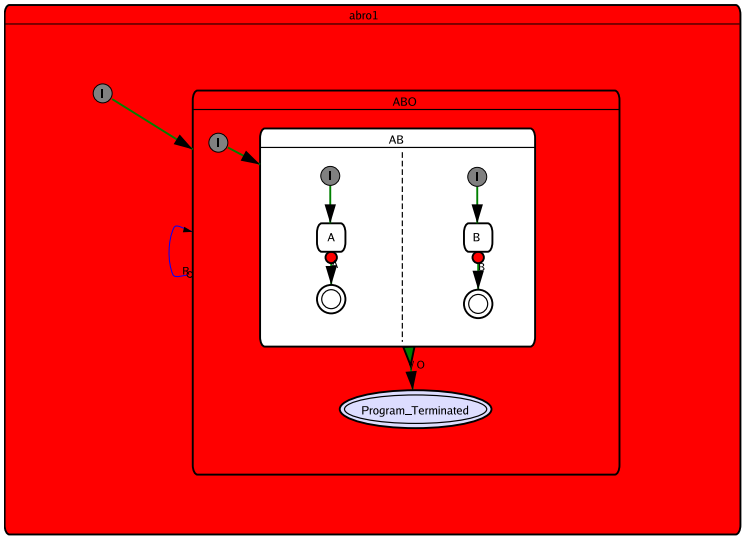


Abbildung 3.1.: Simulation mit originalem statischem Layout

müssen. Markierungen finden analog zur statischer Simulation statt. Ein Beispiel einer Simulation mit Layoutunterstützung befindet sich in Abbildung 3.2. Um den Animationseffekt zu verdeutlichen wurde die einzelnen Bilder übereinander gelegt.

3.1.3. Farbliche Hervorhebung aller MicroSteps

Ein *KIEL*-Simulator gibt als Ergebnis eines Simulationsschrittes eine Liste von `MicroSteps` zurück. Jeder dieser `MicroSteps` wird unterschiedlich markiert. Die Art der Markierungen ist komplett über CSS konfigurierbar (siehe Abschnitt 3.4).

3.1.4. Simulationsschritte in verschiedenen Granularitäten

Der Browser ermöglicht, die Liste von `MicroSteps` auf verschieden Arten zu visualisieren. Man kann die Liste auf einmal, in einer vorgegeben Geschwindigkeit oder durch die Bestätigung jeden Schritts, anzeigen lassen.

3.2. Ansicht der Baumstruktur eines Statecharts

Neben dem gezeichneten Statechart gibt es auch eine Baumansicht vom Statechart. Hier wird der Aufbau eines Statechart deutlich. Sind in Hierarchischen Zuständen, lokale Variablen oder Signale deklariert, werden diese ebenfalls in dieser Ansicht angezeigt. Um herauszufinden, welchem gezeichneten Zustand ein Element aus der Baumdarstellung entspricht, kann man die Zustände in dem Baum markieren, daraufhin wird der entsprechende Zustand in der Zeichenebene ebenso markiert.

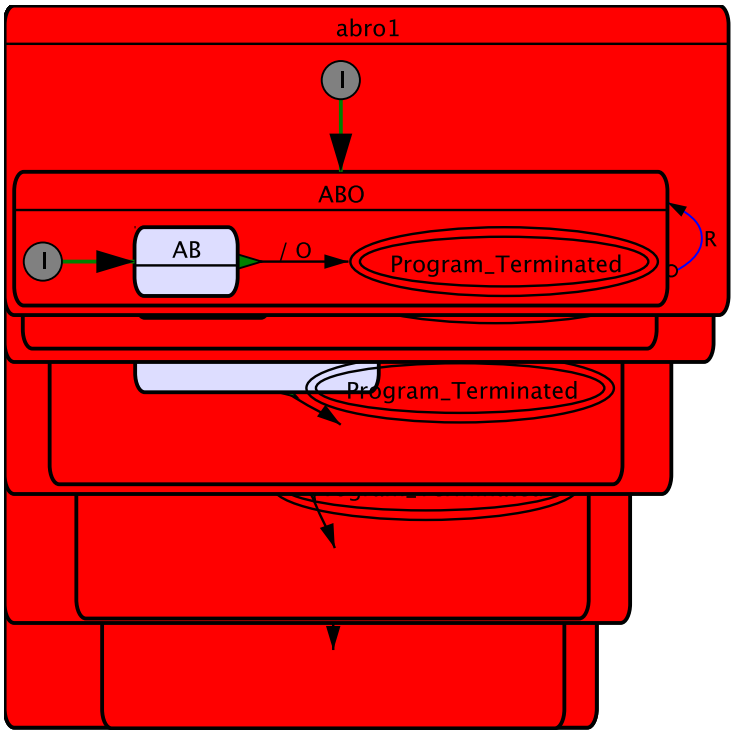


Abbildung 3.2.: Simulation in layoutgestützten dynamischen Sichten

3.3. Zoom

Ist ein Statechart zu groß für die zur Verfügung stehenden Zeichenfläche, kann man die *Scrollbars* nutzen. Alternativ kann man es mit Hilfe des Mausrades das Statechart zoomen. Der so gewählte Zoomfaktor bleibt auch während der Simulation bestehen. Erst der *Reset-Button* setzt auch den Zoomfaktor wieder zurück.

3.4. Konfigurierbarkeit

Der Browser lässt sich mit verschiedenen Dateien konfigurieren. Die meisten Optionen kann man in der Datei `browser.properties` einstellen. Werden hier Werte außerhalb der entsprechenden Wertebereiche eingetragen, wird auf *default* Werte zurückgegriffen.

`Browser.Title`**:** Beschreibt den Titel des Programms, der dem `KielFrame` mitgeteilt wird.

`Browser.AlwaysStepThrough`**:** Man kann dieses Flag auf `true` oder `false` setzten. Ist das Flag auf `true` gesetzt werden beim Klick auf den *MacroStep-Button* nicht alle `MicroSteps` auf einmal gezeichnet sondern nacheinander im Abstand von `MicroStep.SleepTime`.

`Browser.LogLevel:` Erwartet wird ein Wert zwischen `0` und `10`. Dieser Wert gibt in Kombination mit `Browser.LogCompare` an welche Mitteilungen des Browser in dem Browser- Log erscheinen sollen.

`Browser.LogCompare:` Mögliche Werte sind `<=`,`==` oder `>=`. Sie geben die Vergleichsoperation an, mit der der `Browser.LogLevel` mit dem jeweiligen Wert des auszugebenden `Strings` verglichen wird.

`BrowserTree.MarkNodeColor:` Möglicher Wert ist ein RGB Hexadezimales Tripel der Form `#RGB`. Die so definierte Farbe wird zur Markierung in der Baumansicht bzw. auf der Zeichenfläche genutzt.

`BrowserTables.InputEventColor:` Siehe `BrowserTables.VariablesColor`.

`BrowserTables.OutputEventColor:` Siehe `BrowserTables.VariablesColor`.

`BrowserTables.LocalEventColor:` Siehe `BrowserTables.VariablesColor`.

`BrowserTables.VariablesColor:` Möglicher Wert ist ebenfalls ein RGB Hexadezimales Tripel der Form `#RGB`. Es beschreibt die entsprechende Farbe in den Signal Tabellen.

`Animation.Start:` Möglicher Wert ist eine positive Fließkommazahl. Der Wert bestimmt die Zeit in Sekunden, die gewartet wird, bis die Objekte sich während einer Animation bewegen.

`Animation.Duration:` Möglicher Wert ist eine positive Fließkommazahl. Der Wert bestimmt die Dauer einer Animation.

`MicroStep.ExecuteTransition:` Möglicher Wert ist ein CSS [**?**] konformer String. Er legt die Markierung einer ausgeführten Transition fest. Als Standardwert ist „`fill:none; stroke:green; stroke-width:2.0 px;`" eingetragen, also eine grüne Linie mit einer Dicke von zwei Pixeln.

`MicroStep.TestTransition:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung einer getesteten Transition fest. Als Standardwert ist „`fill: none; stroke:blue; stroke-width:1.0 px;`" eingetragen, also eine grüne Linie mit einer Dicke von einem Pixel.

`MicroStep.OnInside:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der eine *OnInside* Aktion ausführt. Als Standardwert ist „`fill:gray;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer grauen Füllung.

`MicroStep.OnExit:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der eine *OnExit* Aktion ausführt. Als Standardwert ist „`fill:green;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer grünen Füllung.

`MicroStep.OnEntry:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der eine *OnEntry* Aktion ausführt. Als Standardwert ist „`fill:blue;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer blauen Füllung.

`MicroStep.StateSuspended:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der gerade suspendiert wird. Als Standardwert ist „`fill:yellow;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer gelben Füllung.

`MicroStep.StateActivated:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der gerade aktiviert wird. Das ist nicht zu verwechseln mit dem gerade aktiven Zustand eines Simulationsschrittes. Als Standardwert ist „`fill:red;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer roten Füllung.

`MicroStep.StateDeactivated:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes fest, der gerade deaktiviert wird. Als Standardwert ist „`fill:white;stroke:black;stroke-width:2.0px;`" eingetragen, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer weißen Füllung.

`MicroStep.ActualConfigState:` Möglicher Wert ist ein CSS konformer String. Er legt die Markierung eines Zustandes aus der aktuellen Konfiguration fest. Das entspricht dem aktiven Zustand eines Simualtionsschrittes. Als Standardwert wird hier „`fill:#ddddff;stroke:black;stroke-width:2.0px;`" genutzt, also eine schwarze Umrandung mit einer Dicke von zwei Pixeln und einer hellen blauen Füllung.

`MicroStep.SleepTime:` Möglicher Wert ist ein positiver ganzzahliger Wert. Der entspricht in Millisekunden der Wartedauer zwischen zwei `MicroSteps`. Die Wartedauer entsteht, wenn nach einem `MicroStep` der *continue-Button* gedrückt wird oder wenn `Browser.AlwaysStepThrough` auf `true` gesetzt ist und dann der *MacroStep-Button* gedrückt wird.

Neben der `browser.properties` Datei gibt es noch eine weitere wichtige Konfigurationsdatei, sie heißt `estudio.ini`. Dort werden die Templates festgelegt mit deren Hilfe das Statechart erzeugt wird. Es gibt dort folgende Einstellungsmöglichkeiten:

`SVG.ANDState:` Verweis auf eine SVG-Template Datei für `ANDState`. Das gerenderte Template entspricht der Abbildung 3.3.

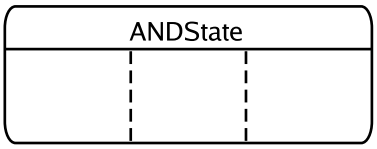


Abbildung 3.3.: Mitgeliefertes ANDState Template

`SVG.ORState:` Verweis auf eine SVG-Template Datei für `ORState`. Das gerenderte Template ist in Abbildung 3.4 dargestellt.

Abbildung 3.4.: Mitgeliefertes ORState Template

`SVG.SimpleState:` Verweis auf eine SVG-Template Datei für `SimpleState`. Das gerenderte Template ist in Abbildung 3.5 dargestellt.

Abbildung 3.5.: Mitgeliefertes SimpleState Template

`SVG.Initial:` Verweis auf eine SVG-Template Datei für `InitialState`. Das gerenderte Template ist in Abbildung 3.6 dargstellt.

Abbildung 3.6.: Mitgeliefertes InitialState Template

`SVG.Suspend:` Verweis auf eine SVG-Template Datei für `Suspend` Zustände. Das gerenderte Template ist in Abbildung 3.7 dargestellt.

Abbildung 3.7.: Mitgeliefertes Suspend Template

`SVG.History`: Verweis auf eine SVG-Template Datei für `History` Zustände. Das gerenderte Template ist in Abbildung 3.8 dargestellt.

Abbildung 3.8.: Mitgeliefertes History Template

`SVG.DeepHistory`: Verweis auf eine SVG-Template Datei für `DeepHistory` Zustände. Das gerenderte Template ist in Abbildung 3.9 darsgestellt.

Abbildung 3.9.: Mitgeliefertes DeepHistory Template

`SVG.Choice`: Verweis auf eine SVG-Template Datei für `Choice` Zustände. Das gerenderte Template ist in Abbildung 3.10 dargestellt.

Abbildung 3.10.: Mitgeliefertes Choice Template

`SVG.Final`: Verweis auf eine SVG-Template Datei für `FinalState`. Das gerenderte Template ist in Abbildung 3.11 dargestellt.

Abbildung 3.11.: Mitgeliefertes FinalState Template

`SVG.FinalComposite`: Verweis auf eine SVG-Template Datei für `FinalANDState` oder `FinalORState`. Das gerenderte Template ist in Abbildung 3.12 dargestellt.

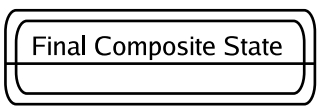


Abbildung 3.12.: Mitgeliefertes FinalANDState oder FinalORState Template

`CSS.RECT`: Dieser CSS-String wird auf alle rechteckigen Außenlinien angewandt, z. B. bei bestimmten Zuständen.

`CSS.CIRCLE`: Dieser CSS-String wird auf alle runden Außenlinien angewandt, z. B. bei allen Pseudo-Zuständen.

`CSS.LINE`: Dieser CSS-String wird auf alle Linien angewandt, z. B. auf Transitionen.

Eine detaillierte Beschreibung wie man solche SVG-Templates erstellt befindet sich in im Abschnitt 4.3.1.

4. Implementierung des KIEL-Browsers

Nachdem im vorherigen Abschnitt die Umgesetzten Funkionen beschrieben worden sind wird in diesem Abschnitt die Implementierung dieser Funktionen näher erläutert. Dabei wird auch auf das Umfeld innerhalb des Projekts *KIEL* eingegangen. Die einzelnen Klassen und der Datenfluss werden ebenso beschrieben, wie das Design des Browsers.

4.1. Einbindung und Rahmenbedingungen im Projektumfeld KIEL

Das Ziel des Projekts *KIEL* ist es ein Modellierungswerkzeug zu entwickeln. Der in dieser Arbeit entstandene Browser ist Teil des Projekts. Die Schnittstellen zu anderen Modulen sind bei diesem Projekt wohldefiniert, von Anfang an wurde auf Modularität geachtet. Der Browser hat Abhängigkeiten von zwei weiteren Modulen, dem Layouter und dem Simulator. Diese Abhängigkeiten beziehen sich nur auf besondere, als Schnittstellen vorgesehene Klassen. Dadurch ist es möglich, alle Layouter bzw. Simulatoren, die über die Schnittstellen verfügbar sind, zu nutzen. Die Hauptkomponente, durch die Klasse `KielFrame` implementiert, wird nur in einer Klasse benutzt. Die Klasse `Browser` stellt die alleinige Schnittstelle für das `KielFrame` dar, indem sie ein `KielComponent` implementiert. Der Vorteil eines solchen Ansatzes ist, dass es dadurch möglich ist, den Browser auch ohne das `KielFrame` zu nutzen.

Die Entwicklung im Projekt *KIEL* findet unter gemeinsamen Richtlinien statt. Dafür gibt es ein *Project-Manual*, welches Regeln und Konventionen enthält. Das komplette Projekt wird mit einer Versionsverwaltung verwaltet und es gibt es ein *Bug-Tracking-System*, um Fehler zu dokumentieren und zu beheben. Der Kontakt mit den einzelnen Entwicklern findet meist per *Email* statt, dazu gibt es eine eigene *Mailing-Liste* mit einem webbasierten Archiv.

Zu den gemeinsamen Richtlinien zählt ein einheitlicher Programmierstil der mit Hilfe eines *Stylecheckers* überprüft wird. Außerdem finden sogenannte *Code-Reviews* statt, die dazu geeignet sind, einen Einblick in den *Code* der anderen Entwickler zu bekommen und dessen *Code* auf schlechten Stil bzw. sonstige Verbesserungsmöglichkeiten zu untersuchen.

4.2. Klassen des Browsers

Der Browser besteht aus vier *Java-Packages*: `kiel.browser`, `kiel.browser.model`, `kiel.browser.view`, `kiel.browser.controller`. Die drei Unterpackages existieren auf Grund des *Model-View-Controller* (kurz: MVC) Ansatzes [10] des Browsers. Dabei enthält das Package `kiel.browser.model` alle Klassen, die das Datenmodell des Browsers verwalten. Hier befinden sich folgende Klassen:

`BrowserModel:` Dies ist die Hauptklasse des Browsers. Hier werden alle Daten des Browsers verwaltet, Änderungen, die über die GUI gemacht werden, wirken sich auf diese Klasse aus. Ändert sich diese Klasse, werden Nachrichten an die Klassen aus dem Package `kiel.browser.view` gesandt, diese können aufgrund der Änderungen feststellen, ob sich Ihre Darstellung ändern muss.

`ExpInformation:` Klasse die Informationen über eine Variable oder ein Signal enthält. Instanzen dieser Klasse werden von den Signal bzw. Variablentabellen dargestellt.

`ModelMessage:` Instanzen dieser Klasse werden vom BrowserModel verschickt, wenn sich Änderungen im Model ergeben haben. Eine Nachricht kann einen Parameter haben. Dieser Parameter ist vom Typ `String`.

`SignalTableModel:` Diese Klasse erweitert das `AbstractTableModel` aus dem Package `javax.swing.table`. Dies ist nötig, um die Signal bzw. Variablen Tabellen effektiv erzeugen zu können. Die Tabellen enthalten beliebige Objekte vom Typ `ExpInformation`. Das `BrowserModel` erzeugt und verwaltet drei `SignalTableModels`, ein Model für die *Input*-Signale, eines für die *Output*-Signale und eines für die lokalen Signale bzw. Variablen.

Das `kiel.browser.view` Package enthält alle Klassen, die GUI bzw. graphische Elemente erzeugen. Es finden sich hier folgende Klassen:

`BrowserGUI:` Eine Abstrakte Klasse, die alle Klassen erweitern müssen, die etwas darstellen wollen. Jede dieser Klassen gibt ein `JComponent` zurück, das in eine Anwendung integriert werden kann. Es verbindet auch die Klassen mit dem `BrowserModel`, so dass sie dessen Nachrichten empfangen können.

`BrowserCanvas:` Die Klasse entspricht der Zeichenfläche. Dazu wird aus der *KIEL*-Datenstruktur eine SVG-Datenstruktur erzeugt, diese wird an ein SVG-Toolkit übergeben, dieses sorgt für die Darstellung bzw. die Animation. Während einer Simulation ist diese Klasse dafür zuständig, die passenden Elemente in der SVG-Datenstruktur zu finden und entsprechend einzufärben.

`BrowserMenuBar:` Diese Klasse enthält den `JMenuBar`. Die angebotenen Menüs sind ein Layouter Menü, in dem man zwischen den verschiedenen Layoutern auswählen kann und ein *Configurations* Menü, in dem man die darzustellende *Configuration* auswählt. Das Auswählen eines Menüpunktes löst eine Aktion in der `MenuController` Klasse des `kiel.browser.controller` Packages aus.

`BrowserToolbar`: Diese Klasse stellt die *Toolbar* dar, über den ein Simulator gesteuert werden kann. Das Drücken der Buttons löst eine Aktion in der `SimulationController` Klasse des `kiel.browser.controller` Packages aus.

`BrowserTree`: Diese Klasse erzeugt eine Baumansicht des aktuellen Statecharts. Diese Klasse enthält eine eingebettete Klasse `MyRenderer`, die für die ansprechende Darstellung der Baumansicht zuständig ist. Wird ein Element im Baum markiert, wird das `BrowserModel` über die Methode `markElement(Node n)` bzw. `removeMarkOnElement(Node n)` darüber informiert.

`GraphicalObjectsLibrary`: In dieser Klasse werden die SVG-Templates verwaltet. Nach Übergabe einer entsprechenden Datei (siehe Abschnitt 3.4) werden die Templates geladen und bearbeitet. Tritt ein Fehler beim Parsen der Templates auf, wird auf ein Set von korrekten Templates als *Resource* zurückgegriffen.

`ResourceLoader`: Dies ist eine *Utility*-Klasse, mit der man als *Resource* eingebundene Bilder als `ImageIcon` laden kann.

`SignalTable`: Hier wird eine Tabelle mit `SignalTableModel` Daten dargestellt. Die Tabelle passt sich in der Breite dem verfügbaren Platz an, daher implementiert diese Klasse einen `ComponentListener`. Es ist eine eingebettete Klasse `ExpressionCellRenderer` enthalten, der für die ansprechende Darstellungen der Tabellenzellen sorgt. Die Möglichkeit Signale per Mausklick an bzw. abzuwählen, wird über das `SignalTableModel`, welches hinter der Tabelle steht, bewerkstelligt.

Der Controller Teil des MVC Ansatzes übernimmt das Package `kiel.browser.controller`. Hier sind folgende Klassen enthalten:

`MenuController`: Hier werden die Aktionen, die durch wählen eines Menüpunktes passieren sollen, ausgeführt. Dabei wird das `BrowserModel` geändert. Das schickt dann entsprechende Nachrichten an alle registrierten `BrowserGUI` Klassen und diese entscheiden, ob sich durch die Änderungen ein neues Aussehen ergibt und ein *Redraw* nötig ist.

`SimulationController`: Hier werden die Aktionen ausgeführt, die durch Drücken eines *Toolbar-Buttons* ausgelöst werden. Es werden ebenfalls Änderungen am `BrowserModel` vorgenommen (z.B. wird ein Simulator dazu bewegt den nächsten Schritt zu berechnen). Das Verschicken von `ModelMessages` resultiert wiederum aus den Änderungen am `BrowserModel`. Diese Klasse erweitert die `Thread` Klasse. Der erzeugte *Thread* sorgt dafür, dass im Hintergrund bei ausgewähltem `Browser.AlwaysStepThrough` und einem Klick auf den Macro-Step Button oder einem Klick auf den continue... Button die `MicroSteps` mit Pausen hintereinander abgespielt werden (siehe Abschnitt 3.4).

Im `kiel.browser` Verzeichnis befinden sich weitere Klassen, die sich nicht auf den MVC Ansatz beziehen:

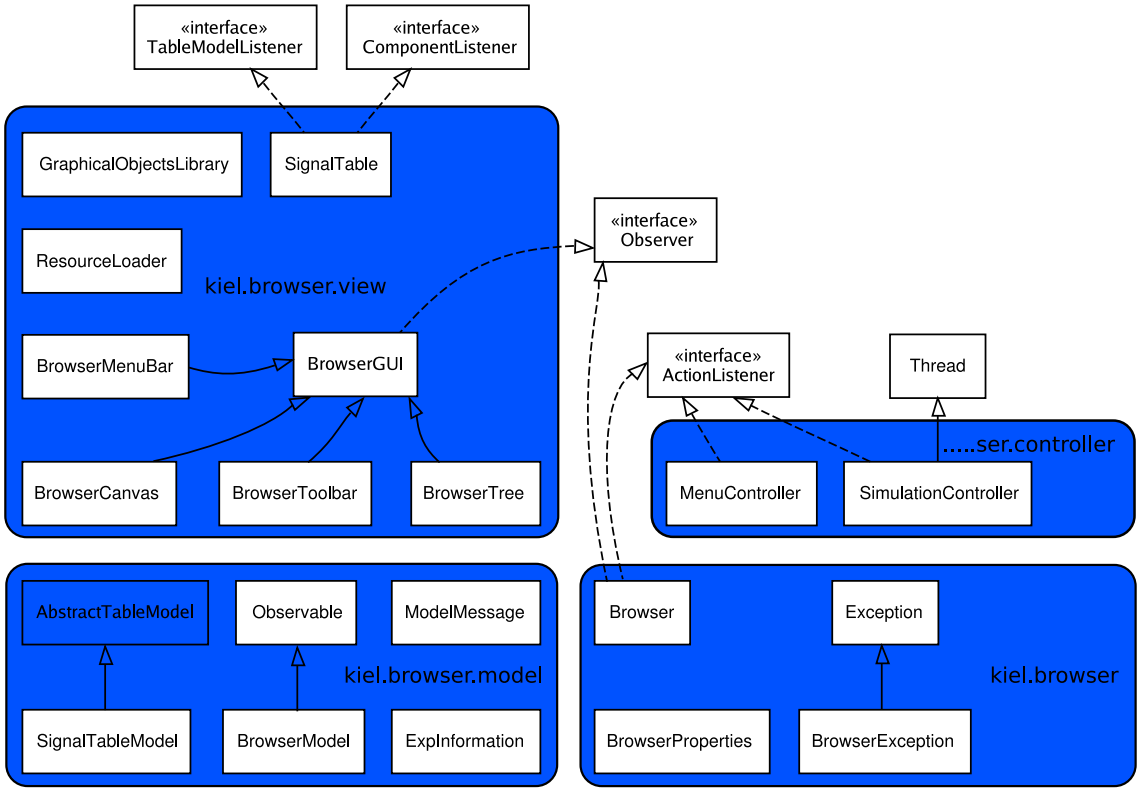


Abbildung 4.1.: Alle Browserklassen im Überblick

`BrowserException:` Diese Klasse ist eine Browser eigene *Exception* Klasse. Muss auf Grund eines Fehlers eine *Exception* geworfen werden, wirft der Browser diese *Exception*.

`Browser:` Dies Klasse stellt das *Interface* zum `KielFrame` dar. Diese Klasse implementiert ein `KielComponent` (siehe Abschnitt 4.1). Es werden ein `BrowserModel` und alle oben vorgestellten `BrowserGUI` Klassen instanziert. Als zusätzliche Komponenten werden hier zwei `JTextPanes` für die Ausgabe der `LogFile` Klassen des `BrowserModel` und des ausgewählten Simulators erzeugt. Außerdem wird eine neue `SignalTable` erzeugt, die alle Signale und Variablen eines Statecharts enthält. Diese Klasse sorgt auch dafür, dass Warnungen und *Exceptions* vom BrowserModel durch entsprechende Methoden im `KielFrame` dargestellt werden.

In Abbildung 4.1 befindet sich eine Übersicht aller Klassen des Browsers. Für eine detaillierte Darstellung aller Methoden der einzelnen Klassen ist auf die *JavaDoc* Dokumentation [25] des gesamten Projekts *KIEL* zu verweisen.

4.3. Statechart Rendering

Um aus einer vorhandenen *KIEL*-Datenstruktur eine SVG-Datenstrukur zu erzeugen wird das komplette Statechart einmal rekursiv durchlaufen und für jeden gefundenen Zustand eine Zeichenroutine aufgerufen. Enthält ein Zustand ausgehende Transitionen wird ebenfalls für die Transition eine Zeichenroutine aufgerufen (siehe Auflistung 4.1).

Auflistung 4.1: Algorithmus zur Erzeugung der SVG-Datenstruktur

Beide Zeichenroutinen geben ein SVG-Gruppenelement zurück, mit Hilfe von solchen Gruppierungen lassen sich Hierarchien erzeugen. SVG-Gruppenelemente haben folgende nützliche Eigenschaften:

- Ändert man die Position einer Gruppe, werden alle enthaltenen Objekte mitbewegt.
- Alle Positionsangaben der beinhaltenden Elemente sind relativ zum Gruppenursprung.

Diese Eigenschaften entsprechen genau den Angaben in den `LayoutInformation` daher können die Position- und Größenangaben direkt übernommen werden. Bei Transitionen enthält die `LayoutInformation` eine Liste von Objekten vom Typ `PathElement`, diese lassen sich ebenfalls direkt in SVG-Pfade umsetzen. SVG bietet die Möglichkeit Pfaden Start- und Endmarkierungen zu geben, durch solche `SVGMarkerElemente` werden Pfeile entsprechend ihres Types dargestellt. Der Aufbau eines Statecharts innerhalb einer SVG Datenstruktur ist anhand eines Beispiels in Abbildung 4.2 illustriert. Wie man aus den Templates mit Hilfe der `LayoutInformation` ein SVG Gruppenelement erzeugt, wird im folgenden Abschnitt erläutert.

4.3.1. Eigene SVG-Templates erstellen

Um eigene SVG-Templates zu erstellen, benötigt man Wissen darüber wie aus den Templates die gezeichneten SVG-Gruppenelemente erzeugt werden. Ein Template ist eine SVG-Datei, die einen bestimmten Aufbau haben muss und über *Ids* SVG Objekten bestimmte Aufgaben zuordnet. In Bezug auf den Aufbau der SVG-Datei wird vorgeschrieben, dass alle Elemente, die dargestellt werden sollen, innerhalb eines Gruppenelementes liegen müssen. Es wird jeweils das erste Gruppenelement zur Darstellung genutzt. Innerhalb dieser Gruppe können beliebige SVG Elemente enthalten sein. Die Elemente werden unverändert übernommen, außer es treten folgende Kombinationen auf:

`<rect id="rect_state">`**:** Hier werden die Attribute `width` und `height` auf die Größe des Zustandes angepasst.

`<rect id="relrect_state">`: Hier werden die Attribute `width` und `height` auf die Größe des Zustandes angepasst. Die vorhanden Werte der Attribute `width` und `height` werden auf die tatsächliche Breite und Höhe addiert.

`<line id="line">`: Hier wird das Attribut `x2` für auf die Breite des Zustandes angepasst.

`<text id="label">`: Hier wird der Name des Zustandes eingefügt und die Position wird angepasst. Bei `CompositeState`s wird die Position mittig im oberen abgegrenzten Bereich festgelegt, sonst horizontal und vertikal zentriert über den gesamten Zustand.

`<ellipse id="circle_state">`: Hier werden die Attribute `cx`, `cy`, `x` und `y` auf die Größe des Zustandes angepasst.

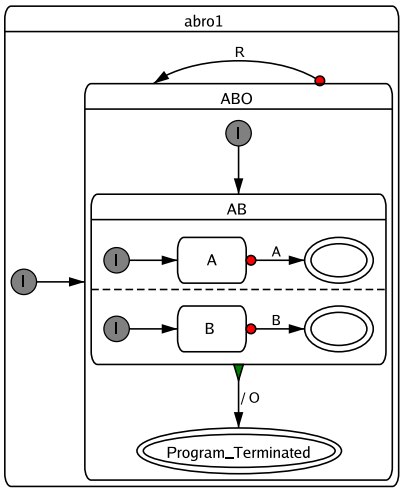
`<ellipse id="relcircle_state">`: Hier werden die Attribute `cx`, `cy`, `x` und `y` auf die Größe des Zustandes angepasst. Die vorhandenen Werte der Attribute `cx` und `cy` werden auf die tatsächlichen Werte addiert.

`<g id="content">`: Diese Gruppe wird skaliert um das selbe Verhältnis wie sich der *rect_state* verändert hat.

Es sollte mindestens ein SVG-Rect-Element mit *ID rect_state* oder ein SVG-Ellipse-Element mit *ID circle_state* in dem Template enthalten sein. Diese Elemente werden für die Einfärbung während der Simulation gesucht. Ein Beispiel für ein erstelltes Template befindet sich in Auflistung 4.3.

Auflistung 4.3: Beispiel eines SVG Templates für *Suspend* Zustände

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD␣SVG␣1.0//EN" "svg-20000110.dtd">
3  <!-- Created with Inkscape (http://www.inkscape.org/) -->
4  <svg
5     id="svg1"
6     xmlns="http://www.w3.org/2000/svg"
7     width="500"
8     height="500">
9    <g>
10           <ellipse id="circle_state"
11              cx="6"
12              cy="6"
13              rx="6"
14              ry="6"
15              style="fill:yellow;␣stroke:black;" />
16           <text text-anchor="middle" x="5" y="2" id="label">S</text>
17   </g>
18 </svg>
```



(a) Gerenderte Grafik

Auflistung 4.2: Aufbau einer gerenderten *KIEL* Datenstruktur als SVG

```
<?xml version="1.0" standalone="no" ?>
<svg height="383.0" width="315.0" x="5.0" y="5.0">
  <g transform="matrix(1␣0␣0␣1␣0␣0)">
    ORState Template --> abro1
    <g>
      Transitionen
    </g>
    <g transform="matrix(1␣0␣0␣1␣63␣61)">
      ORState Template --> ABO
      <g>
         Transition
      </g>
      <g transform="matrix(1␣0␣0␣1␣5␣87)">
        ANDState Template --> AB
        <g transform="matrix(1␣0␣0␣1␣5␣21)">
             Region 1/2
          <g>
            Transition
          </g>
          <g transform="matrix(1␣0␣0␣1␣63␣13)">
            SimpleState Template --> A
          </g>
          <g>
            Transition
          </g>
          <g transform="matrix(1␣0␣0␣1␣5␣21)">
            Inital State Template
          </g>
          <g transform="matrix(1␣0␣0␣1␣163␣13)">
            Final State Template
          </g>
        </g>
        <g transform="matrix(1␣0␣0␣1␣5␣75)">
            Region 2/2
          <g>
            Transition
          </g>
          <g transform="matrix(1␣0␣0␣1␣63␣13)">
            SimpleState Template --> B
          </g>
          <g>
            Transition
          </g>
          <g transform="matrix(1␣0␣0␣1␣5␣21)">
            Initial State Template
          </g>
          <g transform="matrix(1␣0␣0␣1␣163␣13)">
            Final State Template
          </g>
        </g>
      </g>
      <g transform="matrix(1␣0␣0␣1␣41␣271)">
        Final State Template --> Program_Terminated
      </g>
      <g>
        Transition
      </g>
      <g transform="matrix(1␣0␣0␣1␣111␣29)">
        Initial State Template
      </g>
    </g>
    <g>
      Transition
    </g>
    <g transform="matrix(1␣0␣0␣1␣5␣207)">
      Inital State Template
    </g>
  </g>
</svg>
```

(b) Aufbau als SVG

Abbildung 4.2.: Aufbau einer gerenderten *KIEL* Datenstruktur als SVG

5. Ergebnisse

Im vorherigen Abschnitt wurde die Implementierung des Browsers näher erläutert. In diesem Abschnitt werden die Ergebnisse der Arbeit nochmals zusammenfassend erläutert. Aufgabe war es, im Rahmen des Projektes *KIEL* einen Browser für das Anzeigen und Steuern einer Simulation zu entwickeln. Dabei sollte die durch das Projekt *KIEL* geprägte dynamische Visualisierung von Statecharts ebenso umgesetzt werden, wie die klassische Darstellung. Diese Aufgabe wurde mit Hilfe eines SVG-Toolkits umgesetzte. Dieses erlaubt eine nahezu direkte Umsetzung der gegebenen *KIEL*-Datenstruktur in eine SVG-Datenstruktur. Durch die Benutzung von SVG wurde eine sehr hohe Konfigurierbarkeit erzielt. Dadurch ist es möglich sämtliche in der Zeichenfläche sichtbaren Objekte zu verändern ohne eine Zeile *Java-Code* zu verändern. Ebenso ist eine Erweiterung auf individuell darstellbare Zustände wie es in einem Vortrag [22] vorgeschlagen worden ist, einfach möglich. Der zweite Teilaspekt dieser Arbeit ist die Steuerung eines Simulators und wurde ebenfalls komplett umgesetzt. Es werden alle Informationen, die ein Simulator-Modul zur Verfügung stellt ausgewertet und in entsprechende Markierungen umgesetzt. In diesem Zusammenhang ist es ist ebenso möglich, Eingabesignale zu setzen oder ihren Wert zu ändern.

Die Darstellungsqualität ist im Vergleich zu anderen Modellierungswerkzeug sehr hoch, das lässt sich auf den vektoriellen Ansatz des SVG-Toolkits zurückführen. Alle Linien sind weichgezeichnet, dadurch sind keine harten Kanten und störende Pixel zu erkennen. Auch die Qualität der Animationen ist hoch. Um eine flüssige Animationen zu erzeugen, ist jedoch ein leistungsstarker Rechner nötig. Die Bilder pro Sekunde einer Animation erhöhen sich nicht wesentlich wenn man die Darstellungsqualität des SVG-Toolkits heruntersetzt. Ein Geschwindigkeitsvergleich befindet sich in Tabelle 5.1. Hier wurde zwei verschiedene Qualitätseinstellungen (mit Anialiasing und ohne Anialiasing) und zwei verschiedene Darstellungsmöglichkeiten (mit Hardwareunterstützung (`VolatileImage`) und ohne Hardwareunterstützung (`BufferedImage`)) auf ihre Geschwindigkeit hin untersucht.

	`VolatileImage`	`BufferedImage`
Antialiasing	3.27 fps	3.03 fps
Kein Antialiasing	3.31 fps	3.27 fps

Tabelle 5.1.: Geschwindigkeitsvergleich mit verschiedenen Antialiasing Einstellungen[1]

5.1. Mögliche Erweiterungen

Als mögliche Erweiterung für den Browser ist die Ausweitung des Template Ansatzes auf jeden Zustand zu nennen. Damit wäre es möglich, jedem Zustand ein individuelles Aussehen zu geben. Eine weitere Idee ist es, in der Zeichenfläche ein Zustand auswählen zu können und diesen dann in der Baumstruktur zu markieren. Die Signaltabellen könnte man um eine *History*-Funktion ergänzen um sich den Verlauf der Signale bzw. Variablen über den gesamten Zeitraum der Simulation anschauen zu können. Hier wäre auch eine entsprechende Visualisierung denkbar. Zusätzlich sind neue Darstellungsweisen für die Simulation denkbar, die besonders bei Statecharts mit vielen parallelen Zuständen für einen besseren Überblick über die wesentlichen Teile einer Simulation sorgen. Hier ist als erster Ansatz die schon im Editor umgesetzte *Layering*-Technik zu nennen.

[1]Durchschnittswerte bei 10 Durchläufen unter Linux mit Java 1.4.2 auf einem Athlon XP 1700

6. Literaturverzeichnis

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

[2] Charles André. Semantics of S.S.M (Safe State Machine). Technischer bericht, I3S, Sophia-Antipolis, France, 2003. URL `http://www.esterel-technologies.com/v3/?id=50399\&dwnID=48`.

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

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

[5] Batik. Apache XML Batik, 2005. URL `http://xml.apache.org/batik/`. Batik is a Java(tm) technology based SVG toolkit.

[6] Ben Bederson und Jon Meyer. Implementing a zooming user interface: Experience building Pad++. *Software – Practice and Experience*, 28(10): 1101–1135, August 1998. ISSN 0038-0644. URL `http://www3.interscience.wiley.com/cgi-bin/fulltext?ID=10007356&;PLACEBO=IE.pdf;http://www3.interscience.wiley.com/cgi-bin/abstract?ID=10007356`.

[7] Wilfried Brauer. *Automatentheorie*. Teubner, 1984.

[8] cairo. Cairo, 2005. URL `http://cairographics.org/introduction`. Cairo is a vector graphics library designed to provide high-quality display and print output.

[9] Esterel Technologies. Company homepage. `http://www.esterel-technologies.com`.

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

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

[12] Tobias Kloss. Flexibles und Automatisiertes Layout von Statecharts. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Juli 2003.

[13] Tobias Kloss. Automatisches Layout von Statecharts unter Verwendung von GraphViz. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Mai 2005.

[14] Oliver Köth. Semantisches Zoomen in Diagrammeditoren am Beispiel von UML. Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2001.

[15] Lars Kuehl. Transformation von Esterel nach Esterel Studio. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, September 2005.

[16] Florian Lüpke. Implementierung eines Statechart-Editors mit layoutbasierten Bearbeitungshilfen. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Juni 2005.

[17] Mathworks Inc. *Stateflow and Stateflow Coder for use with Simulink — User's Guide*. Mathworks Inc., 6 edition, 2004.

[18] Mathworks Inc. Simulink Documentation, 2005. `http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/`.

[19] André Ohlhoff. Simulating the Behavior of Syncharts. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, November 2004.

[20] Piccolo. Piccolo Homepage, 2005. URL `http://www.cs.umd.edu/hcil/piccolo`. A Structured 2D Graphics Framework.

[21] Stuart Pook, Eric Lecolinet, Guy Vayssaix und Emmanuel Barillot. *Context and Interaction in Zoomable User Interfaces*. ACM Press, 2000.

[22] Steffen Prochnow. Kiel - visualizing dynamics of statecharts, Mai 2005. `http://rtsys.informatik.uni-kiel.de/~rt-kiel/kiel/documents/talks/oberseminar-0505-spr/talk.pdf`.

[23] Steffen Prochnow und Reinhard von Hanxleden. Visualisierung komplexer reaktiver Systeme – Annotierte Bibliographie. Technischer Bericht 406, Christian-Albrechts-Universität Kiel, Institut für Informatik und Praktische Mathematik, Juni 2004.

[24] svgl. SVGL, 2005. URL `http://svgl.sourceforge.net/`. svgl is a library that displays SVG graphics using OpenGL.

[25] The KIEL Project. Project API Documentation, 2004. URL http://www.informatik.uni-kiel.de/~rt-kiel/kiel/kiel/doc/. Kiel Integrated Environment for Layout.

[26] W3C. W3C SVG Homepage, 2005. URL http://www.w3.org/Graphics/SVG/. XML Graphics for the Web.

[27] Mirko Wischer. Ein FileInterface für das KIEL Projekt. Praktikumsbericht, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, 2005.

[28] Zomit. Zomit: A zoomable user interface. URL http://www.infres.enst.fr/net/zomit/. A ZUI Toolkit.

A. Bedienungsanleitung

In diesem Abschnitt wird die Bedienung des Browsers beschrieben, zur Konfigurierung sei auf Abschnitt 3.4 verwiesen.

Browser starten: Wird das *KIEL* Hauptprogramm gestartet, befindet es sich im Editiermodus, um in den Browser zu wechseln muss das Browser *Icon* gedrückt werden. Die Oberfläche des Browser ist in Abbildung A.1 zu sehen.

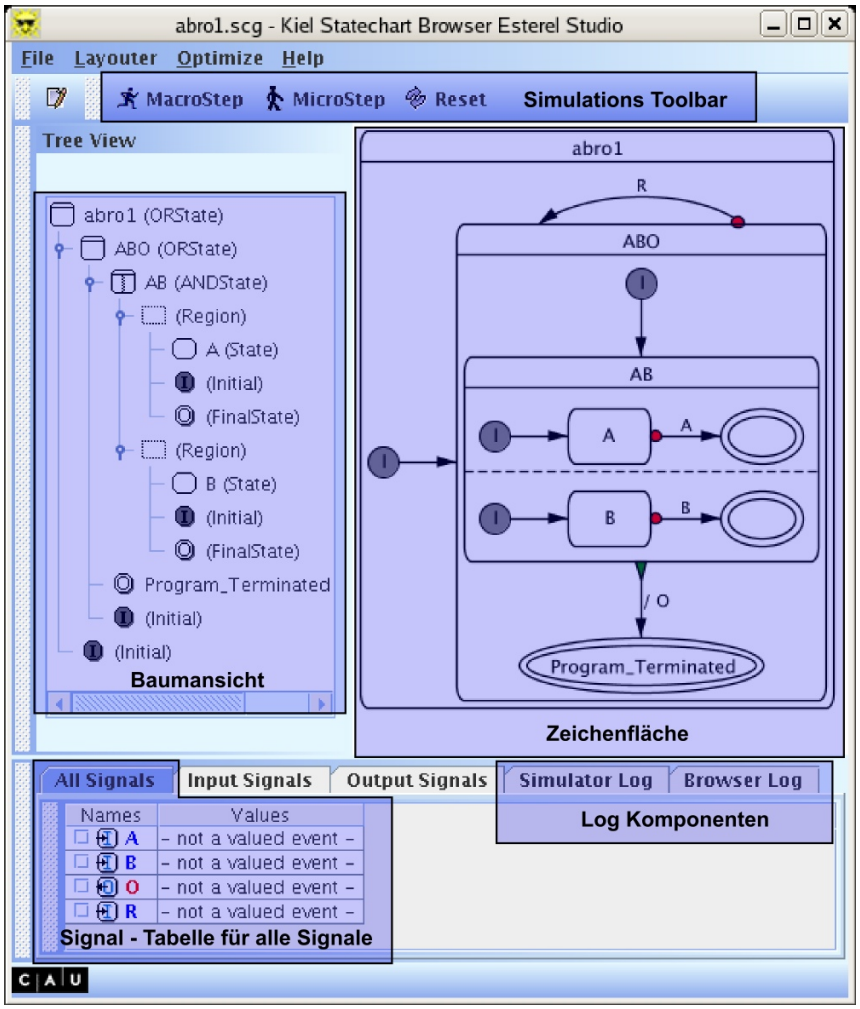


Abbildung A.1.: Oberfläche des Browsers

Statecharts laden: Es gibt zwei Möglichkeiten ein Statechart in den Browser zu übernehmen, um eine Simulation zu starten. Die erste Möglichkeit ist ein State-

chart im Editor zu laden oder zu zeichnen und auf das Browser *Icon* zu klicken. Die zweite Möglichkeit ist ein Statechart direkt über das File-Menü zu öffnen (siehe Abbildung A.2). Beiden Möglichkeiten ist gemeinsam, dass direkt nach dem Laden getestet wird, ob ein passender Simulator für das Statechart vorhanden ist und ob somit das Statechart simuliert werden kann. Ist dies nicht der Fall wird eine Meldung Auskunft darüber geben, warum das Statechart nicht für eine Simulation in Frage kommt. Diese Meldung wird von dem entsprechenden Simulator-Modul erzeugt und ist nicht Teil des Browsers.

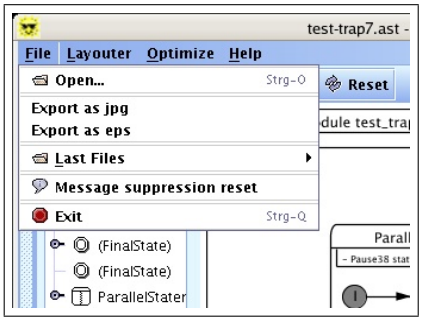


Abbildung A.2.: Laden eines Statecharts

Layouter auswählen: Ein Layouter wird über das Layouter-Menü ausgewählt, das kann jeder Zeit passieren (siehe Abbildung A.3). Nach Anwendung eines Layouters besteht die Möglichkeit wieder zum originalen Layout zurückzukehren. Dies geschieht über das Layouter-Menü und den Button OriginalLayout.

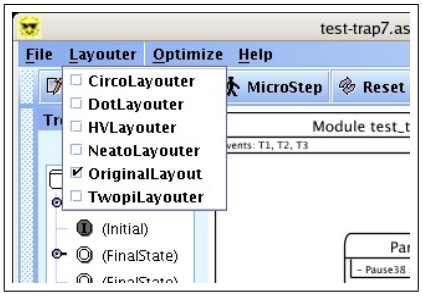


Abbildung A.3.: Verfügbare Layouter

Baumansicht benutzen: In der Baumansicht kann man sich die Struktur des Statecharts anschauen (siehe Abbildung A.4). Hierarchische Zustände können hier ein- und aufgeklappt werden. Sind in einem hierarchischem Zustand lokale Signale oder Variablen deklariert, befindet sich ein Declarations-Ordner in diesem hierarchischem Zustand, der die Namen der Signale bzw. Variablen enthält. Als *Tooltip* der einzelnen Zustände wird ihre *Id* aus der *KIEL* Datenstruktur angezeigt. Um herauszufinden welche Zustande in der Baumansicht welchen Zuständen in der Zeichenfläche entsprechen, genügt es den betreffenden Zustand

mit einem Klick zu markieren. Das funktioniert für alle Zustände, bis auf die Regionen und den Vaterzustand. Es ist auch möglich mehrere Zustände (mit Strg) oder einen Bereich (mit Shift+Endzustand) von Zuständen zu markieren. Die Markierungen lassen sich mit einem Klick auf den Vaterzustand oder eine Region wieder aufheben. Die Markierungen funktionieren auch während einer Simulation.

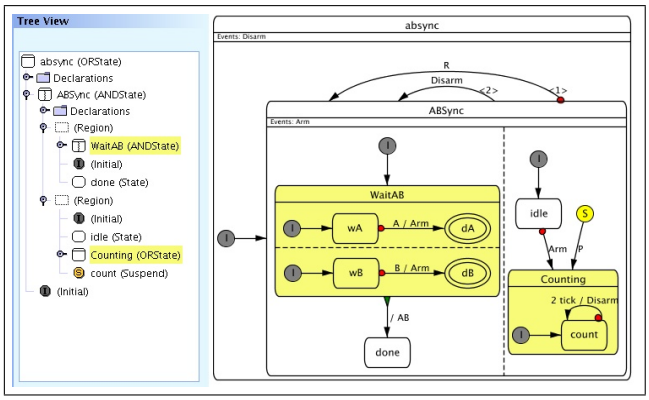


Abbildung A.4.: Auswählen von Zuständen in der Bausansicht

Zeichenfläche anschauen: Hier wird das aktuelle Statechart abgebildet (siehe Abbildung A.5). Während einer Simulation werden Teile des Statechart markiert. Welche Arten von Markierungen es gibt und wie man sie konfiguriert, wird in Abschnitt 3.4 näher beschrieben. Der *Tooltip*, der gesamten Zeichenfläche, zeigt die *Input*- und *Output*-Signale des Statecharts.

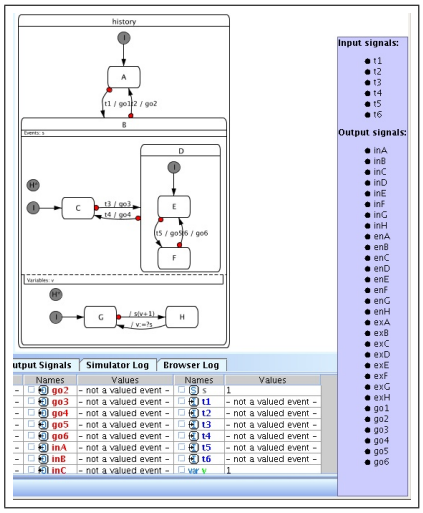


Abbildung A.5.: Zeichenfläche mit Tooltip

Zoom anwenden: Passt ein Statechart nicht komplett auf die Zeichenfläche, hat man die Möglichkeit, die Ansicht des Statecharts zu verkleinern oder zu ver-

größern (siehe Abbildung A.6). Dazu muss sich der Maus-*Cursor* auf der Zeichenfläche befinden. Mit Hilfe von Taste Strg und des Mausrades lässt sich nun das Statechart *zoomen*. Das Zurücksetzen der Simulation durch Drücken des Reset Knopfes bewirkt auch ein zurücksetzen des *Zoom* Faktors auf 100% .

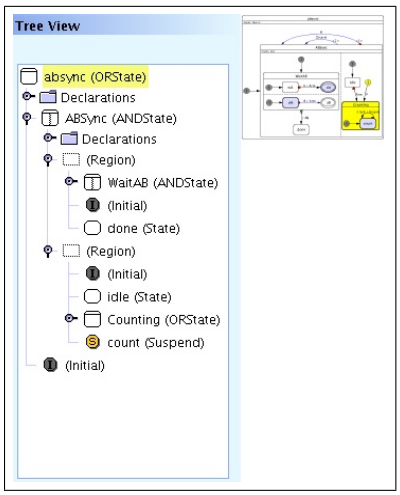


Abbildung A.6.: Ein verkleinertes Statechart während einer Simulation

Signaltabellen benutzen: In den Signaltabellen befinden sich Informationen über die Arten von Signalen bzw. Variablen. Die Tabellen sind nur aktiv, wenn die Simulation eines Statecharts möglich ist. Ist die Simulation nicht möglich und möchte man trotzdem etwas über die Signale erfahren, kann die Baumansicht bzw. der *Tooltip* der Zeichenfläche zur Hilfe genommen werden. Es gibt folgende Tabellen: Eine Tabelle mit allen *Input*-Signalen, eine Tabelle mit allen *Output*-Signalen und eine Tabelle mit allen Signalen (Input, Output, Lokal) und allen Variablen. Die Tabellen sind zweispaltig: In der ersten Spalte stehen die Namen und in der zweiten Spalte die aktuellen Werte (wenn es wertebehaftete Signale oder Variablen sind). Um den zur Verfügung stehenden Platz voll auszunutzen, werden die Tabellen eventuell auf mehrere Spalten verteilt. Um die verschiedenen Arten von Signalen bzw. Variablen zu unterscheiden, sind die Namen entsprechend ihres Typs markiert (siehe Abschnitt 3.4). Ein zusätzliches *Icon* hilft den Typ auf den ersten Blick zu unterscheiden (siehe Abbildung A.7).

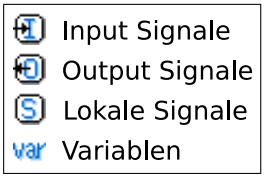


Abbildung A.7.: *Icons* für die Typen in Signaltabellen

Wird ein Signal während der Simulation gesetzt, bekommt es zu diesem Zeitpunkt einen Haken in der entsprechenden Farbe seines Typs (siehe Abbildung A.8). Vom Benutzer lassen sich nur Signale des Typs *Input* aktivieren. Der gewählte Zustand eines *Input*-Signals wird zum Beginn eines `MacroStep` übernommen und gilt während des gesamten Schrittes. Deaktiviert man dieses Signal also während eines `MicroSteps` wird diese Änderung erst im nächsten `MacroStep` berücksichtigt. Wertebehaftete Signale können mit Hilfe eines Doppelklicks in die entsprechende Wertezelle eines *Input*-Signals eingegeben werden. Der Wertebereich entspricht dem eines *Java-Integers*. Nachdem man einem Signal einen Wert zugewiesen hat, wird das Signal automatisch aktiviert.

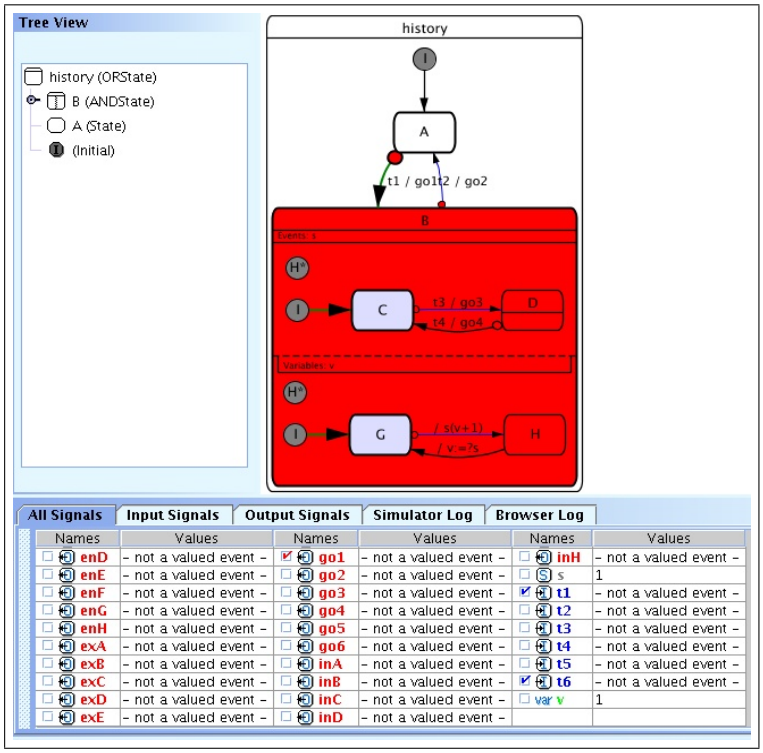


Abbildung A.8.: Eine Signaltabelle während einer Simulation

Logs anschauen: Es gibt zwei verschiede Log-Komponenten, die man sich im Browser anschauen kann, einmal das *Simulator-Log* und einmal das *Browser-Log*. Im *Simulator-Log* stehen die durchgeführten Berechnungen vor einem Simulationsschritt und die aktuellen Schritten während einer Simulation. Im *Browser-Log* stehen die Aktivitäten des Browsers (siehe Abschnitt 3.4). Über ein Kontext Menü, das mit Hilfe eines Rechtsklicks aufgerufen wird, ist es möglich, die Logs zu speichern.

Statecharts simulieren: Um eine Simulation durchzuführen, gibt es die drei *Buttons* MacroStep, MicroStep und Reset im Browser-*Toolbar*. Sind MacroStep und MicroStep ausgegraut, ist keine Simulation möglich. Ansonsten ist die Simu-

lation in zwei Granularitäten möglich (siehe Abschnitt 3.1). Eine MacroStep-Simulation führt eine Reihe von MicroSteps auf einmal aus (ob diese MicroSteps tatsächlich simultan oder schrittweise mit vorgegebenen Pausen angezeigt werden, lässt sich konfigurieren (siehe Abschnitt 3.4). Alternativ kann man direkt mit den elementaren MicroSteps starten. Während dieser Simulation wird der MacroStep Button zum continue... Button dieser lässt dann die Simulation bis zum nächsten MacroStep automatisch weiterlaufen, ohne jeden MicroStep einzeln zu bestätigen.

Konfigurationsdateien ändern Während das Programm läuft werden Änderungen in den Konfigurationsdateien übernommen wenn der Reset-Button ausgelöst wird. Mögliche Änderungen stehen in Abschnitt 3.4.

B. Java-Code für den Browser

B.1. Package `kiel.browser`

B.1.1. `Browser.java`

```
//$Id: Browser.java,v 1.131 2006/06/16 07:47:02 spr 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.Action;
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.util.LogFile;
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: 1.131 $ last modified $Date: 2006/06/16 07:47:02 $
 *          </p>
 *
 * <p>
 * $Log: Browser.java,v $
 * 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.STRUCTURE,
                                        "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());
          }
          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("" + 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(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("" + 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(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(
1880    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();
      }
1890  }

    public EsterelTabPopupMenu() {
      JMenuItem mi;

      mi = new JMenuItem("Transform to SSM");
      mi.addActionListener(
        new AbstractAction("Transform to SSM") {
        public void actionPerformed(final ActionEvent e) {
          try {
1900        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) {
1910          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 {
1920        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() {
1930            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,
1940          "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);
            }
          }
        }
1950    });
        add(mi);

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

          public void actionPerformed(final ActionEvent e) {
            isTimeTriggert = !isTimeTriggert;
1960      }
        });
        timetriggert.setState(false);
        add(timetriggert);

      }
    }


    class PopupListener extends MouseAdapter {
      public void mousePressed(MouseEvent e) {
1970    maybeShowPopup(e);
      }

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

      private void maybeShowPopup(MouseEvent e) {
        if (e.isPopupTrigger()) {
1980      popup.show(e.getComponent(), e.getX(), e.getY());
        }
      }
    }

}
```

B.1.2. BrowserProperties.java

```
// $Id: BrowserProperties.java,v 1.23 2006/05/21 20:24:01 miwi Exp $
package kiel.browser;

import java.awt.Color;

import javax.swing.KeyStroke;

import kiel.browser.model.BrowserModel;
import kiel.util.LogFile;
import kiel.util.preferences.Preferences;

/**
 * <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: 1.23 $ last modified $Date: 2006/05/21 20:24:01 $
 *
 */
public final class BrowserProperties {

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

    /**
     * The property class containing the values. Being moved to a wrapper for
     * kiel.util.preferences.Preferences
     */
    private static Preferences prefs;

    /**
     * Creates Preferences object and reloads settings from file/resource.
     */
    static {
        prefs = Preferences.createInstance("browser", Browser.class
                .getResource("browser.properties"));
        reload();
    }

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

    /**
     * reloads the property file. Files in home directory are preferred.
     */
    public static void reload() {
        prefs.reload();
    }

    /**
     *
     * @param key
     *            String
     * @return KeyStroke
     */
    public static KeyStroke getKeyStroke(final String key) {
        // System.out.println("Requesting: " + key + " = " + getProperty(key) +
        // " "
        // + KeyStroke.getKeyStroke(getProperty(key).replace('+',' ')));
//      if (System.getProperty("os.name").equals("Mac OS X")) {
//              return KeyStroke.getKeyStroke(prefs.getString(key).replace('+', '
  ').replaceAll("ctrl", "meta"));
//      }
        return KeyStroke.getKeyStroke(prefs.getString(key).replace('+', ' '));
    }

    /**
     * @return boolean true if want to do Animations
     */
    public static boolean doAnimations() {
        return prefs.getString("Browser.Animation").equals("true");
    }

    /**
     * @return value of animation start (in seconds)
     *
     */
    public static int getAnimationDuration() {
        final int minDur = 100;
        int dur = prefs.getInt("Animation.Duration");
        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 = prefs.getInt("MicroStep.SleepTime");
        if (stime < minSleepTime) {
            stime = minSleepTime;
        }
        return stime;
```

```
    }

    /**
     * @return String with name for browser tools
     */
    public static String getBrowserTitle() {
        return prefs.getString("Browser.Title");
    }

    /**
     * @return Color to mark nodes in
     */
    public static Color getNodeMarkColor() {
        return prefs.getHexColor("BrowserTree.MarkNodeColor");
    }

    /**
     * @return Color to mark nodes in
     */
    public static Color getEdgeMarkColor() {
        return prefs.getHexColor("BrowserTree.MarkEdgeColor");
    }

    /**
     * @return Color to mark unparsed Labels in estudio sim mode.
     */
    public static Color getStringLabelColor() {
        return prefs.getHexColor("BrowserCanvas.StringLabelColor");
    }

    /**
     * @return Color to show
     */
    public static Color getInputEventColor() {
        return prefs.getHexColor("BrowserTables.InputEventColor");
    }

    /**
     * @return Color to show
     */
    public static Color getOutputEventColor() {
        return prefs.getHexColor("BrowserTables.OutputEventColor");
    }

    /**
     * @return Color to show
     */
    public static Color getLocalEventColor() {
        return prefs.getHexColor("BrowserTables.LocalEventColor");
    }

    /**
     * @return Color to show
     */
    public static Color getVariablesColor() {
        return prefs.getHexColor("BrowserTables.VariablesColor");
    }

    /**
     * @return String
     */
    public static String getExecuteTransition() {
        return prefs.getString("MicroStep.ExecuteTransition");
    }

    /**
     * @return String
     */
    public static String getTestTransition() {
        return prefs.getString("MicroStep.TestTransition");
    }

    /**
     * @return String
     */
    public static String getOnInside() {
        return prefs.getString("MicroStep.OnInside");
    }

    /**
     * @return String
     */
    public static String getOnExit() {
        return prefs.getString("MicroStep.OnExit");
    }

    /**
     * @return String
     */
    public static String getOnEntry() {
        return prefs.getString("MicroStep.OnEntry");
    }

    /**
     * @return String
     */
    public static String getStateSuspended() {
        return prefs.getString("MicroStep.StateSuspended");
    }

    /**
     * @return String
     */
    public static String getStateActivated() {
        return prefs.getString("MicroStep.StateActivated");
    }

    /**
     * @return String
     */
    public static String getStateDeactivated() {
        return prefs.getString("MicroStep.StateDeactivated");
    }

    /**
     * @return String
     */
    public static String getActualConfig() {
        return prefs.getString("MicroStep.ActualConfigState");
    }
```

```
    /**
     * 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 prefs.getString("Browser.AlwaysStepThrough").equals("true")
                && stepThroughOverWriter;
    }

    /**
     * @return boolean true if the interface should be shown
     */
    public static boolean doShowInterface() {
        return prefs.getString("Browser.ShowInterface").equals("true");
    }

    /**
     * @return boolean true if always a priority should be shown.
     */
    public static boolean doAlwaysShowPriorities() {
      return prefs.getString("Browser.AlwaysShowPrios").equals("true");
    }

    /**
     * @return boolean true if the statechart tooltips should be shown
     */
    public static boolean doShowTooltip() {
        return prefs.getString("Browser.ShowTooltip").equals("true");
    }

    /**
     * @return boolean true if the interface should be shown
     */
    public static boolean animateConfigsInMicroSteps() {
        return prefs.getString("Browser.AnimateConfigInMicroSteps").equals(
                "true");
    }

    /**
     * @return int with logLevel
     */
    public static int getLogLevel() {
        return prefs.getInt("Browser.LogLevel");
    }

    /**
     * @return String
     */
    public static String getLogLevelCompareOp() {
        return prefs.getString("Browser.LogCompare");
    }

    /**
     * @return int Font size for internal editor.
     */
    public static int getKitEditorFontSize() {
        return prefs.getInt("KitEditor.FontSize");
    }

    /**
     * @return int threshold for parser updates.
     */
    public static int getUpdateThreshold() {
        return prefs.getInt("ParserThread.Threshold");
    }

    /**
     * @return char
     */
    public static char getTraceFastForwardMnemonic() {
        return prefs.getChar("Browser.Mnemonic.TraceFastForward");
    }

    /**
     * @return char
     */
    public static char getTraceRewindMnemonic() {
        return prefs.getChar("Browser.Mnemonic.TraceRewind");
    }

    /**
     * @return char
     */
    public static char getTraceStopMnemonic() {
        return prefs.getChar("Browser.Mnemonic.TraceStop");
    }

    /**
     * @return char
     */
    public static char getTracePlayMnemonic() {
        return prefs.getChar("Browser.Mnemonic.TracePlay");
    }

    /**
     * @return char
     */
    public static char getTraceStepForwardMnemonic() {
        return prefs.getChar("Browser.Mnemonic.TraceStepForward");
    }

    /**
     * @return char
     */
    public static char getTraceStepBackMnemonic() {
```

```
        return prefs.getChar("Browser.Mnemonic.TraceStepBack");
    }

    /**
     * @return char
     */
    public static char getStateMenuMnemonic() {
        return prefs.getChar("Browser.Mnemonic.State");
    }

    /**
     * @return char
     */
    public static char getTransitionMenuMnemonic() {
        return prefs.getChar("Browser.Mnemonic.Transition");
    }

    /**
     * @return char
     */
    public static char getViewMenuMnemonic() {
        return prefs.getChar("Browser.Mnemonic.View");
    }

    /**
     * @return char
     */
    public static char getStatechartMenuMnemonic() {
        return prefs.getChar("Browser.Mnemonic.Statechart");
    }

    /**
     * @return char
     */
    public static char getEditModeMenuMnemonic() {
        return prefs.getChar("Browser.Mnemonic.EditMode");
    }

    /**
     * @return char
     */
    public static char getMacroStepMnemonic() {
        return prefs.getChar("Browser.Mnemonic.MacroStep");
    }

    /**
     * @return char
     */
    public static char getMicroStepMnemonic() {
        return prefs.getChar("Browser.Mnemonic.MicroStep");
    }

    /**
     * @return char
     */
    public static char getResetMnemonic() {
        return prefs.getChar("Browser.Mnemonic.Reset");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getCountElementsAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.CountElements");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getAlwaysFitScreenAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.AlwaysFitScreen");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getFitScreenAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.FitScreen");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getResetZoomAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.ResetZoom");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getNewStatechartAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.NewStatechart");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getSaveKitAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.SaveKit");
    }

    /**
     * @return KeyStroke
     */
    public static KeyStroke getLoadTraceAccelerator() {
        return prefs.getKeyStroke("Browser.KeyStroke.LoadTrace");
    }

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

    }
```

```
  /**
   * @return String name of layouter benchmark file.
   */
  public static String getLayouterBenchName() {
    return prefs.getString("Browser.LayouterBenchFile");
  }

  /**
   * @return String name of drawing benchmark file.
   */
  public static String getDrawingBenchName() {
    return prefs.getString("Browser.DrawingBenchFile");
  }

  /**
   * @return String name of simulator benchmark file.
   */
  public static String getSimBenchName() {
    return prefs.getString("Browser.SimulatorBenchFile");
  }

  /**
   * @return boolean if should create benchmark file.
   */
  public static boolean benchmarkLayouter() {
    return prefs.getString("Browser.doLayouterBench").equals("true");
  }
  /**
   * @return boolean if should create benchmark file.
   */
  public static boolean benchmarkDrawing() {
    return prefs.getString("Browser.doDrawingBench").equals("true");
  }
  /**
   * @return boolean if should create benchmark file.
   */
  public static boolean benchmarkSim() {
    return prefs.getString("Browser.doSimulatorBench").equals("true");
  }
  /**
   * @return boolean if should create simout file.
   */
  public static boolean writeSimOutput() {
    return prefs.getString("Browser.doSimulatorOutput").equals("true");
  }

}
```

B.1.3. BrowserException.java

```
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);
  }
}
```

B.2. Package kiel.browser.model

B.2.1. BrowserModel.java

```
// $Id: BrowserModel.java,v 1.101 2006/05/22 17:14:32 miwi 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 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.util.LogFile;
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: 1.101 $ last modified $Date: 2006/05/22 17:14:32 $</p>
 *
 * <p>
 * $Log: BrowserModel.java,v $
 * 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
110  *
     * 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
     *
120  * 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>
     *
130  * 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
140  * 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>
150  *
     * 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>
160  *
     * 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>
     *
170  * 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
180  * 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>
190  *
     * 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>
     *
200  * 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
210  * 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>
220  *
     * 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>
     *
230  * 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
240  * 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>
250  *
     * 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>
     *
260  * 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
270  * 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>
280  *
     * 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>
     *
290  * 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
300  * 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
310  * 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>
320  *
     * 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>
     *
330  * 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
340  * 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 {
  /**
   *  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");
    browserLog.log(BrowserProperties.getLogLevel(), "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");
  }

  /**
   *
   * @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(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();
      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;
      }
      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)
        || (objectsChanged != null && objectsChanged.size() > 0)) {
      if (objectsRemoved == null) {
        objectsRemoved = new ArrayList();
      }
      if (objectsAdded == null) {
        objectsAdded = new ArrayList();
      }
      if (objectsChanged == null) {
        objectsChanged = new ArrayList();
      }
//      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");
//      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()) {
//        GraphicalObject obj = (GraphicalObject) iter.next();
//        System.out.println("" + i + ". " + obj + " id: " + obj.getID());
//        i++;
//      }

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

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

      setMode(EDIT);

      setChanged();
      notifyObservers(TABLE_UPDATE);
      setChanged();
      notifyObservers(SIMRESET);
```

```
  }
}

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

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

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

/**
 *
 * @param td TraceData
 */
public final void setTrace(final TraceData td) {
  trace = td;
  trace.play();
  trace.startTrace();
  BrowserProperties.reload();
  resetSimulator(false);
  showStaticView();
  setChanged();
  notifyObservers(REPAINT_TRACEINFO);
}

/**
 * @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 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);
}

/**
 * @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));
    }
  } else if (BrowserProperties.benchmarkLayouter()) {
    layouterBench.start();
    layouterBench.stop();
  }
  setChanged();
  notifyObservers(NEW_CONFIGURATION);
}

/**
 * @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);
    simLog.log(LogFile.INFO, "========== Simulator reset starting ==========");
    if (sim != null && !withoutSimReset) {
      sim.reset();
    }
    clearAllTables();
    setChanged();
    notifyObservers(TABLE_REFRESH);

    step = null;
    microStepNr = -1;
    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 ==========");
    setChanged();
    notifyObservers(COMPUTATION_COMPLETED);
  }

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


  /**
   * Clears all emits and values settings.
   */
  private void clearAllTables() {
    inputSignalTable.clearExp();
    outputSignalTable.clearExp();
    localSignals.clearExp();
    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) {
    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;
  }
}
```

B.2.2. ExpInformation.java

```
// $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());
      }
    }
  }
```

B.2.3. ModelMessage.java

```
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;
  }

}
```

B.2.4. SignalTableModel.java

```
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 {
          if (!info.isUndefined()) {
            return "" + info.getStringValue();
          } else {
            return " - ";
          }
        } 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));
  }
}
```

B.3. Package kiel.browser.view

B.3.1. BrowserGUI.java

```
// $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;
  }

}
```

B.3.2. BrowserCanvas.java

```
// $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();
  }
}
```

B.3.3. BrowserMenuBar.java

```
// $Id: BrowserMenuBar.java,v 1.26 2006/01/16 18:00:16 miwi 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.util.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: 1.26 $ last modified $Date: 2006/01/16 18:00:16 $
 *
 * $Log: BrowserMenuBar.java,v $
 * 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);
              }
            }
                }
           */
        }
      }

    }
```

B.3.4. BrowserToolbar.java

```
// $Id: BrowserToolbar.java,v 1.26 2006/05/22 17:14:32 miwi 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.util.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: 1.26 $ last modified $Date: 2006/05/22 17:14:32 $
 */
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 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());
      }
    });

    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();
    }
  }
}
```

B.3.5. BrowserTree.java

```
// $Id: BrowserTree.java,v 1.28 2006/05/21 20:24:02 miwi 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.util.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: 1.28 $ last modified $Date: 2006/05/21 20:24:02 $
 *
 */
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: 1.28 $ last modified $Date: 2006/05/21 20:24:02 $
     *
     */
    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;
  }
```

B.3.6. GraphicalObjectsLibrary.java

```
// $Id: GraphicalObjectsLibrary.java,v 1.25 2006/02/08 08:32:40 miwi 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.util.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: 1.25 $ last modified $Date: 2006/02/08 08:32:40 $
 *
 */
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 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 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.
120  */
    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";
130 /**
     * Properties identifier for the state.
     */
    public static final String FINALCOMPOSITESTATE = "SVG.FinalComposite";
    /**
     * Cached andstate template.
     */
    private GroupNode andstate = null;
    /**
     * Cached orstate template.
140  */
    private GroupNode orstate = null;
    /**
     * Cached simplestate template.
     */
    private GroupNode simplestate = null;
    /**
     * Cached initialstate template.
     */
    private GroupNode initialstate = null;
150 /**
     * Cached suspendstate template.
     */
    private GroupNode suspendstate = null;
    /**
     * Cached historystate template.
     */
    private GroupNode histroystate = null;
    /**
     * Cached choicestate template.
160  */
    private GroupNode choicestate = null;
    /**
     * Cached junctionstate template.
     */
    private GroupNode junctionstate = null;
    /**
     * Cached finalstate template.
     */
    private GroupNode finalstate = null;
170 /**
     * Cached final compisite state template.
     */
    private GroupNode finalcompstate = null;
    /**
     * Cached deep history state template.
     */
    private GroupNode deephistorystate = null;

    /**
180  * 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.
     */
190 //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";

200 /**
     * 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.
210  */
    private Properties init;
    /**
     * actual loaded ini file.
     */
    private String initfile;
    /**
       /**
        * is the library valid (all fields known???).
        */
220  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/";
    /**
230  *  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 {
      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;
    }

    /**
     * @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;
    }
  }
```

B.3.7. ResourceLoader.java

```
// $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;
    }
  }
}
```

B.3.8. SignalTable.java

```
// $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("Declarated 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("Declarated 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"));
```

B.4. Package kiel.browser.controller

B.4.1. MenuController.java

```
// $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");
//      }
//    }
  }
}
```

B.4.2. SimulationController.java

B.4.3. LayoutController.java

```
package kiel.browser.controller;

import java.awt.event.ActionEvent;

import javax.swing.AbstractAction;

import kiel.browser.Browser;
import kiel.util.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: 1.6 $ last modified $Date: 2006/02/08 07:07:48 $
 */
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);
      }
    }

  }
```