

CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL

Diploma Thesis

Extension of *KIEL* by Stateflow Charts

cand. inform. Adrian Posor

December 16, 2005

Institute of Computer Science and Applied Mathematics
Real-Time and Embedded Systems Group

Prof. Dr. Reinhard von Hanxleden

Advised by:
Steffen H. Prochnow

Eidesstattliche Erklärung

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

Kiel,

Abstract

The *Kiel Integrated Environment for Layout (KIEL)* application is a tool to create and simulate state diagrams that are dialects of Harel's Statecharts. Special features of *KIEL* are its set of automatic layouters and the simulation environment which uses the concept of dynamic charts. This work extends *KIEL* by the ability to import, export, and simulate *Stateflow* charts. The data structure used by *KIEL* to store state diagrams is described in detail followed by an introduction to *Stateflow* and its API. A comparison of the *KIEL* data structure to the *Stateflow* API objects results in an appropriate mapping between them. This mapping is needed to import and export *Stateflow* charts using the API. The simulation of *Stateflow* charts within *KIEL* is realized with the help of *Stateflow*, which performs all necessary calculations.

Keywords *Statechart*, import, export, state diagram, simulation, *Stateflow*, *Matlab*, *KIEL*

Contents

1. Introduction	15
1.1. The <i>KIEL</i> Project	15
1.2. The task within the scope of the <i>KIEL</i> Project	15
1.3. Overview	16
2. Theoretical Basis	17
2.1. Automata	17
2.2. State Diagram	17
2.2.1. The <i>KIEL</i> Data Structure	19
2.2.2. Stateflow Objects	22
2.3. Matlab/Stateflow-API	31
2.4. Comparison of <i>KIEL</i> Data Structure classes with <i>Stateflow</i> API Objects	33
3. Stateflow Extensions of <i>KIEL</i>	35
3.1. Requirements	35
3.2. How to simulate a <i>Stateflow</i> Chart	36
3.3. The input trace Feature	36
3.4. Import and Export of <i>Stateflow</i> Charts	37
3.5. <i>Stateflow</i> Mode of the Editor	37
3.6. Implementation	37
3.6.1. Surroundings	37
3.6.2. Communication with <i>Matlab</i>	39
3.6.3. Finding <i>Stateflow</i> Objects using the API	42
3.6.4. Setting the Simulation Parameters using the API	43
3.6.5. Manipulating the <i>Simulink</i> Model using the API	43
3.6.6. Import of <i>Stateflow</i> Charts	44
3.6.7. Export of <i>Stateflow</i> Charts	47
3.6.8. Trace of input data	48
3.6.9. Simulation of <i>Stateflow</i> Charts	48
3.6.10. Adaption of the Editor	51
4. Concluding Remarks	55
4.1. Results	55
4.2. Future Prospects	55
5. Bibliography	61

Contents

A. Settings	63
B. Java Code	65
B.1. Description	65
B.1.1. Package kiel.fileInterface.matlab	65
B.1.2. Package kiel.fileInterface.M	65
B.1.3. Package kiel.fileInterface.ESI	65
B.1.4. Package kiel.util	66
B.1.5. Package kiel.simulator.matlab	66
B.2. kiel.fileInterface.matlab	67
B.2.1. MatlabStateflowFileFilter	67
B.2.2. MatlabStateflow	68
B.3. kiel.fileInterface.M	71
B.3.1. MFileFilter	71
B.3.2. MFormat	72
B.4. kiel.fileInterface.ESI	76
B.4.1. ESIFileFilter	76
B.4.2. ESIFormat	77
B.5. kiel.util	80
B.5.1. MatlabStateflowGrabber	80
B.5.2. MatlabStateflowGrabberException	96
B.5.3. MatlabStateflowCreator	97
B.5.4. MatlabStateflowCreatorException	109
B.5.5. MatlabProcessInfo	110
B.5.6. MatlabStateflowProperties	112
B.6. kiel.simulator.matlab	113
B.6.1. StateflowSimulator	113
B.6.2. SimulatorException	123
B.7. kiel.editor.controller	124
B.7.1. EditorModeAddJunction	124
B.7.2. ActionEditStateChartEventsForStateflow	125
B.7.3. ActionEditStateChartVariablesForStateflow	127
B.7.4. ActionEditStateEventsForStateflow	131
B.7.5. ActionEditStateVariablesForStateflow	133
B.7.6. ActionEditStateEntryActionsForStateflow	136
B.7.7. ActionEditStateDoActionsForStateflow	137
B.7.8. ActionEditStateExitActionsForStateflow	138
B.7.9. ActionEditStateBindActionsForStateflow	139
B.8. kiel.editor.graph	140
B.8.1. JunctionRenderer	140

List of Tables

2.1. Notation of <i>Stateflow</i> Objects	30
2.2. Comparison of <i>KIEL</i> data structure classes with Stateflow API objects	34
4.1. Performance test results	58

List of Tables

List of Figures

2.1.	The <i>KIEL</i> data structure class diagram	23
2.2.	PseudoState and its subclasses	24
2.3.	State and its subclasses	25
2.4.	Transition and its subclasses	26
2.5.	A Stateflow window showing a diagram	27
2.6.	Stateflow object hierarchy (Source: Stateflow User's Guide [7])	31
2.7.	Stateflow API object hierarchy (Source: Stateflow API reference [6]) .	32
3.1.	The simulation toolbar of the <i>KIEL</i> browser	36
3.2.	The trace toolbar of the <i>KIEL</i> browser	36
3.3.	The <i>Esterel Studio</i> dependent part of the editor's toolbar	37
3.4.	The <i>Stateflow</i> dependent part of the editor's toolbar	37
3.5.	The editor's dialog for editing chart events	38
3.6.	The editor's dialog for editing state variables	38
3.7.	Components of the <i>KIEL</i> application	40
3.8.	Algorithm in pseudocode for import of <i>Stateflow</i> charts	46
3.9.	Algorithm in pseudocode for export of <i>Stateflow</i> charts	49
3.10.	The interface MicroStep and the classes that implement it	52
4.1.	The bintree3 Chart	56
4.2.	The bintree5 Chart	56
4.3.	The traffic light chart	57

List of Figures

Listing of the abbreviations

API	Application Programming Interface
DFA	Deterministic Finite Automaton
GUI	Graphical User Interface
KIEL	Kiel Integrated Environment for Layout
UML	Unified Modelling Language

List of Figures

1. Introduction

The majority of computers in use are embedded into electronic devices that they control. They are called *embedded systems*. The development of software for these *embedded systems* is often not realized using typical programming languages like *C* or *Java*. Instead of this, their behavior is described using a more abstract way: a system model. The system model is used to generate program code. This way, the model does not only serve as a specification language, but also as a graphical programming language. Several modeling tools exist on the market that implement a dialect of the so called *Statecharts*.

1.1. The *KIEL* Project

The *Kiel Integrated Environment for Layout* program [11] is a modeling tool for *Statecharts* and has been developed at the chair of *Embedded and Real-Time Systems*. Its main goal is to provide automatic layout capabilities. Several layouters are integrated into this modeling tool. With their help the developer can focus on the structure of the chart and leave the layout in the hands of the computer.

The tool provides a new approach of showing simulation results. States that are not active are collapsed during the simulation to hide the details of the chart that are not important in the current simulation step. At each simulation step a new view of the chart is shown with the currently active states expanded and the non-active states collapsed. This dynamic view helps to overlook the chart during the simulation.

1.2. The task within the scope of the *KIEL* Project

The given task for this diploma thesis is to expand the *KIEL* application by the ability to load and to simulate charts that have been created with *Matlab/Simulink/Stateflow*. Due to the fact that *Matlab/Simulink/Stateflow* has got an API the solution can be reduced to the communication with *Matlab/Simulink/Stateflow*, which performs all the real work for *KIEL*. The requirements are described in detail in Section 3.1.

1.3. Overview

The following chapter (Chapter 2) provides the theoretical background. It deals with deterministic finite automata, state diagrams in general, the internal data structure that is used by *KIEL* to store the chart in memory (Section 2.2.1), and the *Stateflow* [7] modeling tool by *The Mathworks* [13] (Section 2.2.2). The next chapter (Chapter 3) provides detailed information on the new features that have been added to the *KIEL* application as well as the implementation of these features (Section 3.6).

2. Theoretical Basis

This chapter provides the theoretical background behind the system models this document is about. It describes the internal data structure of the *KIEL* application and introduces the modeling tool *Stateflow* along with its API.

The models used in software engineering to model the behavior of a system are based on finite state machines. A state diagram is a graphical representation of a finite state machine. There are many forms of state diagrams that differ slightly in outlook and semantics. One of the most popular form of state diagram is the so-called *Statechart*, which has been developed in 1987 by David Harel [2]. A variant of *Statecharts* has become a part of UML [8].

2.1. Automata

An automaton is a mathematical model for a finite state machine. Formally, a deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, S_0, F)$, where

- Q denotes a finite set of states.
- Σ denotes a finite set of symbols, called the *alphabet*.
- δ denotes the transition function $\delta : Q \times \Sigma \rightarrow Q$.
- $S_0 \in Q$ denotes the start state.
- $F \subset Q$ denotes a set of final states.

Let $\hat{\sigma} : Q \times \Sigma^* \rightarrow Q$ be the extended transition function which takes a string of symbols as an argument and returns the state in which the automaton will stay after processing the input. The language L accepted by the deterministic finite automaton is

$$L = \{w \in \Sigma^* | \hat{\sigma}(S_0, w) \in F\}$$

2.2. State Diagram

State diagrams are used to represent graphically finite state machines. State transition tables are another possible representation. A directed graph is a classic form of a state diagram. Each edge in the graph represents a transition between two states and each vertex represents a state. Extensions to this state diagram type,

2. Theoretical Basis

and therefore to the state machine itself, do exist. They extend the diagram /the machine by the concept of hierarchy, parallelism, events, actions and others.

A state diagram consists of three basic types of elements: states, pseudostates, and transitions.

State: A state represents a mode of operation. An invariant condition holds as long as the machine resides in a particular state. If the machine is in a particular state, this state is called active, otherwise it is called inactive. If a state contains states, it is called a composite state, otherwise it is called a simple state. A state consists of one or more so-called regions. The states in a region are mutually exclusive while the regions themselves are executed in parallel.

Pseudostate: A pseudostate is a kind of state the machine never resides in. Such a state is neither active nor inactive. The following types of pseudostates generally exist: *initial*, *deepHistory*, *shallowHistory*, *join*, *fork*, *junction*, *choice*, and *terminate*.

- An *initial* pseudostate is the source for a transition to the default state of a composite state. There cannot be more than one initial state in a region.
- A *shallowHistory* pseudostate stores the activity history within a composite state. If the associated composite state is reentered, then the previously active state within this composite state will be activated again. The shallowHistory object only stores the activity history of the level of containment it is placed in.
- A *deepHistory* pseudostate basically works like a shallowHistory object but it also stores the activity history of all states which are children of the states in the level of containment this object is placed in.
- A *join* pseudostate takes two or more concurrent transitions (transitions that originate from concurrent states) as its input and one transition as its output.
- A *fork* pseudostate takes one transition as its input and two or more transitions as its output. It splits a transition into several ones. The execution of the outgoing transitions runs concurrently.
- A *choice* pseudostate, which is used with transitions, provides a decision point in the flow of control. In the UML, it realizes a dynamic conditional branch. The transition actions of the transitions that connect to a choice object are executed directly while the transition segment is taken and before the entire transition is taken. This way the transition actions effect the path which is taken.
- A *junction* pseudostate, like a choice object, provides a decision point in the flow of control. In the UML, it realizes a static conditional branch.

- A *SynchState* is used in conjunction with forks and joins to ensure that one region leaves a particular state or states before another region can enter a particular state or states.
- If a *terminate* pseudostate is entered, then the state machine terminates.

Transition: A transition connects states and pseudo states. Transitions specify which states will become active in the next execution step of the machine. A compound transition is an acyclical chain of transitions joined by pseudostates (except for initial states). If a compound transition is taken, the source state becomes inactive and the destination state becomes active. A transition that connects two states is a special case of a compound transition. Associated with a transition is a trigger, a guard, and an effect.

In Safe State Machines, the special transition types *strong abortion*, *weak abortion*, and *normal termination* exist.

- A *strong abortion* transition, if it is taken, aborts any other ongoing activity. It aborts a state process without letting it perform its reaction in the current instance.
- A *weak abortion* transition lets the aborted state execute its current reaction for the last time, then prevents it from reacting to future instances.
- If all states contained in a composite state have reached a final state then this composite state can be leaved through a *normal termination* transition without the presence of any events.

2.2.1. The *KIEL* Data Structure

This section describes the internal data structure of the *KIEL* application. The data structure is used to represent a chart in memory.

The data structure is designed to support all variants of state diagrams that are similar to those defined in the UML standard. It has been developed using the principles of object oriented programming. The graphical information of a chart is strictly separated from its structure. This is the key to the ability of providing several layouts of the same chart. The following text describes all classes the data structure consists of. The semantics of the elements are described in Section 2.2.

Some classes do just exist to satisfy the needs of *Esterel Studio* [1] charts. On the other hand, some modifications had to be made to support *Stateflow* [7] charts. Figure 2.1 shows a class diagram which gives an overview of the data structure. Figures 2.2, 2.3, and 2.4 provide further details not shown in Figure 2.1. A complete documentation of all fields and methods of these classes can be found in the JavaDoc documentation of the *KIEL* project [12].

StateChart: An instance of this class holds the complete chart. It stores information about the type of chart, the input and output events as well as the input

2. Theoretical Basis

and output variables. The field `rootNode` contains the state that contains all other states of the chart.

State: An instance of this class represents a state. It is the base class for all kinds of states and usually not used directly. It stores the actions and variables which are associated with a state. Its direct subclasses are: `SimpleState`, `CompositeState`, and `FinalState`.

SimpleState: This class does not add any functionality to its base class. It just exists to have a type for a simple state, so that the base class does not need to be used directly.

FinalState: An instance of this class represents the so called final state. If the execution of the diagram reaches this state, then the chart will terminate.

CompositeState: This class is the base class for all states that can contain other states. Direct subclasses are `ANDState`, `ORState`, and `Region`.

ANDState: An instance of this class represents a state which can contain other states that run concurrently. The states contained in such a state are of type `Region`.

ORState: An instance of this class represents a state that can contain other states that do not run concurrently. Only one state included in an `ORState` can be active at an instance of time.

Region: An instance of this class represents a state that is contained in an AND-State and runs concurrently with the other states that are at the same level of hierarchy.

PseudoState: This class is the base class for all pseudo states. Direct subclasses are: `Choice`, `DeepHistory`, `DynamicChoice`, `ForkConnector`, `History`, `InitialState`, `Join`, `Junction`, `Suspend`, and `SynchState`.

Choice: An instance of this class represents a choice object. In the UML, it realizes a dynamic conditional branch, in *Esterel Studio*, it realizes a static conditional branch.

DynamicChoice: An instance of this class represents a dynamic choice object. It realizes a dynamic conditional branch.

History: An instance of this class represents a (shallow) history object.

DeepHistory: An instance of this class represents a deep history object.

InitialState: An instance of this class represents an initial state object. Such an object only appears together with an initial arc. The initial arc points to the state within an `ORState` which is set to be active when the `ORState` is entered.

Join: An instance of this class represents a join object.

Junction: An instance of this class represents a junction object.

SuspendState: An instance of this class represents a suspend state. It is always the destination of a suspension transition. If such a state is active, then the chart is suspended.

SynchState: An instance of this class represents a SynchState object. Such an object is a pseudo state.

ForkConnector: An instance of this class represents a fork connector object.

Transition: This class is the base class for all types of transitions. It contains a Label and a Priority. Direct subclasses are: ConditionalTransition, InitialArc, NormalTermination, StrongAbortion, Suspension, and WeakAbortion.

ConditionalTransition: An instance of this class represents a transition that is associated with a condition that has to be satisfied to take this transition. This class does not add any functionality to its base class. It just exists to have a type for conditional transitions.

InitialArc: An instance of this class represents an initial arc object. Such an object is a special kind of transition. Its source is always an initial state. Its destination is the state that will be activated when its parent, an ORState, is entered.

NormalTermination: An instance of this class represents a special kind of transition that is called Normal-Termination.

StrongAbortion: An instance of this class represents a special kind of transition that is called Strong-Abortion.

WeakAbortion: An instance of this class represents a Weak-Abortion transition.

Suspension: An instance of this class represents a suspension transition. The destination of such a transition is always a suspend state. If the events associated with a suspension transition are present, then the transition will be taken and the chart will be suspended.

Edge: An instance of this class represents an edge. An edge always links two nodes. It stores a source and a destination. Direct subclass is: Transition.

Node: An instance of this class represents a node. It stores its parent, all incoming and outgoing edges/transitions and its name. Direct subclasses are: PseudoState, and State.

2. Theoretical Basis

GraphicalObject: All objects that have a graphical representation are derived from this class. This class is not used directly. It contains an unique identifier. Direct subclasses are: `DelimiterLine`, `Edge`, `Node`, `Priority`, and `TransitionLabel`.

DelimiterLine: An instance of this class represents a line between two regions. It does not contain any graphical information like the position of the line.

Priority: Every transition has a priority that is represented by this class. It contains a value that is the value of the priority.

TransitionLabel: This is the base class for all kinds of transition labels. Direct subclasses are: `CompoundLabel`, `StringLabel`.

CompoundLabel: An instance of this class represents a label that is composed of a condition, an effect, and a trigger.

StringLabel: An instance of this class represents a transition label as a string. The string has to be parsed to split it into a condition, an effect, and a trigger. The `CompoundLabel` is the parsed form of the `Stringlabel`.

Variable: This class is the base class for all kinds of variables. Currently, variables of type integer (32-bit, 16-bit, 8-bit), double, float, boolean and string are supported. Direct subclasses are: `StringVariable`, `IntegerVariable`, `Integer16Variable`, `UInt32Variable`, `UInt16Variable`, `UInt8Variable`, `Integer8Variable`, `DoubleVariable`, `FloatVariable`, and `BooleanVariable`.

Constant: This class is the base class for all kinds of constants. Currently, constants of type integer (32-bit, 16-bit, 8-bit), unsigned integer (32-bit, 16-bit, 8-bit), double, float, boolean, and string are supported. Direct subclasses are: `StringConstant`, `IntegerConstant`, `Integer16Constant`, `Integer8Constant`, `UInt32Constant`, `UInt16Constant`, `UInt8Constant`, `DoubleConstant`, `FloatConstant`, `BooleanFalse`, and `BooleanTrue`.

2.2.2. Stateflow Objects

Stateflow is a modeling tool by *The Mathworks* [13]. It borrows concepts from *Statecharts* which have been first introduced by David Harel [2]. Therefore, you can treat *Stateflow* as a *Statechart* variant. Figure 2.5 shows the *Stateflow* application window. The following text describes *Stateflow* charts in detail.

Stateflow diagrams are embedded into a *Simulink* environment which serves as a source of input and a sink of output to the diagram. The output of a diagram controls a physical plant modeled in *Simulink*. A *Stateflow* diagram represents an object with discrete modes denoted as *states*. It reacts to events by switching from one state to another. The behavior of the object modeled by the diagram depends

2.2. State Diagram

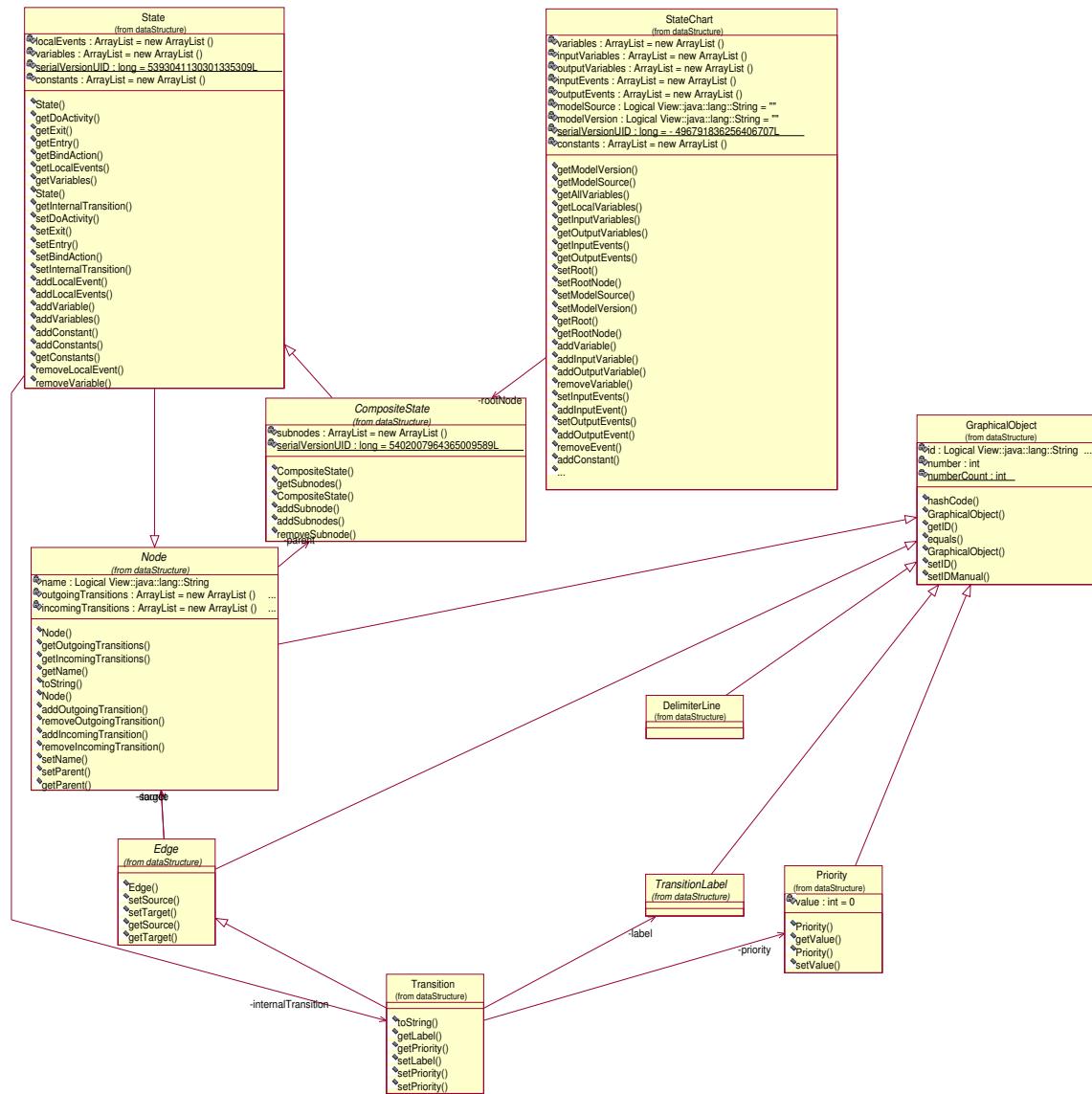


Figure 2.1.: The *KIEL* data structure class diagram

2. Theoretical Basis

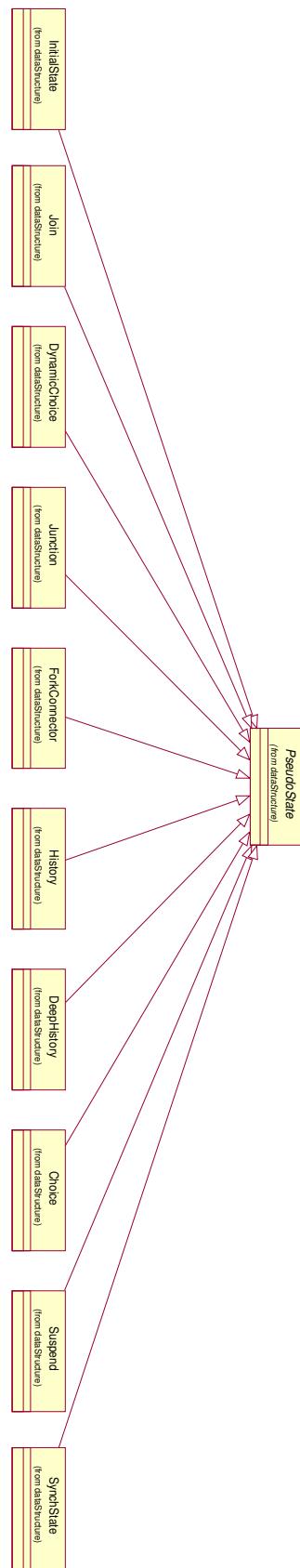


Figure 2.2.: PseudoState and its subclasses

2.2. State Diagram

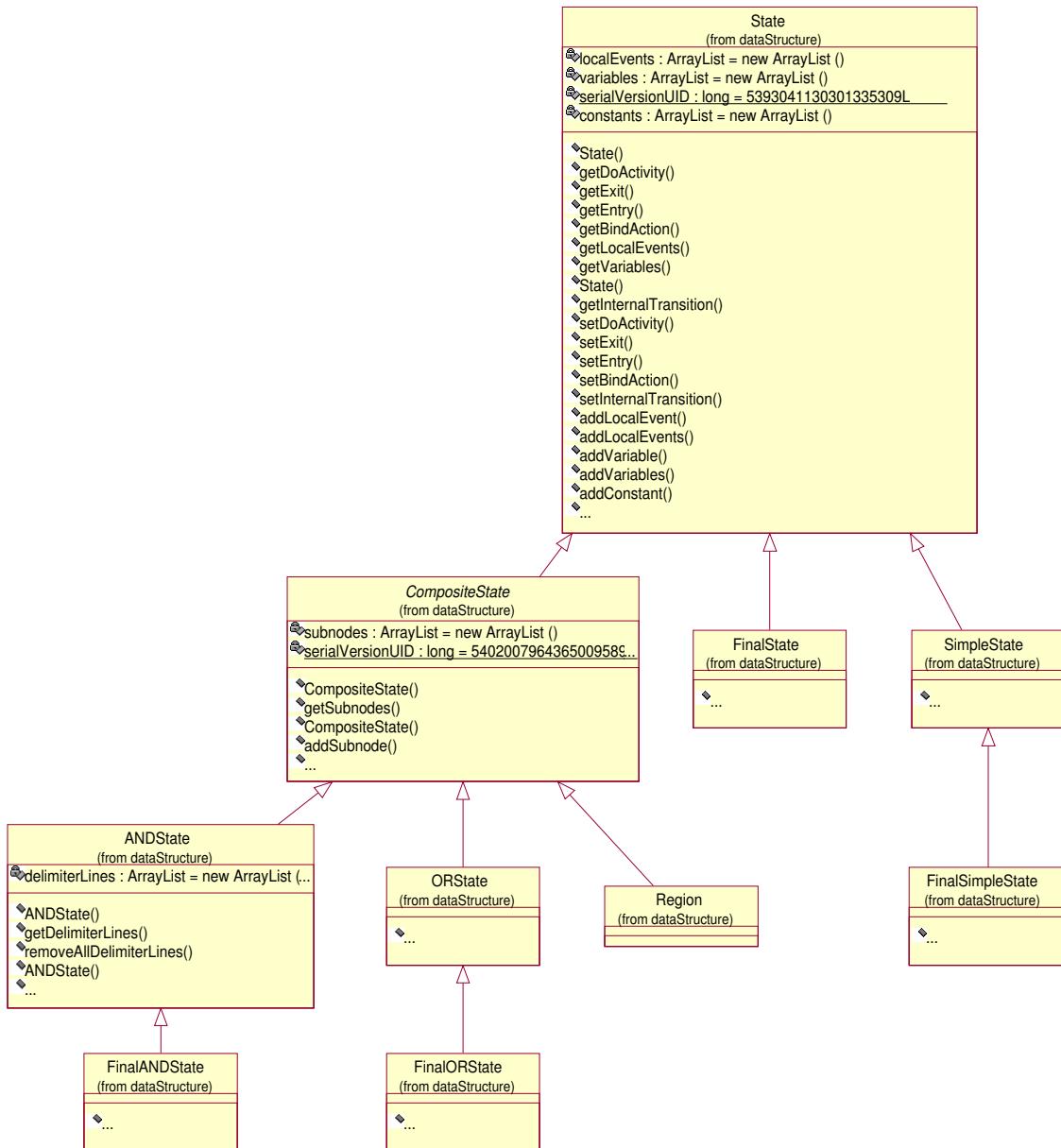


Figure 2.3.: State and its subclasses

2. Theoretical Basis

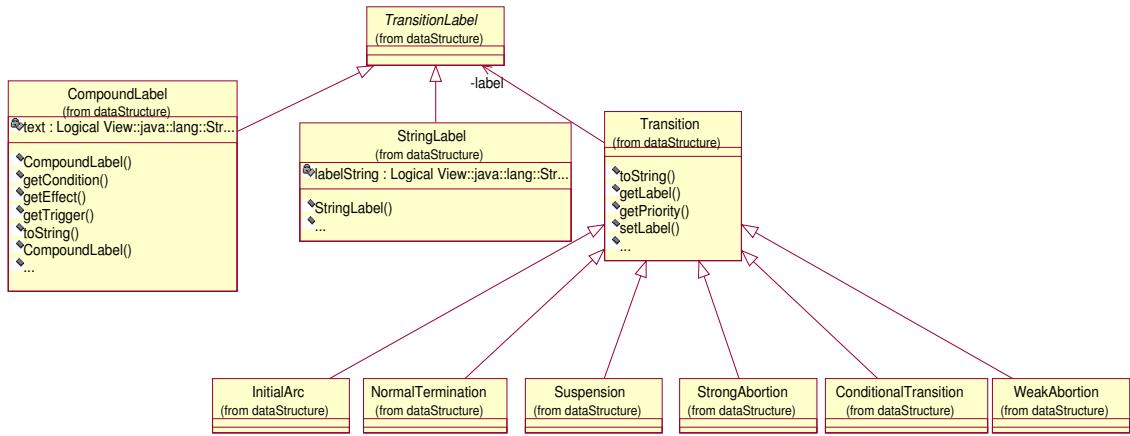


Figure 2.4.: Transition and its subclasses

on the current state the diagram is in and on how the diagram switches from one state to another.

A *Stateflow* diagram consists of:

- A set of graphical objects
- A set of non-graphical objects
- Defined relationships between those objects

Figure 2.6 shows all *Stateflow* objects and their hierarchy. Table 2.1 shows the notation of the graphical objects.

States

A state describes a particular mode of the system. It can be either active or inactive. The switch between active and inactive takes place due to events and conditions. In other words, the execution of the diagram is driven by the occurrence of events, which make states become active or inactive. At any instance of time, there is a set of active states. Every state has a parent, that means it is contained by another state. This is the concept of hierarchy. The diagram itself is called the Stateflow diagram root and it contains all other states. The activity history of a state can be recorded in a *history junction*, which makes it possible to make future activity depend on past activity. States have labels that can specify actions, which are executed in sequence according to the type of any particular action. The supported types of actions are *entry*, *during*, *exit*, *on event_name*, and *bind*.

The entry actions of a state are executed when the state becomes active. Its exit actions are executed when the state becomes inactive. Its during actions are executed when the state is active, an event occurs, and no transition out of that state can be taken. Data and events associated with a bind keyword can only be changed by actions of the state and its children they belong to. Other states can

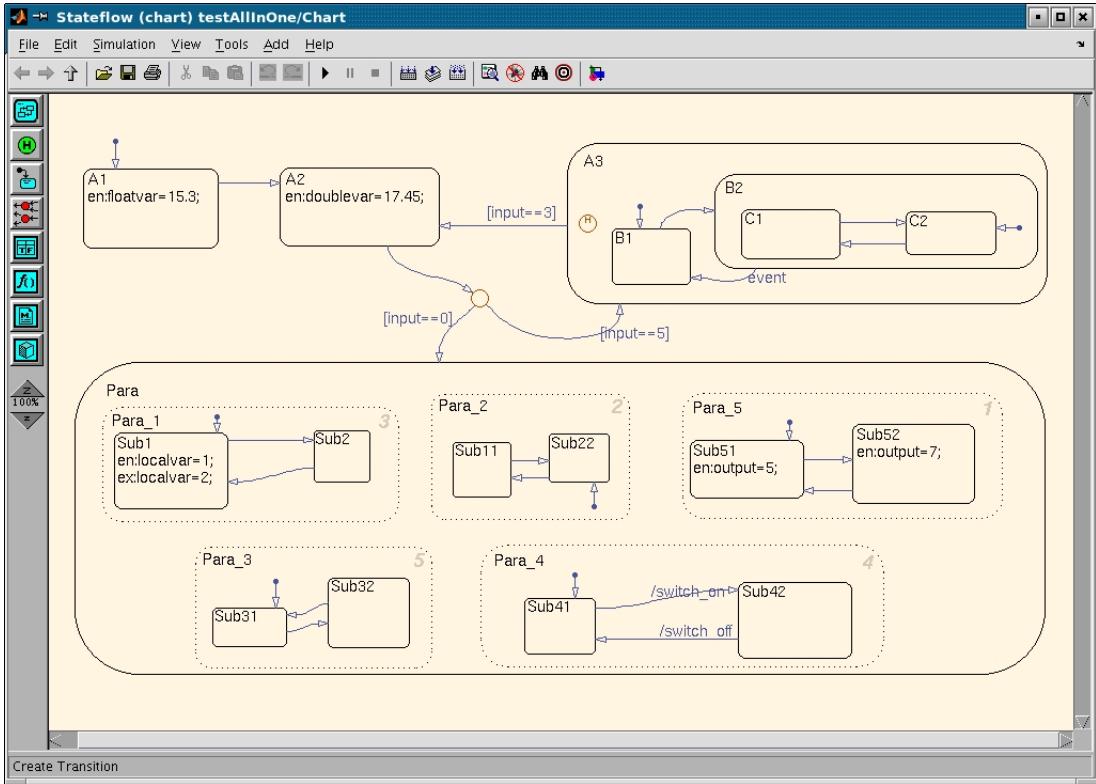


Figure 2.5.: A Stateflow window showing a diagram

only read the bounded data. The `on event_name` actions are executed when the state they belong to is active and the event `event_name` occurs.

To support parallelism *Stateflow* distinguishes between the two *decomposition* types “parallel and” and “exclusive or”. A state of decomposition type “exclusive or” contains exclusive (OR) states. Only one of the OR states can be active at an instance of time. On the other hand, a state of decomposition type “parallel and” contains parallel (AND) states. All of the AND states at the same level in the hierarchy are either active or not active at the same time. AND states are drawn as dashed rectangles, and they are independent of each other.

States can be classified as superstates, substates, or just as states. A state is a superstate if it contains other states, which are called substates. A state that is neither a superstate nor a substate is just a state.

The label of a state has to be of the following format:

```
name/
entry:entry action
during:during action
exit:exit action
on event_name:on event_name action
bind:events, data
```

2. Theoretical Basis

The label always begins with the name of the state followed by an optional “/” character. The lines that follow the name are keyword prefixes that identify particular types of actions. The supported types of actions have already been presented above. Each keyword prefix is followed by an action statement. Multiple actions for each action type have to be separated either by a carriage return, a semicolon, or a comma. There can be multiple `on event_name` lines thus specifying multiple `on event_name` actions for different events. Each keyword is optional, and the keywords do not have to appear in a particular order. The action statements are a part of the *Stateflow* action language.

Transitions

A transition usually links one object to another. It has a source, a destination, and a label. The transition label defines the circumstances under which the associated transition is taken. To take a transition means to leave the state that is the source of the transition and to enter the state that is the destination. The occurrence of an event together with the satisfaction of a particular condition causes a transition to be taken. Transitions are drawn as curved lines with an arrowhead at the destination side.

Default transitions specify which OR state within a superstate of decomposition type “exclusive or” has to be entered when the superstate is entered.

The label of a transition has to be in the following format:

`event [condition] {condition_action}/transition_action` The event named in the label causes the transition to be taken if the event is present and the given condition is true. If no event is named in the label, then the transition will react to any event. It is possible to specify multiple events in the label by using the logical OR operator. The condition in the label is a boolean expression. If the condition evaluates to true and the named event in the label is present, then the associated transition will be valid to be taken. The condition action, which follows the condition and is enclosed in curly braces, is executed as soon as the condition is evaluated to true and right before the destination of the transition has been determined to be valid. In case no condition is specified, the condition action will be executed anyway. The transition action is executed right before the destination state has been entered.

Events

Events are non-graphical objects, which drive the execution of a diagram. If an event occurs, then the status of every state in the diagram will be evaluated. This can result in a transition to be taken or an action to be executed. Events can be created at any level in the hierarchy of the diagram. They are visible to the state they belong to and to its children only. An event is present for one calculation step only. After all new active states have been determined, all events are absent again.

An event can be:

- Local to the *Stateflow* diagram

- An input to the *Stateflow* diagram from its *Simulink* model
- An output to its *Simulink* model from the *Stateflow* diagram
- Exported to a code destination that is external to the diagram and the *Simulink* model
- Imported from a code source which is external to the diagram and the *Simulink* model

The last two points are not relevant for this work.

Data

Data objects are variables that hold numerical values. They are non-graphical objects.

A data object can be:

- Local to the *Stateflow* diagram
- An input to the *Stateflow* diagram from its *Simulink* model
- An output to its *Simulink* model from the *Stateflow* diagram
- Non-persistent temporary data
- Defined in the Matlab workspace
- A constant
- Exported to a code destination that is external to the diagram and the *Simulink* model
- Imported from a code source that is external to the diagram and the *Simulink* model

The last two points are not relevant for this work.

Connective Junctions

Connective junctions are graphical objects, which are used together with transitions to provide a control flow construct. They are displayed as small circles. A connective junction divides a complete transition into transition segments. Each connective junction is a decision point between alternate transition paths. Connective junctions can be used to construct flow diagrams. That are diagrams which consist of transitions and connective junctions only. It is important to notice that an event cannot trigger a transition from a connective junction to a destination state.

2. Theoretical Basis

Object Name	Graphical Notation
State	
Transition	
Default Transition	
Junction	
History	

Table 2.1.: Notation of *Stateflow* Objects

History Junctions

A history junction records the most recently active state of a superstate. If a superstate containing a history junction is entered, then the state that had been active as the superstate became inactive before will become active again. In case the superstate is entered for the first time, the active state will be determined by the default transition. Please note that the history junction only applies to the level of the hierarchy in which it appears.

Actions

Actions appear in transition and state labels. Their execution is part of the diagram execution. Transition actions are executed when the associated transition is taken. Actions in states are executed according to their type as described in Section 2.2.2. An action can be a function call, the broadcast of an event, the assignment of a value to a variable, and so on. The operators that are available in the action language are similar to the operators known from high level programming languages like *C*. Among these are Multiplication, Division, Addition, Subtraction, bitwise operators, logical operators, Assignment, pointer and address operators. There are also temporal logic operators, which is something usually not found in programming languages. For example, the operator `after` takes an event and an expression `n` that evaluates to a positive integer value. The operator evaluates to `true` if the given event has occurred `n` times.

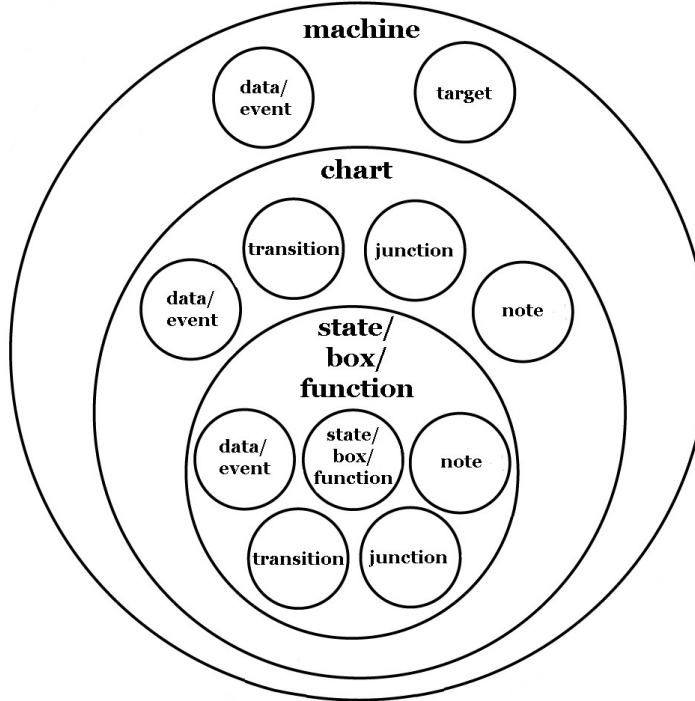


Figure 2.6.: Stateflow object hierarchy (Source: Stateflow User’s Guide [7])

2.3. Matlab/Stateflow-API

Matlab/Simulink/Stateflow provides a set of commands that let the user load, save, create, and manipulate state diagrams, define settings, start a simulation, retrieve simulation results. Nearly everything that can be done using the GUI can also be done using Matlab’s commands. The user can enter these commands at the Matlab prompt. It is possible to write these commands into a file, so that they form a script. Scripts automate the work like the editing of a diagram. They can be invoked within the command line like the build-in commands. Although *The Mathworks* calls this set of commands an API, it is not designed to be used within a programming language. However, as long as it is possible to attach Matlab’s input and output streams to a process that wants to interact with Matlab, this process can generate Matlab “scripts” on-the-fly. A study of the possibility of a simulation using the API was done by Jan Täubrich [10].

Each object in *Stateflow*, including the graphical and non-graphical objects described in 2.2.2, has a corresponding object in the API. Figure 2.7 shows all API objects and their hierarchy. An API object bundles a set of specific functions and properties. The change of a property of an API object also changes the property of the corresponding *Stateflow* object. The API objects, which correspond to the objects described in Section 2.2.2, have the same hierarchy as their counterparts. There are some objects that do not fit into the hierarchy model like the **Editor**

2. Theoretical Basis

and the `Clipboard` object. The `Editor` object exists to provide access to purely graphical aspects of a chart like the size of the chart in pixel. The `Clipboard` object provides copy and paste functionality. The `Machine` object in figure 2.6 corresponds to the `model` object in the API. This object represents the *Simulink* model, which includes the *Stateflow* diagram. The `Root` object contains all *Stateflow* API objects. It exists to have a means to distinguish *Stateflow* API objects from API objects of other tools like *Simulink*. The `Chart` object represents the *Stateflow* diagram. There can be multiple diagrams within a model.

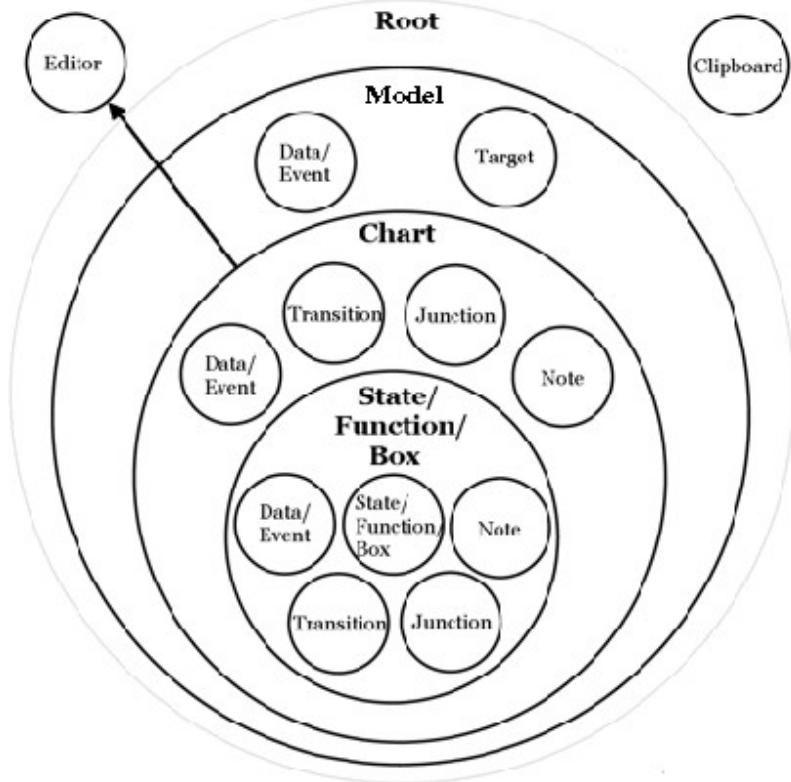


Figure 2.7.: Stateflow API object hierarchy (Source: Stateflow API reference [6])

API objects are accessed through variables that contain references to these objects. Such variables are called *handles*. API object properties correspond to the values one can set using the diagram editor. API object methods provide a means to create, delete, change, and find the object they belong to. All objects have some properties and methods in common. For example, the `Id` and the `Description` property are common properties. On the other hand, there are properties and methods that are specific to the type of object, like the `Source` or `DecompositionType` properties. A complete list of all methods and properties can be found in the *Stateflow* API reference [6].

By convention, all property names start with a capital letter, while all method names start with a letter in lowercase. In names that consist of concatenated words, each word starts with a capital letter.

2.4. Comparison of KIEL Data Structure classes with Stateflow API Objects

To access a property or method of an object, the so-called dot notation is used. One writes down the name of the variable that contains the handle to the object followed by a dot character and then the name of the property or method one wants to access. It is allowed to write down nested dot expressions. If a function call needs to return more than one object, it returns an array of objects. To access a specific item in an array, one has to append the index of the item needed enclosed in round braces. Example:

```
label1 = sA1.innerTransitionsOf(1).LabelString
```

Special constructor methods exist to create new objects. They return handles to the newly created objects. The constructor methods used in this work are: `Stateflow.Data`, `Stateflow.Event`, `Stateflow.Junction`, `Stateflow.State`. The created objects have a destructor method called `delete`.

2.4. Comparison of *KIEL* Data Structure classes with *Stateflow* API Objects

To represent a *Stateflow* chart in the *KIEL* application, it is necessary to associate each *Stateflow* API object with an object in the *KIEL* data structure. The Table 2.2 associates the classes of the *KIEL* data structure with the objects of the *Stateflow* API. Not all objects of the *KIEL* data structure do exist in the API and the other way around. As long as no support for graphical functions and truth tables is needed, no new classes need to be created in the *KIEL* data structure (except for the support of variables of type *double* and *float*). **Action** and **Event** do not exist as objects in the *Stateflow* API. They are included as strings in transition labels. The transition labels need to be parsed to create the objects **Action** and **Event**. Because *Stateflow* uses dashed rectangles to represent parallel states instead of delimiter lines, these delimiter lines (represented by **DelimiterLine**) have to be generated during the import of *Stateflow* charts.

2. Theoretical Basis

KIEL data structure	Stateflow API
StateChart	Stateflow.Chart
ORState	Stateflow.State.Decomposition= 'EXCLUSIVE_OR' Stateflow.State.Type='OR'
ANDState	Stateflow.State.Decomposition= 'PARALLEL_AND' Stateflow.State.Type='AND'
SimpleState	non-existent, use an ORState
FinalState	non-existent
Final*	non-existent
InitialState	non-existent
PseudoState	non-existent
Junction (not dynamical)	Stateflow.Junction.type= 'CONNECTIVE_JUNCTION'
History (shallow)	Stateflow.Junction.type= 'HISTORY_JUNCTION' Stateflow.Junction.labelX="H"
DeepHistory, Choice, DynamicChoice, ForkConnector, Join, Suspend, SynchState	non-existent
Transition	Stateflow.Transition
TransitionLabel/StringLabel	Stateflow.Transition.LabelString
CompoundLabel	non-existent
ConditionalTransition	Stateflow.Transition
NormalTermination,	
StrongAbortion, WeakAbortion	
Suspension	non-existent
Region	non-existent
Signal	non-existent
Variable	Data
StringVariable	non-existent
Event	Stateflow.Events
Actions/Action	non-existent as Obj.; in transition label only
non-existent	Machine, Box, Note, Target
non-existent	GraphicalFunction
non-existent	TruthTable
GraphicalObject	non-existent
Node, Edge	non-existent
DelimiterLine	non-existent; has to be generated
Priority	exists only implicitly

Table 2.2.: Comparison of *KIEL* data structure classes with Stateflow API objects

3. Stateflow Extensions of KIEL

The previous chapter includes the background information needed to understand what this diploma thesis is about. The following sections introduce the new features that have been added to the *KIEL* application. The features are described from the user's point of view. Details of the implementation are given after it.

3.1. Requirements

The *KIEL* program is supposed to be able to import charts that have been created with *Stateflow* [7] and are stored in model files that have the file name extension "mdl". Charts that make use of one or more of the following features do not have to be supported: interlevel transitions, graphical functions, truth tables, *C*-function calls. Even no support is required for model files that include *Simulink* blocks or more than one *Stateflow* chart. The layout of the imported chart should be as close as possible to that of the original chart. For charts that include subcharts, it is not required to preserve the original layout during import, because the *KIEL* program does not support the concept of subcharts. Furthermore, the program has to be able to simulate *Stateflow* charts. Macro step simulation has to be supported, micro step simulation is not necessary. Simulation results have to be shown in the browser in the same way it is already done for *Esterel Studio* [1] charts. The table shown in the Browser has to be extended by tabs for input variables, output variables, and input events.

Events are a concept of *Stateflow*, while signals are a concept of *Simulink*. Signals trigger events. Simulating charts with *Stateflow*, the user specifies the values of the input signals for each instance of time. The change in value of a signal triggers the corresponding event either in case a falling edge or a rising edge occurs, or in both cases according to the type of the trigger. In *KIEL* the possibility of specifying events instead of signals has to be added to provide two different kind of views of the simulation input. *KIEL* has to support the file formats used by *Stateflow* and *Esterel Studio* to store the simulation input data. All simulation input data entered by the user has to be stored and the possibility to save this data in a format that *Stateflow* or *Esterel Studio* can read has to be provided. This is the input trace feature and it is described in detail in Section 3.3.

The editor included in the *KIEL* application is designed for the creation of *Esterel Studio* charts. It provides a special mode to support the creation of *Stateflow* charts, but this mode is not fully implemented. The editor is supposed to support the creation of *Stateflow* charts that make use of all features not excluded above.

3. Stateflow Extensions of KIEL



Figure 3.1.: The simulation toolbar of the *KIEL* browser

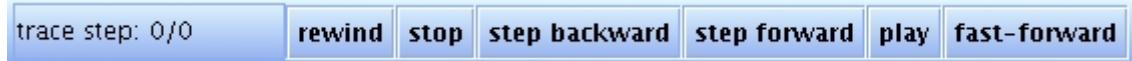


Figure 3.2.: The trace toolbar of the *KIEL* browser

The following sections describe the features that have been added to *KIEL* followed by details of their implementation.

3.2. How to simulate a *Stateflow* Chart

Charts are simulated inside the browser. All variables, signals, and events can be set in the table in the lower half of the browser window. The user can choose to specify either signals or events, but not both at the same time. The difference between signals and events is described in Section 3.1. A click onto **MacroStep** (Figure 3.1) computes the next step according to the current state of the chart and the input data from the table. As a result of a simulation step, all active states are marked, and the table shows the current values of all variables and which events are currently present. If the user selects one of the automatic layouters, then states that are not active are collapsed (interior is not shown) and the layout changes to hide non-interesting parts of the chart. The simulation of micro steps is not available for *Stateflow* charts. Figure 3.1 shows the simulation toolbar of the browser.

3.3. The input trace Feature

The input made by the user at each simulation step is recorded for the purpose of playback at a later date. The recorded input is called the *input trace*. The buttons **rewind**, **stop**, **step backward**, **step forward**, **play** and **fast-forward** (Figure 3.2) have been added to the browser to allow control over the recorded input data. The text field on the left side of the **rewind** button shows the current trace step number and the total number of trace steps that have been recorded. Every time the user clicks onto **MacroStep** (Figure 3.1) the input data (the set of signals that are present and the values of variables) of the current step is added to the trace. The user can replay the simulation steps he has made by rewinding the trace and clicking onto **step forward** to replay one simulation step or onto **play** to replay the simulation from the current step up to the last step. If the user clicks onto **MacroStep** while the current trace step is not the last one, then all steps that follow the current one are deleted and the next step is replaced by the new one. The input trace can be saved in the ESI file format as well as in the *Matlab M* file format.

Figure 3.3.: The *Esterel Studio* dependent part of the editor’s toolbarFigure 3.4.: The *Stateflow* dependent part of the editor’s toolbar

3.4. Import and Export of Stateflow Charts

The user can load a *Stateflow* chart by clicking onto “File”, and then clicking onto “Open..”. In the following dialog the user has to choose a file with the extension “mdl”. The user can specify “Matlab/Stateflow” in the file filter box to see “mdl” files only. To save a *Stateflow* chart the user has to click onto “Save” or “Save as..” in the “File” menu. For details on the implementation read Sections 3.6.6 and 3.6.7.

3.5. Stateflow Mode of the Editor

The editor supports the creation of *Esterel Studio* charts as well as of *Stateflow* charts. In dependence of the type of the chart currently edited, the editor shows one of two different toolbars (Figure 3.3 and 3.4). The tools used to create a deep history, a suspend state, a final state, and a dynamic choice are not present in *Stateflow* mode. There is a tool for the creation of a junction instead. The context menus are also different. For example, there is only one type of transition in *Stateflow*, so you cannot convert a transition to a weak, a strong, or a normal one. The dialogs that allow the editing of variables and of events have been re-designed (Figure 3.6 and 3.5). Most of the data types available in *Stateflow* are supported, and it is possible to set the trigger type of events as well as the scope of variables and of events.

3.6. Implementation

This section focuses on the implementation details of the added features described before. The overall structure of the *KIEL* application is presented. Details on the specific usage of the *Stateflow* API within *KIEL* are covered, and the algorithms used to import and export charts are described.

3.6.1. Surroundings

The *KIEL* program is written in *Java*, Version 1.4.2. An advantage of this language is its independence of any particular operating system. Care has been taken not to

3. Stateflow Extensions of KIEL

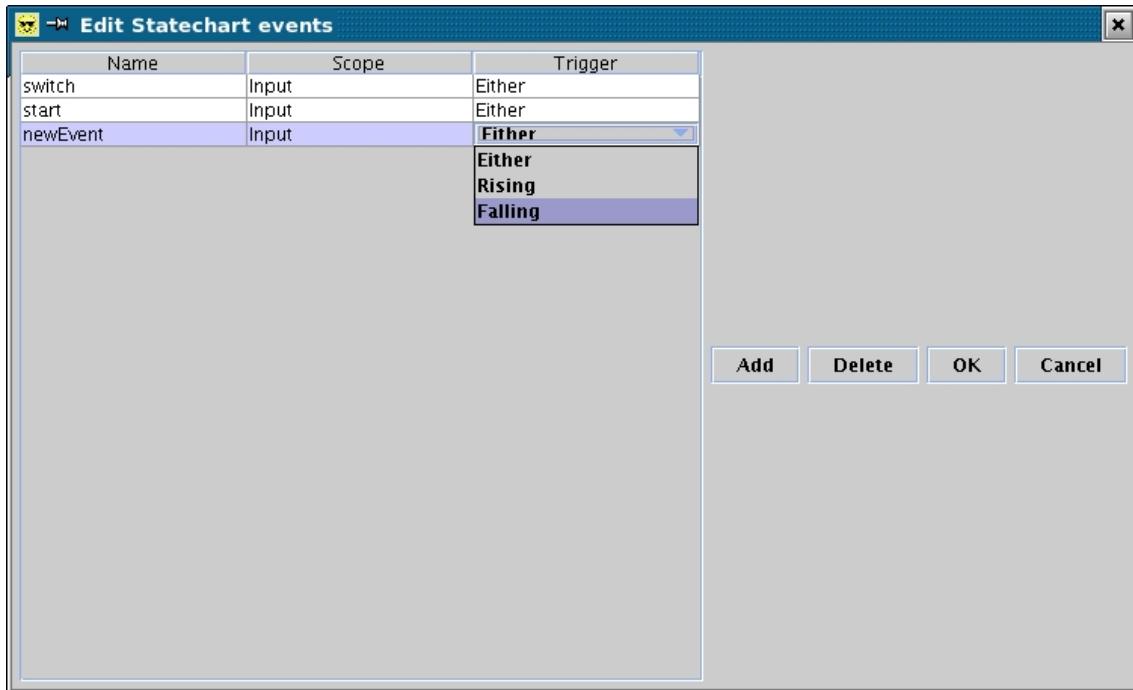


Figure 3.5.: The editor's dialog for editing chart events

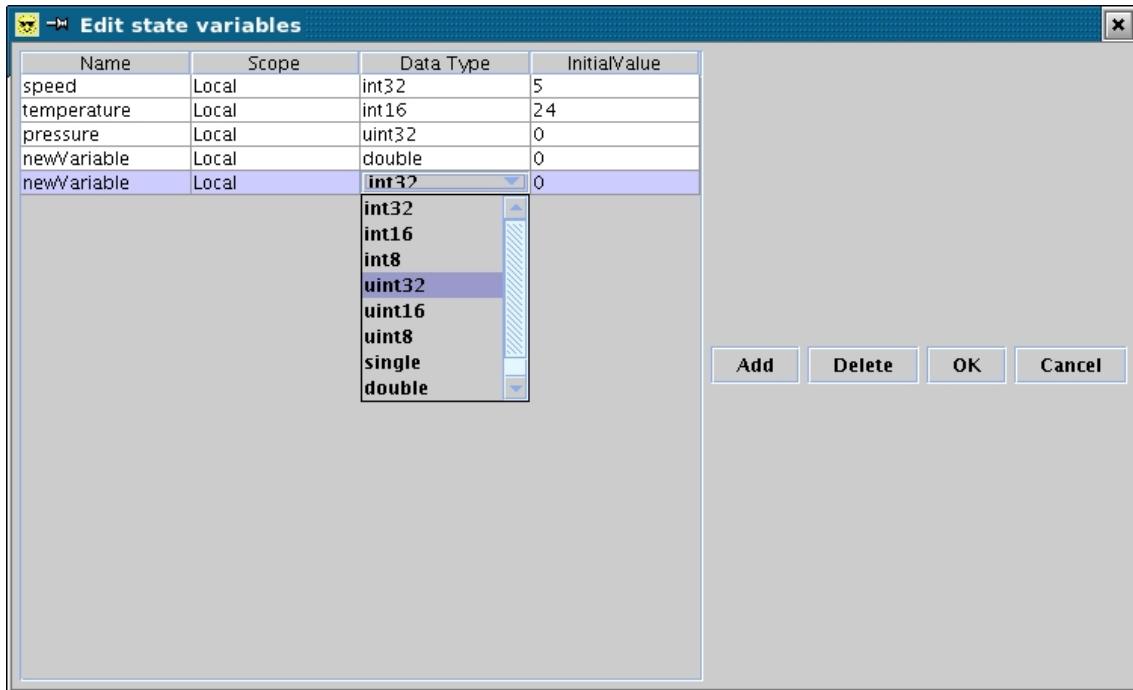


Figure 3.6.: The editor's dialog for editing state variables

include any operating system specific program code. *KIEL* is regularly tested on SuSE Linux 9.2 and Microsoft Windows XP.

Structure of *KIEL*

The *KIEL* program consists of several independent components, which have been developed in the scope of student projects and diploma theses. These components communicate through well defined interfaces, which make them easily interchangeable. They all share the *KIEL* data structure, which is described in Section 2.2.1. Figure 3.7 shows the components as well as their communication paths. The following components do currently exist:

Browser: It allows control of the simulator and shows the simulation results [15].

Editor: It enables the user to create new charts and to modify existing ones [4].

Layouter: The layouter [3] provides several layout algorithms for automatic layout from within the browser and the editor.

Simulator: It performs the necessary calculations for the simulation of the charts. Currently, a simulator for *Esterel Studio* charts is implemented [9].

File Interface: Several import and export filters exist as plug-ins to load and save charts and simulation input data [14]. Currently, it is possible to load and save *Stateflow* charts and *Esterel Studio* charts. ESI and Matlab M files can be read and saved.

KielFrame: This is the main component that provides the program's main window. The main window displays either the browser or the editor. It is possible to switch between both at any time. *KielFrame* also exists to separate all components of the program to keep them independent from each other. Therefore, it encapsulates the interfaces to the simulator, the layouter, and to the file interface. Neither the browser nor the editor calls any function of the other components directly; they call functions of the *KielFrame* instead, which then calls the appropriate functions of the modules.

3.6.2. Communication with *Matlab*

Usually, the user enters *Matlab* commands at *Matlab*'s command line or he writes them into a script file. To use the API from within *KIEL*, one could write a routine that generates a *Matlab* script, and then let *Matlab* execute this script. *Matlab* would write its response into a file that the *KIEL* program can read. This way of communicating with *Matlab* produces unnecessary disk accesses. *Java* provides a way to directly communicate with other applications using input and output streams. These streams can be obtained by the methods `getInputStream()` and

3. Stateflow Extensions of KIEL

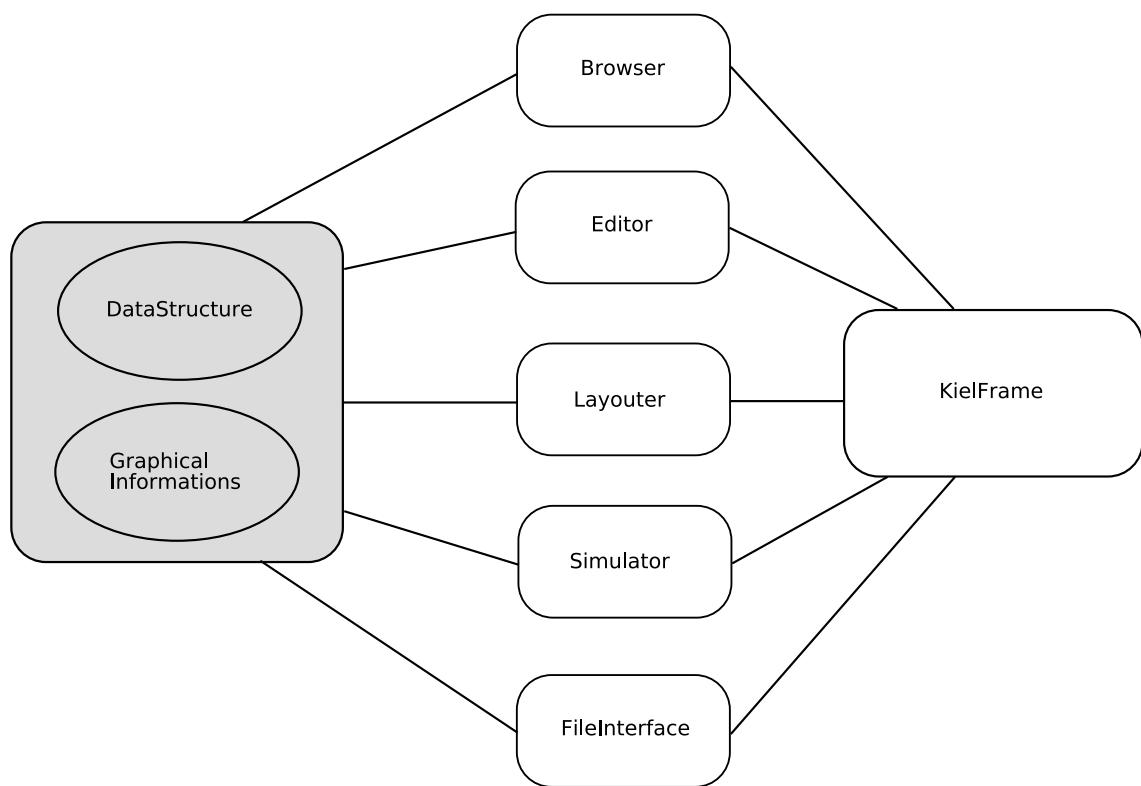


Figure 3.7.: Components of the *KIEL* application

`getOutputStream()` of the class `Process`. An instance of the `Process` class is obtained during the creation of the *Matlab* process with one of the `Runtime.exec` methods.

Example of communication with *Matlab*: Let `states` be a handle to states in one level of hierarchy of a chart that has to be imported. The following messages are exchanged between *KIEL* and *Matlab* to retrieve information about a state.

```
size(states)
```

```
ans =
```

```
    2      1
```

```
>>
```

```
states(1).Position
```

```
ans =
```

```
    312.6471  247.3765   90.0000   60.0000
```

```
>>
```

```
states(1).Type
```

```
ans =
```

```
OR
```

```
>>
```

```
states(1).IsSubchart
```

```
ans =
```

```
0
```

```
>>
```

```
states(1).Id
```

```
ans =
```

```
20
```

```
>>
```

3. Stateflow Extensions of KIEL

```
states(1).Name
```

```
ans =  
      off  
>>
```

The following 3 sections describe how to use the API to find objects in the *Stateflow* chart, and how to manipulate them as well as how to set the simulation parameters with the API. They provide the details of the communication with *Matlab*. For an introduction to the API have a look at Section 2.3.

3.6.3. Finding *Stateflow* Objects using the API

The `find` method provides a way to find objects in the diagram. Practically, this method searches for all objects that are contained in the object the `find` method belongs to. It is possible to restrict the search depth. Therefore, it is not necessary to traverse the diagram object by object to find what you are searching for. The `find` method needs to know the type of object it has to find and, optionally, one or more property names and their values.

Example: Let `m` denote a variable containing a handle to a model object. Then the following command will return every State object with the name “on” in the model `m`:

```
onState = m.find('-isa','Stateflow.State',  
                  '-and','Name','On')
```

If `find` finds more than one object, then it will return an array of objects. It is important to note that the zeroth level of containment is always included in the answer of `find`. To limit the search depth, one uses the `-depth` specifier.

Example: Let `sA` denote a variable containing a handle to a state object. Then the following command will return every object that `sA` contains up to a level of containment of 2:

```
objArray = sA.find('-depth',2)
```

Using `find` one can find child objects only. To find the parent object, one has to use the `up` method.

Getting and setting the Properties of Objects

One can get the property of an object just by writing down the object handle variable followed by a “.” and the property, or by using the `get` method. Example:

```
d = obj.Description  
d = obj.get('Description')
```

One can set the property of an object by using the assignment operator or the `set` method. Example:

```
obj.Description = 'This is a description for the object.'
obj.set('Description', 'This is a description of the object')
```

3.6.4. Setting the Simulation Parameters using the API

The simulation parameters can be set using the API. The object `Simulink.ConfigSet` contains all settings for the simulation and for code generation. The settings appear in the Model Explorer as the item *Configuration (Active)*. One can get the actual configuration with the function `getActiveConfigSet`. Let `acs` denote the variable to which we want to save the handle to the current configuration object, and let `model` denote the variable containing a handle to the current model. Then the following command will store a handle to the actual configuration object into the variable `acs`:

```
acs = getActiveConfigSet(model)
```

The function `set_param` provides the possibility to change the values of parameters of the configuration object. To choose the discrete solver that is of type fixed-step, one has to enter the following two commands:

```
acs.set_param('SolverType', 'Fixed-step')
acs.set_param('Solver', 'FixedStepDiscrete')
```

Finally, we will need a way to change the simulation stop time and the sample time:

```
acs.set_param('StopTime', '5')
acs.set_param('FixedStep', '1.0')
```

In the example above, we choose a stop time of 5 seconds and a sample time of 1 second.

3.6.5. Manipulating the *Simulink* Model using the API

It is possible to modify the *Simulink* model with the help of the API. Commands exist to add, delete, and link *Simulink* blocks. To add a block to the model, one uses the `add_block('src', 'dst')` command. `add_block` copies the block with the full path name '`src`' to a new block with the full path name '`dst`'. The *Simulink* library provides an extensive set of blocks. One usually builds a model using these predefined blocks.

The parameters of the new block are identical to those of the original. The `set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)` function can be used to change the parameters of the block specified by '`obj`'. Pairs of parameters and values are given to `set_param` to set the parameters to the new values. To position a block, one changes the `position` parameter to a vector with

3. Stateflow Extensions of KIEL

coordinates according to the following scheme: [left top right bottom]. The origin of the coordinate system is the upper left corner.

One can link two blocks together using the `add_line('sys','oport','iport')` command. It adds a line from the output port of one block to the input port of the other block. `'iport'` and `'oport'` which consist of a block name and a block port identifier in the form `block/'port'`. Usually, ports are numbered from top to bottom or from left to right.

3.6.6. Import of *Stateflow* Charts

It is assumed that the user who wants to work with *Stateflow* charts has a *Matlab/Simulink/Stateflow* license. Since the *KIEL* application is not meant to replace *Matlab/Simulink/Stateflow*, the solution of the import problem may require it. One possible solution is to write a parser for *Simulink* model files. Every time the format of these model files changes, the parser has to be adapted. It has shown that the model file format is not documented. It has a great similarity to the API objects, but there are differences. Another possible solution is to let *Matlab* load the model file itself, and then use its API to get information about the chart object by object. This information is used to create an equivalent chart using the *KIEL* data structure. The API is well documented and unlikely to undergo dramatic changes in the future. This solution is therefore easier to accomplish. The second solution has been implemented and is described in detail in the following part. The class *MatlabStateflowGrabber* includes all the program code that is necessary to import a chart from *Matlab*.

The *KIEL* application starts *Matlab* when a *Stateflow* model file (file name extension “mdl”) has to be loaded. It then writes *Matlab* commands to the input stream associated with the *Matlab* process and receives *Matlab*’s response through the corresponding output stream. *Matlab*’s responses contain all the information that is needed to construct an equivalent chart using the *KIEL* data structure.

Import Strategy

The algorithm is shown in pseudocode in Figure 3.8. Starting at the root, the chart is traversed downwards level by level of containment. This allows the reuse of variables in the *Matlab* workspace and avoids a recursive approach. For each object in the chart an equivalent object is created using the classes of the *KIEL* data structure. Since interlevel transitions are not supported by *KIEL*, it is possible to complete the construction of one level of containment before the next level is entered. States have to be constructed before transitions are constructed, otherwise the algorithm might try to link a transition to a state which has not been created yet. The transitions in *Stateflow* refer to their source and destination states. While the transitions are generated in *KIEL*, it is necessary to associate the states in *Stateflow* with the states created in *KIEL*. The association is realized with a *HashMap* of the

`java.util` package. The states have ids in *Stateflow* that are used as keys for the `HashMap`.

Delimiter lines are used in *KIEL* to separate regions from each other. They do not appear in *Stateflow* charts because regions are drawn as dashed rectangles instead. Therefore, it is necessary to generate them during the process of import. To calculate the coordinates for the delimiter lines, the regions are sorted from left to right and from up to down, so that the upper left region is the first one and the lower right region is the last one. Between two adjacent regions a vertical delimiter line is set if they lie side by side, a horizontal line is set if they lie one upon another.

The positions of graphical objects are absolute in *Stateflow* according to the coordinate system of its editor. In contrast to *Stateflow*, the positions are relative to the positions of their parent objects in *KIEL*. During import, all position coordinates are converted from absolute to relative coordinates.

Stateflow supports the concept of subcharts. A subchart is a state that contains a complete chart. This concept is used to hide complexity. The contents of a subchart is never shown together with its parent. The *KIEL* application does not support this concept. Therefore, the original layout cannot be imported for charts which contain subcharts. One of the automatic layouters is invoked if a chart that has to be imported contains one or more subcharts.

Actions are represented as objects in the *KIEL* data structure. In contrast to this, there is no API object to represent actions. Actions are just stored as string labels in states and transitions. It is necessary to extract the actions from the state and transition labels. For each action that appears in a label within the chart we import, an object is created. However, the contents of the action is not parsed because the simulation is done by *Stateflow*, so we do not need to know what the action consists of, but we need to know which actions exist and of which type they are to be able to represent them correctly in the *KIEL* application. The contents of an action element is stored as an unparsed string in *KIEL*.

The spatial coordinates of transition destination end points are not available through the API. Instead, transition destination end points are specified as o'clock positions in the range of 0 to 12. The spatial coordinates are calculated from the o'clock value and the spatial coordinates of the center of the destination object. The spacial coordinates of transition source end points are available through the API. They are just transformed into coordinates that are relative to the upper left corner of the parent state. In addition to the source end and destination end point each transition in *Stateflow* has a midpoint. These 3 points are used to construct a path which consists of a `MovetoPath` and a `QuadraticCurvetoPath` path element. The `MovetoPath` path element contains the transition source end point as its target point. The `QuadraticCurvetoPath` path element contains the transition source end point and the transition midpoint as its supporting points and the transition destination end point as its target point. A path contains the graphical information which determines the layout of a transition.

3. Stateflow Extensions of KIEL

```
procedure grabChart
    set chart to "empty"
    call getAndCreateGlobalEvents with chart and rootState
    call getAndCreateGlobalVariables with chart and rootState
    call grabOneLevel with rootState and chart returning children
    set parentStates to children

    repeat
        set nextLevelParentStates to "empty"

        for each element in parentStates
            set parent to element
            call grabOneLevel with parent and chart returning children
            append children to nextLevelParentStates
        end for

        set parentStates to nextLevelParentStates
    until nextLevelParentStates contains more than zero elements
    return chart
end procedure

procedure grabOneLevel(parent, chart)
    set children to children of parent

    for each element in children
        call getAndCreateStates with element and chart
        call getAndCreateVariables with element and chart
        call getAndCreateEvents with element and chart
        call getAndCreateJunctions with element and chart
        call grabAndCreateTransitions with element and chart
    end for
    call createDelimiterLines with children and chart
    return children
end procedure
```

Figure 3.8.: Algorithm in pseudocode for import of *Stateflow* charts

3.6.7. Export of Stateflow Charts

To export a chart it is possible to write a mdl file directly, which is difficult due to the lack of a good and complete documentation of the file format. Therefore, the same API based approach as for the import feature has been chosen for the export feature. In a first step, a new chart that is equivalent to the current chart in the *KIEL* application is generated in *Stateflow* with the use of the API. In a second step, *Matlab* is told to save the new chart to a file. In case the user switches from the *KIEL* editor to the browser, the second step is omitted. The class `MatlabStateflowCreator` implements the *Stateflow* chart export feature.

Export strategy

The algorithm is shown in pseudocode in Figure 3.9. The chart is created level by level of containment in a non-recursive manner. For each level, states and junctions are created before any transitions are created to avoid the situation where a transition is going to be linked with a state or a junction that has not been created so far.

To confer unique names to variables in the *Matlab* workspace, the return value of the `java.lang.object.hashCode()` function is used at the end of every variable name. Variable names of variables that contain handles to states consist of the state's name followed by an underscore and the corresponding hash code of the object that contains the state. Variable names of variables that contain handles to transitions consist of the prefix `trans_` and the hash code of the object that contains the transitions. Variable names of variables that contain handles to junctions or history entities are constructed in the same way except the prefix is `junc_` or `hist_` respectively.

The labels of states are constructed out of the name of the state and out of the actions that are associated with the state. Each type of action has a prefix as described in Section 2.2.2.

Delimiter lines are omitted because they do not exist in *Stateflow*. Regions are called parallel states or AND states in *Stateflow* and are drawn as dashed rectangles. The positions of graphical objects are relative to their parent in *KIEL*, but absolute in *Stateflow*, so they have to be converted during the export.

The source end points and destination end points of transitions cannot be set directly as spatial coordinates, but they have to be specified as o'clock positions. An o'clock position is in the range of 0 to 12. These points are stored as coordinates in *KIEL*, so the appropriate o'clock value is calculated from the point coordinates and the center point coordinates of the source or destination object respectively. Source end points of default transitions form an exception because an o'clock value makes no sense here. The x and y coordinates are set directly for source end points of default transitions. The spatial coordinates of the transition midpoint are taken from the path element that specifies a point between the source and the destination end point. If the path consists of n elements with $n \geq 3$, then the target point

3. Stateflow Extensions of KIEL

of the $n/2$ -th path element is taken as midpoint. If the path consists of 2 path elements only and the second path element is of type `QuadtraticCurvetoPath`, then the second supporting point of this path element is taken as midpoint. If the path consists of 2 path elements and the second element is of type `lineto`, then the midpoint is calculated from the source point and end point coordinates.

3.6.8. Trace of input data

The package `kiel.simulationTrace` contains the classes `TraceData` and `TraceStep`, which implement the trace feature. An instance of the class `TraceData` holds the input trace data. For each trace step it contains an instance of the class `TraceStep`. A `TraceStep` contains a list of objects of type `Signal` for signals which are present, of type `SignalValue` for valued signals which are present, of type `VariableValue` for variables. It is necessary to load the chart before the trace file is loaded because the objects of type `Signal`, `IntegerSignal`, and `Variable`, which are contained in the loaded chart, are needed to construct the `TraceStep` objects. If the trace file is loaded first, then the trace steps will not contain any objects which makes the loaded trace useless.

3.6.9. Simulation of *Stateflow* Charts

The class `StateflowSimulator` in the package `kiel.simulator.matlab` contains the implementation of the simulator for *Stateflow* charts. It implements the interface `Simulator`. The *Stateflow* simulator does not perform the calculations itself. It just collects the input data from the tables of the browser and sets the variables in the *Matlab* workspace accordingly. Next, it sends the simulation command to *Matlab*, collects the simulation results and creates objects of type `MicroStep` according to these results. These `MicroStep` objects are put into a `MacroStep` object, which is the final result of a simulation step. The order of the micro steps does not represent the order of the computations made by *Stateflow*. The list of micro steps is not complete, but it contains all micro steps that are needed to display the result of a simulation step. Micro steps that are needed are those that determine the active states as well as the values of output signals and output variables. Micro steps that determine which transitions have been checked for validity are not needed. This information cannot be retrieved by *Stateflow* and is therefore unavailable. That is why the micro step simulation mode cannot be implemented for *Stateflow* charts as long as *Stateflow* is used to perform the simulation.

`MicroStep` is an interface that is implemented by the following classes: `ChangeValue`, `SignalStatus`, `StateStatus`, and `TransitionStatus`. Figure 3.10 shows the corresponding class diagram. Objects of type `TransitionStatus` are not used by the simulator because the kind of information represented by them is not available. Objects of type `ChangeValue` contain new values for variables and for valued signals. They are used to report changes of values of variables and of valued signals. Direct subclasses are: `SignalValue` and `VariableValue`. The class `SignalStatus` has the

```

procedure createChart
    call createGlobalEvents
    call createGlobalVariables
    call createOneLevel with rootState returning children
    set parentStates to children

    repeat
        set nextLevelParentStates to “empty”

        for each element in parentStates
            call createOneLevel with element returning children
            append children to nextLevelParentStates
        end for

        set parentStates to nextLevelParentStates
    until nextLevelParentStates contains no more than zero elements
end procedure

procedure createOneLevel(parent)
    set children to children of parent

    for each element in children
        call createState with element
        call createJunctions with element
        call createTransitions with element
        call createLocalEvents with element
        call createLocalVariables with element
    end for

    return children
end procedure

```

Figure 3.9.: Algorithm in pseudocode for export of *Stateflow* charts

3. Stateflow Extensions of KIEL

following direct subclasses: `SignalAbsent`, `SignalPresent`, and `SignalUnknown`. `SignalAbsent` represents the absence of the signal it contains, `SignalPresent` represents the presence of the signal it contains. The status of a signal contained in a `SignalUnknown` object is unknown. An object of type `StateActivated` in the list of micro steps means the state it contains is entered. It contains a list of all states that are currently active. An object of type `StateDeactivated` means the state it contains is leaved. The other types of micro steps that exist are not used in the *Stateflow* simulator.

Before a chart can be simulated, the simulink environment has to be prepared to allow input from and output to the *Matlab* workspace. The browser calls the method `setStateChart(StateChart)` after a chart has been loaded or the user has switched from the editor to the browser. This method deletes all simulink blocks except for the chart block, adds a `From Workspace` block for each input variable and one for all input signals that is connected to the trigger port of the chart. For each state, the `HasOutputData` option is set to true. This results in an output signal for each state whose value is 1 for every active state and 0 for every non-active state. All output signals are connected to a `Mux` block whose output is connected to a `To Workspace` block named `simout`. For each output variable, the option `SaveToWorkspace` is set to true. Their output ports remain unconnected. Section 3.6.5 explains how to manipulate the *Simulink* environment using the API. The solver type is set to `Fixed-step`, the solver to `FixedStepDiscrete`. The sample time is set to 1 second. Section 3.6.4 explains how to set the simulation parameters using the API.

If the user clicks onto the `MacroStep` button in the simulation toolbar of the browser and the browser is in the signal simulation mode, then the browser calls the method `emit(Event, int)` for each input signal of the chart. The integer parameter is the value of the signal. If the browser is in the event simulation mode while the user clicks onto `MacroStep`, then it calls the method `emit(Event)` instead. The browser calls the method `setVariable(Variable, String)` for each input variable. The second parameter is the string representation of the value of the variable. After that, it calls the method `nextStep()` which performs one simulation step and returns a `MacroStep` object that includes all simulation results. All states that are included in the list of active states in the last `StateActivated` object are marked as being active by the browser. For each `VariableValue` object, the browser changes the value of the corresponding variable to the new value contained in that object.

The method `nextStep()` constructs a matrix which contains the values of the input signals for each simulation step. Let m denote the total number of simulation steps and n the total number of input ports. Then the matrix has m rows and $(n + 1)$ columns. Column 1 row j contains the number j of the step, the entry in column i ($1 < i \leq n$), row j , contains the value at step j of the input signal whose port number is i . There is a row for each simulation step. The matrix is stored in a variable defined in the workspace. The name of the variable is equal to the name of the `From Workspace` block that is connected to the trigger port of the chart. Due to this name equality *Simulink* attaches the contents of the variable to this `From`

Workspace block. For each input variable a variable with equal name plus the postfix “VarMat” is defined in the workspace. The corresponding **From Workspace** blocks also have these names. Due to this name equality *Simulink* attaches the contents of these variables to the corresponding **From Workspace** blocks. Such a variable in the workspace contains a matrix with as many rows as there are simulation steps to simulate and 2 columns. The first column contains the step numbers and the second column contains the values of the variable at the corresponding step. After all these variables that include the input data for the simulation are defined, the method **nextStep** sets the parameter **StopTime** to the current step number (Each step number equals to 1 second.) and invokes the function **sim**, which belongs to the API. *Simulink* simulates the chart and stores the simulation results in variables that are defined in the workspace. For each output variable, *Simulink* creates a variable in the workspace whose name is equal to the name of the variable in the chart. The values of these variables are those determined at the end of the simulation. The variable **simout** contains the values of all output signals at each step as a matrix. This includes the output signals that report whether a particular state is active at a particular simulation step. The simulator examines the last two rows of this matrix. (The last row corresponds to the last simulation step and therefore contains the final simulation results.) It creates a **StateActivated** object for each state that is active in the last step and has been inactive in the previous step, and it creates a **StateDeactive** object for each step that is inactive in the last step and has been inactive in the previous step. Every active state that does not contain any states is added to the list of currently active states. This list is used to create a **Configuration** object. For each output signal that does not belong to a state, the simulator creates a **SignalPresent** object if the signal value switches between 1 and 0 (event is present) and a **SignalAbsent** object if the signal switches between 1 and 0 (event is not present). The simulator creates a **VariableValue** object for every output variable and local variable whose value has changed. All created micro steps are put into a **MacroStep** object which is returned as result by the method **nextStep**.

3.6.10. Adaption of the Editor

The type of the chart is stored in the field **modelSource** of the class **StateChart**. The editor calls the method **getModelSource()** to check whether the current chart is designed for *Esterel Studio* or for *Stateflow*. The toolbar is constructed by the method **updateToolBarAndMenuBar()** of the class **Editor** in dependence of the type of the current chart. The items the toolbar consists of are instances of the classes found in the package **kiel.editor.controller**. The class **EditorModeAddJunction** has been added to the package to support the creation of junctions. The graphical representation of a junction is determined by the class **JunctionRenderer**, which has been added to the package **kiel.editor.graph**. The mouse cursor that is shown while a junction is placed on the canvas is created using the image **JunctionAndCursor.gif**, which resides in the package **kiel.editor.resources**. The method **getMenuItems-**

3. Stateflow Extensions of KIEL

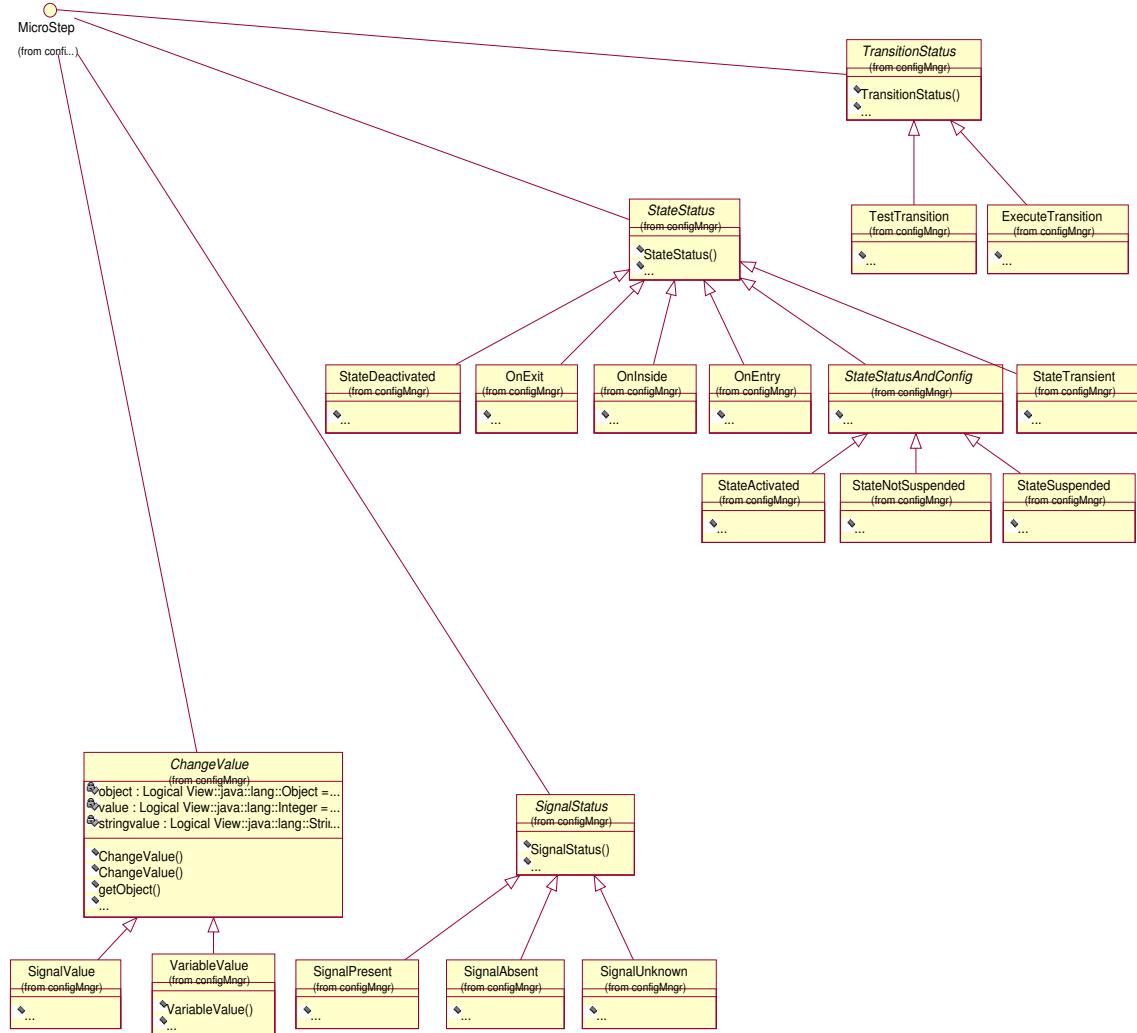


Figure 3.10.: The interface MicroStep and the classes that implement it

3.6. Implementation

`ForSelected()` of the class `MyJGraph` determines the menu items that appear in the context menu. In *Stateflow* mode, the items to change the type of a transition and the item that converts a state into a final state are omitted.

The reaction of the editor to user input is implemented by the classes in the package `kiel.editor.controller`. To implement new dialogs for editing variables and events without changing the dialogs that already exist for *Esterel Studio* charts, the following new controller classes have been added: `ActionEditStateChartEventsForStateflow`, `ActionEditStateChartVariablesForStateflow`, `ActionEditStateEventsForStateflow`, `ActionEditStateVariablesForStateflow`. As dialog window a `javax.swing.JDialog` is used. Variables and events are displayed in a table. This table is an instance of `javax.swing.JTable`. The classes that handle the user input of transition labels and actions in states cannot be used for *Stateflow* charts because they parse the entered strings with the help of the compound label parser that has been developed to support *Esterel Studio* charts. The following classes have been added to support unparsed user input of lables and actions: `ActionEditStateEntryActionsForStateflow`, `ActionEditStateDoActionsForStateflow`, `ActionEditStateExitActionsForStateflow`, and `ActionEditStateBindActionsForStateflow`.

3. Stateflow Extensions of KIEL

4. Concluding Remarks

This chapter summarizes the results of this work and mentions possible future enhancements. It includes the results of a performance test.

4.1. Results

The task was to add the ability to load, save, create, edit, and simulate *Stateflow* charts to the *KIEL* application. This required changes in the file interface, the browser, and the editor. These features were implemented with the help of the *Stateflow* application, which runs concurrently to *KIEL* and performs the core of the task. The communication between *KIEL* and *Matlab/Simulink/Stateflow* has been implemented using input and output streams. The commands that are exchanged are those defined by the *Simulink* and *Stateflow* API. The toolbar and the menu items of the editor as well as the dialogs for editing variables and events have been adapted to support *Stateflow* charts.

All goals mentioned in Section 3.1 have been achieved. The new *KIEL* application is well suited for the creation and the simulation of *Stateflow* charts. However, it lacks support for *Simulink* and for code generation. Therefore, it cannot serve as a replacement for *Matlab/Simulink/Stateflow*, but it is a good extension for it.

The execution of and the communication with *Matlab* consumes calculation time. The performance of *KIEL* with respect to loading, saving, and simulating *Stateflow* charts has been tested with charts of different complexity. The first loading or saving of a *Stateflow* chart requires the start of *Matlab*. If *Matlab* has already been started by *KIEL*, then the process of loading or saving a chart consumes less time. The first simulation step requires the compilation of the chart by *Stateflow*, which consumes additional time. All further simulation steps consume less time. Table 4.1 shows the results of time measurements for loading and simulation of four charts with different complexity, which are shown in Figure 4.1 to Figure 4.3. The measurements have been made on an AMD Athlon XP 3000+ processor with 1 GB RAM.

4.2. Future Prospects

The simulator of *KIEL* deletes all blocks in the *Simulink* diagram except for the chart block and constructs its own *Simulink* diagram. Therefore, it is not possible to simulate an existing chart within the *Simulink* diagram it has been developed for. A future version of the simulator might preserve the existing *Simulink* diagram.

4. Concluding Remarks

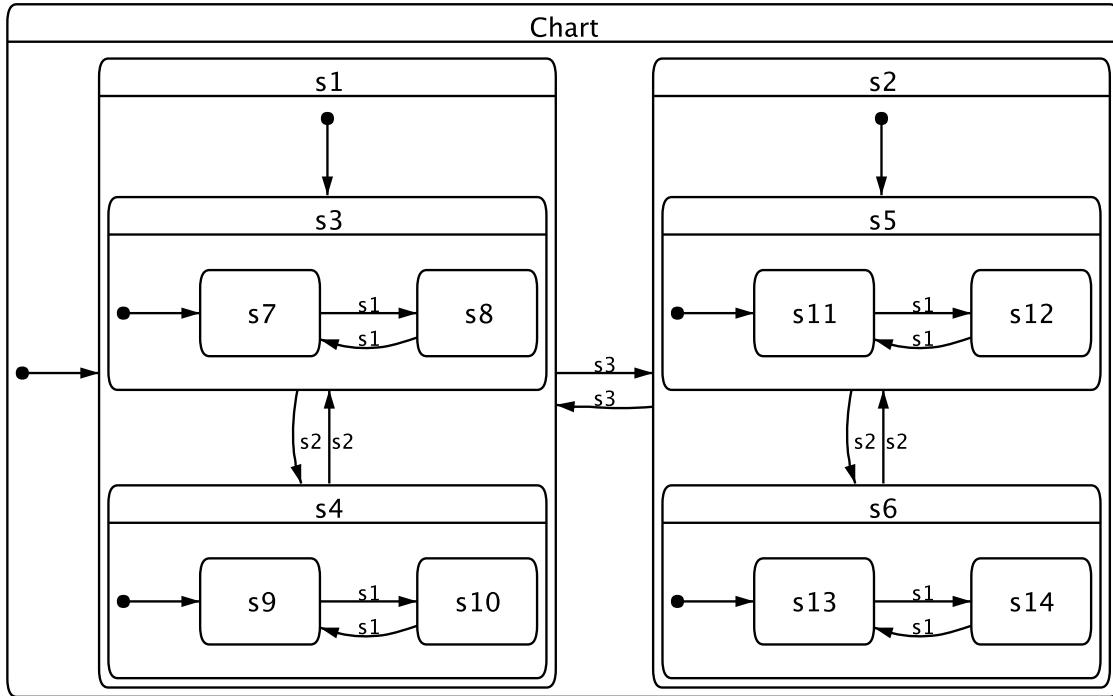


Figure 4.1.: The bintree3 Chart



Figure 4.2.: The bintree5 Chart

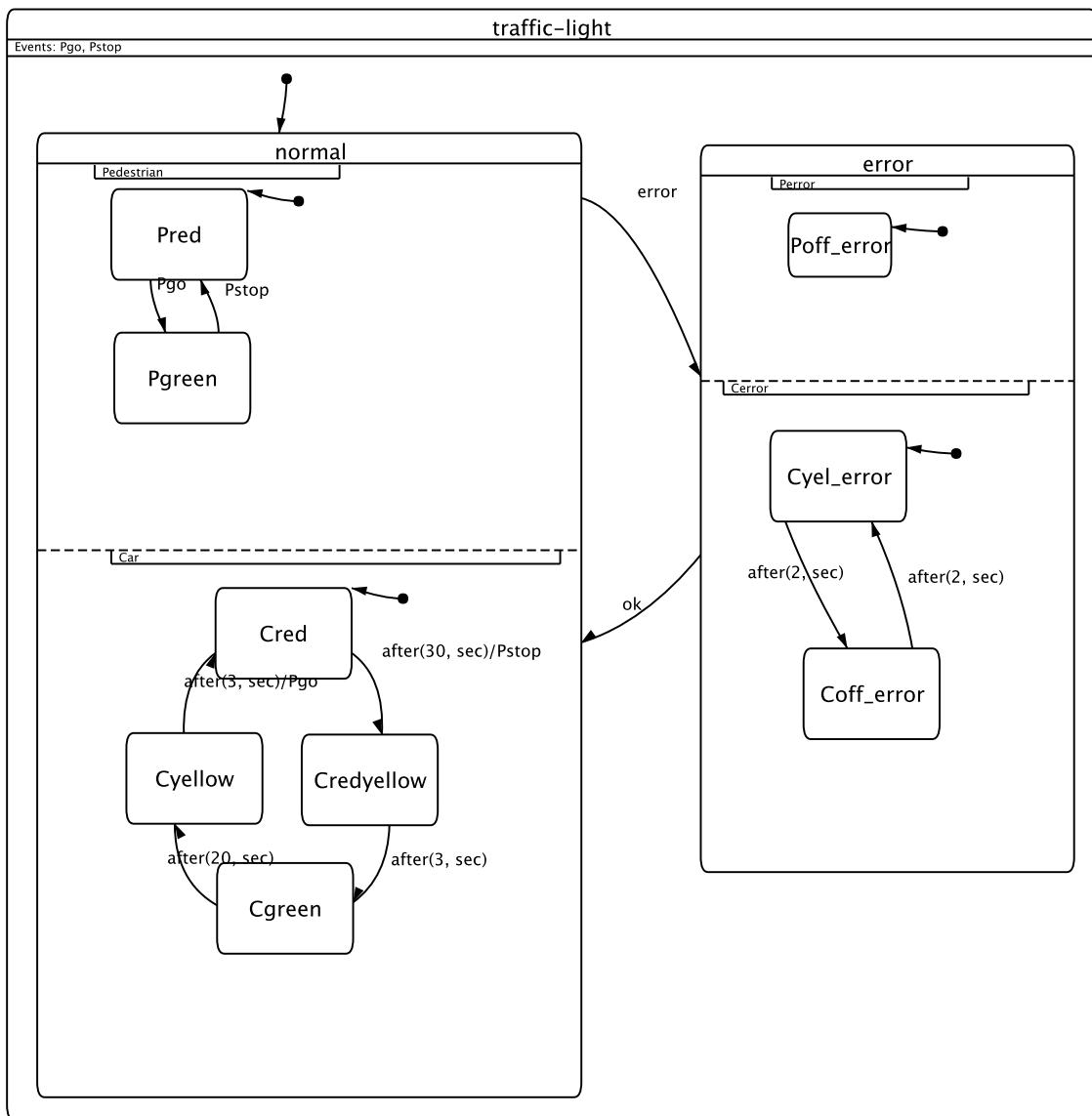


Figure 4.3.: The traffic light chart

4. Concluding Remarks

Chart	States	Transitions	loading Time	min Simulation Step Time	max Simulation Step Time
bintrie3	21	35	934 ms	221 ms	798 ms
bintrie5	93	155	1764 ms	341 ms	2162 ms
bintrie7	381	635	3993 ms	889 ms	11176 ms
traffic-light	15	25	2236 ms	249 ms	645 ms

Table 4.1.: Performace test results

4.2. Future Prospects

This approach does not provide the same level of control over the input to the chart because some input data is usually provided by other *Simulink* blocks. It is necessary to tap the input signals (lines) to be able to display the input data at each simulation step. To tap these signals, the simulator needs to analyse the *Simulink* diagram to determine which blocks provide input to the chart as well as the coordinates of these blocks. To get the output data, one can either tap the output signals in the same way as the input signals or one can use the “save to workspace” feature of *Simulink*. See Section “Exporting Output Data to the *Matlab* workspace” and “Logging Signals” in the *Simulink* documentation [5] for further details.

The *Stateflow* diagram elements Truth Table, Graphical Function, Box, embedded Matlab Function, and Note are currently not supported. Therefore, charts that make use of these elements can neither be imported into nor be created with *KIEL*. To add support for these diagram elements, it is necessary to add appropriate classes to the *KIEL* data structure and to extend the browser, the editor, and the simulator. A graphical representation for these diagram elements has to be implemented in the browser and the editor.

Interlevel transitions are currently not supported. The import and export algorithms for *Stateflow* charts used here cannot handle such transitions. The layouter module cannot handle such transitions either because the layout algorithms used there have been designed for graphs which do not support the concept of hierarchy.

4. Concluding Remarks

5. Bibliography

- [1] Esterel Technologies. Company homepage. <http://www.esterel-technologies.com>.
- [2] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [3] Tobias Kloss. Automatisches Layout von Statecharts unter Verwendung von GraphViz. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, May 2005.
- [4] Florian Lüpke. Implementierung eines Statechart-Editors mit layoutbasierten Bearbeitungshilfen. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, June 2005.
- [5] Mathworks Inc. *Simulink – Simulation and Model-Based Design*. Mathworks Inc., 2005. URL http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_using.pdf.
- [6] Mathworks Inc. Stateflow Application Programming Interface, 2005. <http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/>.
- [7] The MathWorks, Inc. *Stateflow and Stateflow Coder User's Guide, Version 6*. Natick, MA, September 2005. URL http://www.mathworks.com/access/helpdesk/help/pdf_doc/stateflow/sf_ug.pdf.
- [8] Object Management Group. OMG Unified Modeling Language Specification, Version 1.5, March 2003. <http://www.omg.org/cgi-bin/doc?formal/03-03-01>.
- [9] André Ohlhoff. Simulating the Behavior of Synccharts. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, November 2004.
- [10] Jan Täubrich. Untersuchung der nicht-interaktiven Simulation von Stateflow-Statecharts. Praktikumsbericht, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, April 2005.
- [11] The KIEL Project. Project Homepage, 2004. URL <http://www.informatik.uni-kiel.de/~rt-kiel/>. Kiel Integrated Environment for Layout.

5. Bibliography

- [12] The KIEL Project. Project API Documentation, 2004. URL <http://www.informatik.uni-kiel.de/~rt-kiel/kiel/kiel/doc/>. Kiel Integrated Environment for Layout.
- [13] The Mathworks, Inc. Company homepage. <http://www.mathworks.com>.
- [14] Mirko Wischer. Ein FileInterface für das KIEL Projekt. Praktikumsbericht, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, 2005.
- [15] Mirko Wischer. Ein Browser für die Visualisierung dynamischer Sichten von Statecharts. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, 2005.

A. Settings

This chapter describes the available settings for the *Stateflow* simulator and for the file interface plug-in for “.mdl” files.

MatlabProcessInfo.matlabExecutableFileName

This parameter specifies the file name of the *Matlab* executable file. It may also specify the complete path to the executable file.

matlabPrompt

This parameter specifies the matlab prompt. The standard setting is “>>”, but this may differ from version to version, especially for student versions.

matlabErrorIndicator

It specifies the string that *Matlab* outputs in error messages. Error messages are recognized by this string. The standard setting is “????”.

MatlabStateflowGrabber.chartScale

This factor is used to scale the chart during import. A factor greater than 1 may help to solve the problem of labels that do not fit into their states because of the font size. The standard setting is 1.

MatlabProcessInfo.nodisplay

This parameter specifies whether *Matlab*, which is started by *KIEL*, will display any windows. Possible values are “true” and “false”. The standard setting is “true”.

StateflowSimulator.inputVariableNamePostfix

This parameter specifies the string that is attached to “From Workspace” blocks for input variables. The standard setting is “VarMat”.

A. Settings

B. Java Code

This chapter introduces all implemented classes. It includes the complete program code of all extensions that have been made to the *KIEL* application.

B.1. Description

B.1.1. Package `kiel.fileInterface.matlab`

This package includes the plug-in for “mdl” files.

MatlabStateflowFileFilter: This class is used in the file dialog to filter files according to their file name extension. Only files with the file name extension “.mdl” are returned by this filter.

MatlabStateflow: This class contains the code that is used by the file interface to load and save “.mdl” files. It starts *Matlab* and uses the classes `MatlabStateflowGrabber` and `MatlabStateflowCreator` to do the real work of import and export.

B.1.2. Package `kiel.fileInterface.M`

This package includes the plug-in for “m” files. This plug-in supports only files that include simulation input data. It is used for the input trace feature.

MFileFilter: This class is used in the file dialog to filter files according to their file name extension. Only files with the file name extension “.m” are returned by this filter.

MFormat: This class contains the code that is used by the file interface to load and save “.m” files. It creates a vector variable for each input signal and each input variable with the corresponding values for each simulation step. It cannot handle all features of this file format. It can be used in conjunction with the input trace feature only.

B.1.3. Package `kiel.fileInterface.ESI`

This package includes the plug-in for “.esi” files. This file format is used in *Esterel Studio* for the input trace feature.

B. Java Code

ESIFileFilter: This class is used in the file dialog to filter files according to their file name extension. Only files with the file name extension “.esi” are returned by this filter.

ESIFormat: This class contains the code that is used by the file interface to load and save “.esi” files.

B.1.4. Package kiel.util

This package contains utility classes, which can be used by any other class of the *KIEL* application. Some of these classes are of special interest for this work.

MatlabStateflowGrabber: It uses the *Stateflow* API to collect all needed information about a chart in *Stateflow* and constructs an equivalent chart using the *KIEL* data structure.

MatlabStateflowCreator: It uses the *Stateflow* API to construct a chart in *Stateflow* that is equivalent to the chart currently loaded in *KIEL*.

MatlabProcessInfo: This class holds a reference to the *Matlab* process and its input and output stream. It is responsible for the start and termination of *Matlab*.

MatlabStateflowProperties: It is used to load and to store settings of components that are related to *Stateflow*.

B.1.5. Package kiel.simulator.matlab

This class contains the simulator for *Stateflow* charts.

StateflowSimulator: This class implements the simulator for *Stateflow* charts. The simulator uses *Matlab/Simulink/Stateflow* to do the simulation.

SimulatorException: This exception is thrown by the stateflow simulator in case an error occurs with the input or output streams or *Matlab* reports an error.

B.2. kiel.fileInterface.matlab

B.2.1. MatlabStateflowFileFilter

```

//$Id: MatlabStateflowFileFilter.java,v 1.1 2005/06/03 12:41:24 apo_Exp $ 
/*
 * (non-Javadoc)
 * @see javax.swing.filechooser.FileFilter#accept(javax.io.File)
 */
public boolean accept(File f) {
    if(f.isDirectory())
        return true;
    return (
        f.canRead() && f.isFile()
        && f.getPath().toLowerCase().endsWith(".mdl"));
}

/*
 * (non-Javadoc)
 * @see javax.swing.filechooser.FileFilter#getDescription()
 */
public String getDescription() {
    return "Matlab/Simulink/Stateflow";
}

/*
 * (non-Javadoc)
 * @see javax.swing.filechooser.FileFilter#getDescription()
 */
public String getExtension() {
    return ".mdl";
}

```

20

B.2.2. MatlabStateflow

```

    throw new FileInterfaceException(input);
}

/*
 * Reads the specified file and returns the chart constructed with
 * the KIEL datastructure.
 * @param f - the file to be read.
 * @return the statechart that is contained in the file.
 * @throws FileInterfaceException if an error is encountered
 * by Matlab.
 */
120 public final View getReadView() throws FileInterfaceException {
    return MatlabStateflowGrabber.getView();
}

/*
 * Writes the chart specified by the parameters <code>s</code> and <code>
 * </code> into the file specified by <code>f</code>.
 * @param s - the chart to be saved. Only structural information is given
 * here.
 * @param v - the corresponding view. Only graphical information is given
 * here.
 * @param f - the file name of the file that has to be saved into.
 * @return the file name of the file that has been stored. May differ
 * in the file extension from the file name specified by the parameter.
 */
130 public final StateChart readatChartDocument(final File f)
    throws FileInterfaceException {
    loadBench.start();
    loadBench.start();
    try {
        if (MatlabProcessInfo.matlabProcess == null) {
            startMatlabBench.start();
            log.log(0, "Starting Matlab ...");
            input = MatlabProcessInfo.startMatlab();
            startMatlabBench.stop();
            startMatlabBench.finishBenchmark();
            log.log(0, input);
        }
    } catch (Exception ex) {
        MatlabProcessInfo.toMatlabStream.write("sfexit");
        MatlabProcessInfo.toMatlabStream.newLine();
        MatlabProcessInfo.toMatlabStream.flush();
        MatlabProcessInfo.fromMatlabStream.skip(3);
        log.log(0, "Telling Matlab to load " + f.getAbsolutePath());
        MatlabProcessInfo.toMatlabStream.write(
            "open -system '" + f.getAbsolutePath() + "'\"");
        MatlabProcessInfo.toMatlabStream.newLine();
        MatlabProcessInfo.toMatlabStream.flush();
        checkMatlabResponse();
        log.log(0, input);
        log.log(0, "Done");
        MatlabStateflowGrabber.currentFileName = f.getName();
        loadBench.stop();
        loadBench.finishBenchmark();
        return MatlabStateflowGrabber.grabStateflow();
    } catch (IOException e) {
        MatlabStateflowGrabber.closeLogFile();
        throw new FileInterfaceException("An error occurred while"
            + " starting Matlab or communicating with it :" + e);
    } catch (MatlabStateflowGrabberException me) {
        MatlabStateflowGrabber.closeLogFile();
        throw new FileInterfaceException("MatlabStateflowGrabber:" +
            + " Internal Error : " + me);
    }
}

/*
 * Determines whether this plug-in can write files.
 * @return always <code>true</code>
 */
140 public final void writeStateChartDocument(final StateChart s,
    final File f) {
    loadBench.printOutputToFile(new File("SaveBench" + f.getName() + ".txt"));
    loadBench.start();
    try {
        MatlabStateflowCreator.createChart(s, v);
        MatlabProcessInfo.toMatlabStream.write("ssave('untitled', 'u' ");
        MatlabProcessInfo.toMatlabStream.write(" + f.getAbsolutePath() + ".mdl");
        MatlabProcessInfo.toMatlabStream.newLine();
        MatlabProcessInfo.toMatlabStream.flush();
        checkMatlabResponse();
    } catch (Exception ex) {
        MatlabStateflowCreator.closeLogFile();
        log.log(0, "An error occurred while saving the chart : " +
            + ex.getMessage());
    }
    if (!f.getAbsolutePath().endsWith(".mdl")) {
        return new File(f.getAbsolutePath() + ".mdl");
    }
    saveBench.stop();
    saveBench.finishBenchmark();
    return f;
}

/*
 * Returns an instance of the file filter for 'mdl' files.
 * @return the file filter.
 */
150 public final FileFilter getStateChartFileFilter() {
    return MatlabStateflowFileFilter.getInstance();
}

/*
 * Determines whether this plug-in can write files.
 * @return always <code>true</code>
 */
160 public final void writeNewFile() {
    MatlabStateflowGrabber.closeLogFile();
    throw new FileInterfaceException("An error occurred while"
        + " starting Matlab or communicating with it :" + e);
}
}

/*
 * Returns an instance of the file filter for 'mdl' files.
 * @return the file filter.
 */
170 public final void writeNewFile() {
    MatlabStateflowGrabber.closeLogFile();
    throw new FileInterfaceException("An error occurred while"
        + " starting Matlab or communicating with it :" + e);
}
}

```

B. Java Code

```
/*
 * Returns the name of this plug-in.
 */
public final String getName() {
    return "Matlab/Stateflow Plug-in";
}

/*
 * Determines whether this plug-in can read files.
 * @return always <code>true</code>.
 */
public final boolean canRead() {
    return true;
}

/*
 * Returns the name of this plug-in.
 */
public final String getName() {
    return "Matlab/Stateflow Plug-in";
}
```

B.3. kiel.fileInterface.M

B.3.1. MFileFilter

```
//$Id:  
package kiel.fileInterface.M;  
  
import java.io.File;  
  
import javax.swing.filechooser.FileFilter;  
  
/**  
 * @author apo  
 * @version $Revision: 1.1 $ last modified $Date: 2005/08/26 09:23:02 $  
*/  
public class MFileFilter extends FileFilter {  
    /**An instance counter.*/  
    private static MFileFilter instance = null;  
    /**Ensures that only one instance exists.*/  
    public static MFileFilter getInstance(){  
        if(instance == null){  
            instance = new MFileFilter();  
        }  
        return instance;  
    }  
    public boolean accept(File f) {  
        if(f.isDirectory()){  
            return true;  
        }  
        return (  
            f.canRead() && f.isFile()  
            && f.getPath().toLowerCase().endsWith(".m"));  
    }  
    public String getDescription(){  
        return new String("Matlab\u2014Trace\u2014File");  
    }  
}
```

B.3.2. MFileFormat

```

//$$Id: MFileFormat.java 1.2 $ Last modified $Date: 2005/12/15 13:31:12 $
/*
 * Author: apo
 * Version $Revision: 1.2 $
 */
public final class MFileFormat extends TraceFileInterface {
    /**
     * Input variables and input signals.
     */
    private ArrayList inputs;
    /**
     * The string that is currently processed.
     */
    private String currentLine;
    /**
     * The object that is currently added to the trace.
     */
    private Object inputObject;
    /**
     * Reads the file specified by the parameter <code>f</code> and creates
     * a <code>TraceData</code> object according to the contents of the file.
     * @param f - the file to be read.
     * @return the trace that is stored in the file.
     * @throws FileInterfaceException if an error occurs while opening the
     * file.
     */
    public TraceData readTraceFile(final File f)
        throws FileInterfaceException {
        TraceData trace = new TraceData();
        TraceStep step;
    }
}

try {
    FileReader datareader = new FileReader(f);
    LineNumberReader datainput = new LineNumberReader(datareader);
    currentLine = datainput.readLine();
    String[] tokenizedLine;
    int stepNumber;
    import javax.swing.filechooser.FileFilter;
    import kiel.configmgr.SignalValue;
    import kiel.configmgr.VariableValue;
    import kiel.datastructure.Variable;
    import kiel.datastructure.eventexp.Event;
    import kiel.datastructure.eventexp.IntegerSignal;
    import kiel.datastructure.eventexp.Signal;
    import kiel.fileInterface.FileInterfaceException;
    import kiel.fileInterface.TraceFileInterface;
    import kiel.simulationTrace.TraceData;
    import kiel.simulationTrace.TraceStep;
    /**
     * @author apo
     * @version $Revision: 1.2 $
     */
    public final class MFileFormat extends TraceFileInterface {
        /**
         * Input variables and input signals.
         */
        private ArrayList inputs;
        /**
         * The string that is currently processed.
         */
        private String currentLine;
        /**
         * The object that is currently added to the trace.
         */
        private Object inputObject;
        /**
         * Reads the file specified by the parameter <code>f</code> and creates
         * a <code>TraceData</code> object according to the contents of the file.
         * @param f - the file to be read.
         * @return the trace that is stored in the file.
         * @throws FileInterfaceException if an error occurs while opening the
         * file.
         */
        public TraceData readTraceFile(final File f)
            throws FileInterfaceException {
            TraceData trace = new TraceData();
            TraceStep step;
        }
    }
}

```

B. Java Code

B. Java Code

```

    }
    if (inputs.get(i). instanceof Variable) {
        String name = ((Variable) varval[i]).getObjec()
            .getname();
        if (nameList.get(i).equals(name)) {
            String val = varval[i].getStringValue();
            recentValue = val;
            dataoutput.write(stepNumber + "\u00a0" + val);
            written = true;
            break;
        }
    }
    if (!written) {
        dataoutput.write(stepNumber + "\u00a0" + recentValue);
        stepNumber++;
    }
    dataoutput.write("\n");
    dataoutput.flush();
    dataoutput.close();
    return f;
} catch (IOException io) {
    throw new FileInterfaceException(io.getMessage());
}
}

240 /**
 * Returns an instance of the M file filter.
 * @return an instance of the M file filter.
 */
public FileFilter getTraceFileFilter() {
    return MFileFilter.getInstance();
}

250 /**
 * Sets the input signals and input variables that are used to construct
 * the <code>TraceData</code> object.
 * @param in - the input variables and input signals that appear in the
 * trace file. Inputs that are not specified here will not be build into
 * the trace!/
*/
public void setInputs(final ArrayList in) {
    inputs = in;
}

260 /**
 * Searches for the object that is referenced by the name in the trace
 * file and writes this into the global variable <code>inputObject</code>.
 */
private void searchForInputObject() {
    for (int i = 0; i < inputs.size(); i++) {
        if (inputs.get(i). instanceof Event) {
            getname();
            if (currentLine.startsWith(((Event) inputs.get(i))
                .getname())) {
                inputObject = inputs.get(i);
            } else {
                inputObject = null;
            }
        }
    }
}

270 /**
 * Searches for the object that is referenced by the name in the trace
 * file and writes this into the global variable <code>inputObject</code>.
 */
private void searchForInputObject() {
    for (int i = 0; i < inputs.size(); i++) {
        if (inputs.get(i). instanceof Event) {
            getname();
            if (currentLine.startsWith(((Event) inputs.get(i))
                .getname())) {
                inputObject = inputs.get(i);
            } else {
                inputObject = null;
            }
        }
    }
}

280 /**
 * Searches for the object that is referenced by the name in the trace
 * file and writes this into the global variable <code>inputObject</code>.
 */
private void searchForInputObject() {
    for (int i = 0; i < inputs.size(); i++) {
        if (inputs.get(i). instanceof Event) {
            getname();
            if (currentLine.startsWith(((Event) inputs.get(i))
                .getname())) {
                inputObject = inputs.get(i);
            } else {
                inputObject = null;
            }
        }
    }
}

290 /**
 * Gets the initial value of the variable specified by <code>obj</code>.
 * @param obj - the variable whose initial value is of interest.
 * @return the initial value of the variable specified by the parameter.
 */
private String getInitialValue(final String obj) {
    Iterator it = inputs.iterator();
    Object next;
    while (it.hasNext()) {
        next = it.next();
        if (next instanceof Variable) {
            if (((Variable) next).getName().equals(obj)) {
                return ((Variable) next).getValue();
            }
        }
    }
    return null;
}

300 /**
 * Generates a list of names of objects referenced in the trace.
 * @param td - the trace for which the names list to be created.
 * @return a list of all names that appear in the trace.
 */
private ArrayList generateNamesList(final TraceData td) {
    TraceStep step;
    ArrayList namesList = new ArrayList();
    td.play();
    td.startTrace();
    while (td.hasNextStep()) {
        step = td.getNextStep();
        VariableValue[] varval = step.getVariableAssignments();
        SignalValue[] signal = step.getSignalAssignments();
        for (int i = 0; i < signal.length; i++) {
            if (!namesList.contains((Signal) signal[i].getObject()))
                .getname());
            namesList.add((Signal) signal[i].getObject());
                .getname());
        }
    }
    return namesList;
}

310 /**
 * Generates a list of names of objects referenced in the trace.
 * @param td - the trace for which the names list to be created.
 * @return a list of all names that appear in the trace.
 */
private ArrayList generateNamesList(final TraceData td) {
    TraceStep step;
    ArrayList namesList = new ArrayList();
    while (td.hasNextStep()) {
        step = td.getNextStep();
        VariableValue[] varval = step.getVariableAssignments();
        SignalValue[] signal = step.getSignalAssignments();
        for (int i = 0; i < signal.length; i++) {
            if (!namesList.contains((Signal) signal[i].getObject()))
                .getname());
            namesList.add((Signal) signal[i].getObject());
                .getname());
        }
    }
    return namesList;
}

320 /**
 * Gets the initial value of the variable specified by <code>obj</code>.
 * @param obj - the variable whose initial value is of interest.
 * @return the initial value of the variable specified by the parameter.
 */
private String getInitialValue(final String obj) {
    Iterator it = inputs.iterator();
    Object next;
    while (it.hasNext()) {
        next = it.next();
        if (next instanceof Variable) {
            if (((Variable) next).getName().equals(obj)) {
                return ((Variable) next).getValue();
            }
        }
    }
    return null;
}

330 /**
 * Sets the input signals and input variables that are used to construct
 * the <code>TraceData</code> object.
 * @param in - the input variables and input signals that appear in the
 * trace file. Inputs that are not specified here will not be build into
 * the trace!/
*/
public void setInputs(final ArrayList in) {
    inputs = in;
}

340 /**
 * Gets the initial value of the variable specified by <code>obj</code>.
 * @param obj - the variable whose initial value is of interest.
 * @return the initial value of the variable specified by the parameter.
 */
private String getInitialValue(final String obj) {
    Iterator it = inputs.iterator();
    Object next;
    while (it.hasNext()) {
        next = it.next();
        if (next instanceof Variable) {
            if (((Variable) next).getName().equals(obj)) {
                return ((Variable) next).getValue();
            }
        }
    }
    return null;
}

```


B.4. kiel.fileInterface.ESI

B.4.1. ESIFileFilter

```

//$Id:
package kiel.fileInterface.ESI;

import java.io.File;
import javax.swing.filechooser.FileFilter;

/*
 * @author apo
 * @version $Revision: 1.1 $ last modified $Date: 2005/08/11 12:41:40 $
 */
public class ESIFileFilter extends FileFilter {

    /**
     * An instance counter.*/
    private static ESIFileFilter instance = null;
    /**
     * Ensures that only one instance exists.*/
    public static ESIFileFilter getInstance(){
        if(instance == null){
            instance = new ESIFileFilter();
        }
        return instance;
    }

    /**
     * Returns true if the file is an ESI trace file.
     */
    public boolean accept(File f) {
        if(f.isDirectory()){
            return true;
        }
        return (
            f.canRead() && f.isFile()
            && f.getPath().toLowerCase().endsWith(".esi"));
    }

    /**
     * Returns the description of the filter.
     */
    public String getDescription() {
        return new String("ESI Trace File");
    }
}

```

B.4.2. ESIFileFormat

```

//$Id:
package kiel.fileInterface.ESI;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.LineNumberReader;
import java.util.ArrayList;
import javax.swing.filechooser.FileFilter;
import kiel.configMgr.SignalValue;
import kiel.configMgr.VariableValue;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.eventexp.Signal;
import kiel.fileInterface.FileInterfaceException;
import kiel.fileInterface.TraceFileNotFoundException;
import kiel.simulationTrace.TraceData;
import kiel.simulationTrace.TraceStep;
import kiel.simulationTrace.TraceTrace;

/*
 * @author apo
 * @version $Revision: 1.4 $ last modified $Date: 2005/12/15 13:31:12 $
 */
public final class ESIFileFormat extends TraceFileInterface {
    /**
     * Input variables and input signals.
     */
    private ArrayList inputs;
    private String currentLine;
    private Object inputObject;
    /**
     * The string that is currently added to the trace.
     */
    /**
     * The object that is currently processed.
     */
    /**
     * Reads the file specified by the parameter <code>f</code> and creates
     * a <code>TraceData</code> object according to the contents of the file.
     * Param f - the file to be read.
     * Returns the trace that is stored in the file.
     * Throws FileInterfaceException if an error occurs while opening the
     * file.
     */
    public TraceData readTraceFile(final File f)
        throws FileInterfaceException {
        TraceData trace = new TraceData();
        TraceStep step;
    }
}

try {
    FileReader datareader = new FileReader(f);
    LineNumberReader datainput = new LineNumberReader(datareader);
    currentLine = datainput.readLine();
    step = new TraceStep();
    while (currentLine != null) {
        while (!currentLine.trim().endsWith(";""))
            if ((currentLine.trim().equals("")) ||
                ((currentLine.startsWith("\n")))) {
                searchForInputObject();
                if (inputObject != null) {
                    if (inputObject.getClass().equals(
                        IntegerSignal.class)) {
                        if (currentLine.indexOf('==' ) == -1) {
                            step.add(new SignalValue(
                                (IntegerSignal)inputObject,
                                new Integer(currentLine.substring(
                                    currentLine.indexOf('==' )
                                    + 1).trim()))));
                        } else {
                            step.add(new SignalValue(
                                (IntegerSignal)inputObject,
                                null));
                        }
                    } else if (inputObject.getClass().equals(
                        Signal.class)) {
                        step.add((Signal)inputObject);
                    } else if (inputObject instanceof Variable) {
                        if (currentLine.indexOf('==' ) == -1) {
                            step.add(new VariableValue(
                                (Variable)inputObject,
                                (currentLine.substring(
                                    currentLine.indexOf('==' )
                                    + 1).trim())));
                        } else {
                            step.add((Event)inputObject);
                        }
                    } else {
                        currentLine = datainput.readLine();
                    }
                }
            }
        /**
         * Reads the file specified by the parameter <code>f</code> and creates
         * a <code>TraceData</code> object according to the contents of the file.
         * Param f - the file to be read.
         * Returns the trace that is stored in the file.
         * Throws FileInterfaceException if an error occurs while opening the
         * file.
         */
        public TraceData readTraceFile(final File f)
            throws FileInterfaceException {
            TraceData trace = new TraceData();
            TraceStep step;
        }
    }
} catch (IOException io) {
    throw new FileInterfaceException(io.getMessage());
}
return trace;
}

```

B. Java Code

```

    /*
     * Writes the trace specified by the parameter <code>td</code> into the
     * file specified by <code>f</code> in the ESI file format.
     * Param td - the trace to be saved.
     * Param f - the file name to save the trace into.
     * Returns the file name the trace has been written into.
     * Throws FileInterfaceException if an error occurs while opening the
     * file.
     */
    public File writeTraceFile(final Tracedata td, final File f)
        throws FileInterfaceException {
        try {
            TraceStep step;
            FileWriter dataoutput = new FileWriter(f);
            dataoutput.write("%_This_file_was_automatically_generated_by"
                + " _ESIfileFormat.writeTraceFile(\n\"");
            180
            td.play();
            td.startTrace();
            int stepNumber = 1;
            while (td.hasNextStep()) {
                dataoutput.write("%_STEP_" + stepNumber + "\n");
                step = td.getNextStep();
                Event[] ev = step.getEvents();
                Signal[] sg = step.getSignals();
                IntegerSignal[] intsg = step.getValuedSignals();
                VariableValue[] varval = step.getVariableAssignments();
                SignalValue[] signal = step.getSignalAssignments();
                for (int i = 0; i < ev.length; i++) {
                    dataoutput.write(ev[i].getName());
                    dataoutput.write("\n");
                    dataoutput.flush();
                }
                for (int i = 0; i < sg.length; i++) {
                    dataoutput.write(sg[i].getName());
                    dataoutput.write("\n");
                    dataoutput.flush();
                }
                for (int i = 0; i < intsg.length; i++) {
                    dataoutput.write(intsg[i].getName());
                    dataoutput.write("\n");
                    dataoutput.flush();
                }
                190
                for (int i = 0; i < varval.length; i++) {
                    Integer val = varval[i].getVal();
                    dataoutput.write((Signal) varval[i].getObject()
                        .getName());
                    dataoutput.write("\n");
                    dataoutput.write(val.toString());
                    dataoutput.write("\n");
                    dataoutput.flush();
                }
                200
            }
            210
            220
        }
        /*
         * Sets the input signals and input variables that are used to construct
         * the <code>TraceData</code> object.
         * Returns an instance of the ESI file filter.
         * @param in - the input variables and input signals that appear in the
         * trace file. Inputs that are not specified here will not be build into
         * the trace!
         */
        public void setInputs(final ArrayList in) {
            inputs = in;
        }
        /*
         * Searches for the object that is referenced by the name in the trace
         * file and writes this into the global variable <code>inputObject</code>.
         */
        private void searchForInputObject() {
            for (int i = 0; i < inputs.size(); i++) {
                if (inputs.get(i) instanceof Event) {
                    inputObject = ((Event) inputs.get(i)).getName();
                }
            }
        }
        140
        150
        160
        170
    }

```

```
        } else {
            inputObject = null;
        }
    }
}
```

B.5. kiel.util

B.5.1. MatlabStateflowGrabber

```

//$Id: MatlabStateflowGrabber.java,v 1.24 2005/12/12 16:58:56 spr Exp $
package kiel.util;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
10 import java.util.HashMap;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.List;
import java.util.ArrayList;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Constant;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.History;
20 import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Priority;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
30 import kiel.dataStructure.StringLabel;
import kiel.dataStructure.StringVariable;
import kiel.dataStructure.Transition;
import kiel.dataStructure.TransitionLabel;
import kiel.dataStructure.Variable;
import kiel.dataStructure.action.ActionsStringLabel;
import kiel.dataStructure.booleanp.BooleanExpression;
import kiel.dataStructure.booleanp.BooleanFalse;
import kiel.dataStructure.booleanp.BooleanTrue;
40 import kiel.dataStructure.booleanp.BooleanVariable;
import kiel.dataStructure.doublexp.DoubleConstant;
import kiel.dataStructure.doublexp.DoubleVariable;
import kiel.dataStructure.eventxp.Event;
import kiel.dataStructure.eventxp.IntegerSignal;
import kiel.dataStructure.floatxp.FloatConstant;
import kiel.dataStructure.floatxp.FloatVariable;
import kiel.dataStructure.int16xp.Integer16Constant;
50 import kiel.dataStructure.int8xp.Integer8Constant;
import kiel.dataStructure.intexp.IntegerConstant;
import kiel.dataStructure.intexp.IntegerVariable;

```

import kiel.dataStructure.uint16xp.UInt16Constant;
 import kiel.dataStructure.uint16xp.UInt16Variable;
 import kiel.dataStructure.uint32xp.UInt32Constant;
 import kiel.dataStructure.uint32xp.UInt32Variable;
 import kiel.dataStructure.uint8xp.UInt8Variable;
 import kiel.graphicalInformations.CompositeLayoutInformation;
 import kiel.graphicalInformations.DelimiterLayoutInformation;
 import kiel.graphicalInformations.EdgeLayoutInformation;
 import kiel.graphicalInformations.LabelLayoutInformation;
 import kiel.graphicalInformations.MoveToPath;
 import kiel.graphicalInformations.NodeLayoutInformation;
 import kiel.graphicalInformations.Point;
 import kiel.graphicalInformations.Properties;
 import kiel.graphicalInformations.QuadraticCurveToPath;
 import kiel.graphicalInformations.View;

/** * @author Adrian Posor (apo)

70 * @version \$Revision: 1.24 \$ last modified \$Date: 2005/12/12 16:58:56 \$

*/

* @author Adrian Posor (apo)

* @version \$Revision: 1.24 \$ last modified \$Date: 2005/12/12 16:58:56 \$

*/

* The file name of the currently processed chart.

*/

* The name of the currently processed chart.

*/

* The hash map used to map between created objects of the data structure
 * and objects in the Stateflow chart.

*/

* The hash map used to map between created objects of the data structure
 * and objects in the Stateflow chart.

*/

* The string that indicates an error in Matlab, usually '???'.

*/

* The Matlab prompt, usually '>>'.

*/

* The string that indicates an error in Matlab, usually '???'.

*/

* The buffered reader used to read Matlab's output.

*/

* The buffered reader used to read Matlab's output.

*/

* The string matlabErrorIndicator;

*/

* The string matlabPrompt;

*/

* The string that indicates an error in Matlab, usually '???'.

*/

* The Matlab prompt, usually '>>'.

*/

* The string matlabErrorIndicator;

*/

* The buffered reader used to read Matlab's output.

*/

* The buffered reader fromMatlab;

*/

private static BufferedReader bufferedReader;

```

    /**
     * The buffered writer used to send commands to Matlab.
     */
    private static BufferedWriter toMatlab;

    /**
     * The view which includes the graphical information (layout) of the
     * currently processed chart.
     */
    private static View actualView;

    /**
     * The coordinates of the chart's boundary.
     */
    private static int[] chartBoundary = new int[2];

    /**
     * The factor used to scale the chart during import.
     */
    private static float chartScale = MatlabStateflowProperties.getChartScale();

    /**
     * The last answer of Matlab.
     */
    private static String input = "";

    /**
     * The file the log is written into.
     */
    private static LogFile log = null;

    /**
     * A flag that indicates whether the currently processed chart contains
     * any subcharts.
     */
    private static boolean containsSubcharts = false;

    /**
     * The command to obtain the Stateflow root object.
     */
    private static final String GET_ROOT_COMMAND = "rt=sfroot";

    /**
     * The command to obtain the Simulink Model.
     */
    private static final String GET_MODELS_COMMAND =
        "model=rt.find('-.isa','Simulink.BlockDiagram')";

    /**
     * The command to obtain the Stateflow chart.
     */
    private static final String GET_CHART_COMMAND =
        "chart=model.find('-.isa','Stateflow.Chart')";

    /**
     * The command to obtain states which are direct children of their parent.
     */
    private static final String GET_STATES_COMMAND =
        ".find('-.isa','Stateflow.State',-depth',1)";

    /**
     * The command to obtain transitions which are direct children of their
     * parent state.
     */
    private static final String GET_TRANSITIONS_COMMAND =
        ".find('-.isa','Stateflow.Transition',-depth',1)";

    /**
     * The command to obtain junctions which are direct children of their
     * parent state.
     */
    private static final String GET_JUNCTIONS_COMMAND =
        ".find('-.isa','Stateflow.Junction',-depth',1)";

    /**
     * The command to obtain local events.
     */
    private static final String GET_EVENTS_COMMAND =
        ".find('-.isa','Stateflow.Event',-depth',1)";

    /**
     * The command to obtain input events.
     */
    private static final String GET_INPUTEVENTS_COMMAND =
        ".find('-.isa','Stateflow.Event',-depth',1,'Scope','Input')";

    /**
     * The command to obtain output events.
     */
    private static final String GET_OUTPUTEVENTS_COMMAND =
        ".find('-.isa','Stateflow.Event',-depth',1,'Scope','Output')";

    /**
     * The command to obtain local events.
     */
    private static final String GET_LOCALEVENTS_COMMAND =
        ".find('-.isa','Stateflow.Event',-depth',1,'Scope','Local')";

    /**
     * The command to obtain variables.
     */
    private static final String GET_VARIABLES_COMMAND =
        ".find('-.isa','Stateflow.Data',-depth',1)";

    /**
     * The command to obtain the editor object of the currently
     * processed chart.
     */
    private static final String GET_EDITOR_COMMAND = "editor=chart.editor";
}

private static final String INITIAL_VALUE = ".props.initialValue";

/**
 * The suffix to obtain the initial value of a variable.
 */
private static final String DATA_TYPE = ".DataType";

private static final String DTYPE = ".Type";

```

B. Java Code

```

private static final String DEST = ".Destination";
    /**
     * The suffix to obtain the identifier of a Stateflow object.
    */
    private static final String ID = ".Id";
```

230

```

    /**
     * The suffix to obtain the name of a Stateflow object.
    */
    private static final String NAME = ".Name";
```

```

    /**
     * The suffix to obtain the label of a state of transition.
    */
    private static final String LABEL = ".LabelString";
```

240

```

    /**
     * The suffix to obtain the position of a transition label.
    */
    private static final String LABEL_POS = ".LabelPosition";
```

300

```

    /**
     * The suffix to obtain the position of a graphical Stateflow object.
    */
    private static final String POSITION = ".Position";
```

250

```

    /**
     * The suffix to obtain the center coordinates of a junction.
    */
    private static final String JUNC_POS = ".Position.Center";
```

310

```

    /**
     * The suffix to obtain the radius of a junction.
    */
    private static final String JUNC_RADIUS = ".Position.Radius";
```

260

```

    /**
     * The suffix to obtain the midpoint of a transition.
    */
    private static final String TR_MIDPOINT = ".MidPoint";
```

270

```

    /**
     * The suffix to obtain the position and size of
     * the window that contains the chart.
    */
    private static final String CHART_SIZE = ".WindowPosition";
```

330

```

    /**
     * The suffix to obtain the source end point of a transition.
    */
    private static final String SRC_EP = ".SourceEndPoint";
```

280

```

    /**
     * The suffix to obtain the destination of a transition.
    */
    private static final String SRC = ".Source";
```

340

```

    /**
     * Size of the input buffer (for receiving Matlab's answer).
    */
    private static final int BUFF_SIZE = 20;
```

```

    /**
     * Utility classes should not have a public or default constructor.
    */

```

```

private MatlabStateflowGrabber() {
    /* by Matlab.
     */
}

/** Receives output from Matlab. This method uses a buffered reader
 * that is connected to Matlab's standard output. The stream is read
 * until a Matlab prompt is encountered. Matlab's answer is stored
 * in the global variable <code>input</code>. If an error is indicated
 * by Matlab, then an exception is thrown.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
private static void checkMatlabResponse() throws IOException {
    410
    MatlabStateflowGrabberException answer = null;
    String input = null;
    char[] inputBuf = new char[BUFF_SIZE];
    int numRead = 0;

    while (!input.endsWith(matlabPrompt)) {
        numRead = fromMatlab.read(inputBuf, 0, inputBuf.length);
        if (numRead > 0) {
            input += new String(inputBuf, 0, numRead);
        }
    }

    log.log(0, input);
    if (input.indexOf(matlabErrorIndicator) != -1) {
        closeLogFile();
        throw new MatlabStateflowGrabberException(input);
    }
}

/** Sends a command to Matlab and receives Matlab's response. This method
 * calls <code>checkMatlabResponse</code> to receive Matlab's answer to
 * the command.
 * @param command - the command that has to be sent to Matlab.
 * @return Matlab's answer to the command.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
350
private static String[] handleMatlabCommand(final String command)
throws IOException, MatlabStateflowGrabberException {
    440
    log.log(0, "HandleMatlabCommand:" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    return input.trim().split("\\s+");
}

/** Obtains the size of variable (array) <code>s</code> in the
 * Matlab workspace.
 * @param s - the variable (array) in the Matlab workspace one wants
 * to know the size of.
 * @return the size of the variable, that is the number of elements
 * in the array.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
360
private static int getArraySizeOf(final String s) throws IOException {
    480
    MatlabStateflowGrabberException answer = null;
    String[] answer = handleMatlabCommand("size(" + s + ")");
    return Integer.parseInt(answer[2]);
}

/** Obtains the number of transitions that have been previously
 * obtained.
 * @return the number of previously obtained transitions.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
370
private static int getNumberOfTransitions() throws IOException {
    490
    MatlabStateflowGrabberException answer = null;
    String[] answer = handleMatlabCommand("transitions.size");
    return Integer.parseInt(answer[2]);
}

/** Obtains the number of junctions that have been previously
 * obtained.
 * @return the number of junctions previously obtained.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
380
private static int getNumberOfJunctions() throws IOException {
    500
    MatlabStateflowGrabberException answer = null;
    String[] answer = handleMatlabCommand("junctions.size");
    return Integer.parseInt(answer[2]);
}

/** Obtains the data type of the variable specified by
 * the <code>var</code> argument.
 * @param var - the variable one wants to know the data type of
 * @return the data type of the variable (int32, int16, int8,
 * single, double, boolean, unsigned integer (32, 16, 8 bit).
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
390
private static String getDataType(final String var) throws IOException {
    510
    MatlabStateflowGrabberException answer = null;
    String[] answer = handleMatlabCommand(var + "DTYPE");
    return answer[2];
}

/** Obtains the scope of the variable specified by
 * the <code>var</code> argument.
 * @param var - the variable one wants to know the scope of.
 * @return the scope of the variable (input, output, local).
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabberException if an error is encountered
 * by Matlab.
 */
400
460

```

B. Java Code

```

/*
 * retrieves the coordinates of the label of the object obj.
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
private static int[] getObjectLabelPosition(final String obj)
throws IOException, MatlabStateflowGrabbberException {
    String[] answer = handleMatlabCommand(obj + LABEL_POS);
    int[] pos = new int[4];
    if (answer.length > 7) {
        float scale = Float.parseFloat(answer[2]);
        pos[0] = (int) (Float.parseFloat(answer[4]) * scale * chartScale);
        pos[1] = (int) (Float.parseFloat(answer[5]) * scale * chartScale);
        pos[2] = (int) (Float.parseFloat(answer[6]) * scale * chartScale);
        pos[3] = (int) (Float.parseFloat(answer[7]) * scale * chartScale);
    }
    return pos;
}

/*
 * obtains the name of the Stateflow object specified by
 * the <code><obj></code> argument.
 * param obj - the object one wants to know the name of
 * returns the name of the object.
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
private static String getObjectName(final String obj) throws IOException,
530
    MatlabStateflowGrabbberException {
    log.log(0, "GetObjectName:" + obj + NAME);
    toMatlab.write(obj + NAME);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    String[] answer = input.trim().split("\n+");
    return answer[1];
}

/*
 * obtains the position of the object specified by
 * the argument <code><obj></code>.
 * param obj - the object of which one wants to know the position.
 * returns the coordinates of the object [x, y, width, height].
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
private static int[] getObjectPosition(final String obj) throws IOException,
540
    MatlabStateflowGrabbberException {
    String[] answer = handleMatlabCommand(obj + POSITION);
    int[] pos = new int[4];
    if (answer.length > 7) {
        float scale = Float.parseFloat(answer[2]);
        pos[0] = (int) (Float.parseFloat(answer[4]) * scale * chartScale);
        pos[1] = (int) (Float.parseFloat(answer[5]) * scale * chartScale);
        pos[2] = (int) (Float.parseFloat(answer[6]) * scale * chartScale);
        pos[3] = (int) (Float.parseFloat(answer[7]) * scale * chartScale);
    }
    return pos;
}

/*
 * obtains the position of the object specified by
 * the argument <code><obj></code>.
 * param obj - the state or transition one wants to get the label of.
 * returns the label of obj.
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
private static String getObjectTypeLabel(final String obj) throws IOException,
550
    MatlabStateflowGrabbberException {
    log.log(0, "GetObjectLabel:" + obj + LABEL);
    toMatlab.write(obj + LABEL);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    String[] answer = input.trim().split("\n+");
    if (answer[1].equals("?")) {
        return new String();
    }
    return answer[1];
}

/*
 * obtains the position of the label of the object specified by
 * the <code><obj></code> argument.
 * param obj - the object (state or transition) for which one
 * wants to know the label position [x, y, width, height].
 */
560
private static String getTransitionLabel(final String obj) throws IOException,
    MatlabStateflowGrabbberException {
    log.log(0, "GetObjectLabel:" + obj + LABEL);
    toMatlab.write(obj + LABEL);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    String[] answer = input.trim().split("\n+");
    if (answer[1].equals("?")) {
        return new String();
    }
    return answer[1];
}

/*
 * obtains the position of the label of the object specified by
 * the argument <code><obj></code>.
 * param obj - the state or transition one wants to get the label of.
 * returns the label of obj.
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
570
private static String getBoundaryLabel(final String obj) throws IOException,
    MatlabStateflowGrabbberException {
    String[] answer = handleMatlabCommand(obj + BOUNDARY);
    int[] pos = new int[4];
    if (chartBoundary[0] < (pos[0] + pos[2])) {
        chartBoundary[0] = (pos[0] + pos[2]);
    }
    if (chartBoundary[1] < (pos[1] + pos[3])) {
        chartBoundary[1] = (pos[1] + pos[3]);
    }
    return pos;
}

/*
 * obtains the position of the label of the object specified by
 * the <code><obj></code> argument.
 * param obj - the object (state or transition) for which one
 * wants to know the label position [x, y, width, height].
 */
580
private static int[] getBoundaryPosition(final String obj) throws IOException,
    MatlabStateflowGrabbberException {
    String[] answer = handleMatlabCommand(obj + POSITION);
    int[] pos = new int[4];
    if (chartBoundary[0] < (pos[0] + pos[2])) {
        chartBoundary[0] = (pos[0] + pos[2]);
    }
    if (chartBoundary[1] < (pos[1] + pos[3])) {
        chartBoundary[1] = (pos[1] + pos[3]);
    }
    return pos;
}

```

```

    /**
     * Obtains the size of the currently processed chart.
     * Returns the size of the currently processed chart [width, height].
     * Throws IOException if an error occurs in the input or output stream.
     * by Matlab.
     */
    private static int[] getChartSize() throws IOException {
        MatlabStateflowGrabbberException if an error is encountered
        by Matlab.
    }

    /**
     * Obtains the unique identifier of the Stateflow object
     * specified by <code><obj></code>.
     * param obj - the object of which one wants to know the id.
     * Returns the id of obj.
     * Throws IOException if an error occurs in the input or output stream.
     * by Matlab.
     */
    private static Integer getObjectID(final String obj)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand("obj" + ID);
        return new Integer(answer[2]);
    }

    /**
     * Obtains the source of the transition specified by <code><trans></code>.
     * param trans - the transition of which one wants to know the source.
     * Returns the id of the source of the transition <code><trans></code>.
     * Throws IOException if an error occurs in the input or output stream.
     * by Matlab.
     */
    private static Integer getSourceID(final String trans)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand("source=" + trans + SRC);
        if (answer[2].equals("[]")) {
            this transition has got no source: initial
        }
        return new Integer(-1);
    }

    /**
     * Obtains the destination of the transition specified
     * by <code><trans></code>.
     * param trans - the transition of which one wants to know
     * the destination.
     * Returns the id of the destination of the transition <code><trans></code>.
     */
    private static Integer getDestinationID(final String trans)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand("dest=" + trans + DEST);
        if (answer[2].equals("[]")) {
            this destination has got no destination: initial
        }
        return new Integer(-1);
    }

    /**
     * Obtains the source end point of the transition specified
     * by <code><trans></code>.
     * param trans - the transition of which one wants to know
     * the source end point.
     * Returns the source end point [x, y].
     */
    private static int[] getTransitionSourceEndPoint(final String trans)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand(trans + SRC_EP);
        int[] sep = new int[2];
        sep[0] = (int) (Float.parseFloat(answer[2]) * chartScale);
        sep[1] = (int) (Float.parseFloat(answer[3]) * chartScale);
        return sep;
    }

    /**
     * Obtains the midpoint of the transition specified by <code><trans></code>.
     * param trans - the transition of which one wants to know the midPoint.
     * Returns the midpoint [x, y].
     */
    private static int[] getTransitionMidPoint(final String trans)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand(trans + TR_MIDPOINT);
        int[] pos = new int[2];
        pos[0] = (int) (Float.parseFloat(answer[2]) * chartScale);
        pos[1] = (int) (Float.parseFloat(answer[3]) * chartScale);
        return pos;
    }

    /**
     * Obtains the destination o'clock value of the transition specified
     * by <code><trans></code> and calculates the coordinates of the
     * destination end point.
     */
    private static int[] getTransitionDestEndPoint(final String trans)
        throws IOException, MatlabStateflowGrabbberException {
        String[] answer = handleMatlabCommand(trans + TR_DESTINATION);
        int[] dest = new int[2];
        dest[0] = (int) (Float.parseFloat(answer[2]) * chartScale);
        dest[1] = (int) (Float.parseFloat(answer[3]) * chartScale);
        return dest;
    }
}

```

B. Java Code

```

    * @throws MatlabStateflowGrabberException if an error is encountered
    * by Matlab.
    */
    private static int[] getTransitionDestinationEndPoint(final String trans,
        final Node destState)
        throws IOException {
            MatlabStateflowGrabberException {
                710
                    int[] intersection = new int[2];
                    Point pIntersect = null;
                    NodeLayoutInformation nli = actualView.getLayoutInformation(destState);
                    int[] pos = (nli.getPosition(), nli.getWidth(), nli.getHeight());
                    nli.getHeight();
                    Point corner1 = new Point(pos[0], pos[1]);
                    Point corner2 = new Point(pos[0] + pos[2], pos[1]);
                    Point corner3 = new Point(pos[0] + pos[2], pos[1] + pos[3]);
                    Point corner4 = new Point(pos[0], pos[1] + pos[3]);
                    Point center = new Point(pos[0] + pos[2] / 2, pos[1] + pos[3] / 2);
                    float oclock = getDestinationOClock(trans);
                    double oclockRad = (2 * Math.PI / 12) * oclock;
                    double c = Math.sqrt((double) ((pos[2] / 2) * (pos[2] / 2) +
                        (pos[3] / 2) * (pos[3] / 2)));
                    double alpha1 = Math.asin((double) (pos[2] / 2) / c);
                    double alpha2 = ((2 * Math.PI - 4 * alpha1) / 2) + alpha1;
                    double alpha3 = alpha2 + 2 * alpha1;
                    double alpha4 = 2 * Math.PI - alpha1;
                    720
                    int x = (int) (Math.sin(oclockRad) * 100); x = x + center.getX();
                    int y = (int) (Math.cos(oclockRad) * 100); y = -y + center.getY();
                    Point p2 = new Point(x, y);
                    Line direction = new Line(center, p2);
                    Line line = null;
                    if ((oclockRad <= alpha1) || (oclockRad > alpha4)) {
                        line = new Line(corner1, corner2);
                    } else if ((oclockRad <= alpha2) && (oclockRad > alpha1)) {
                        line = new Line(corner2, corner3);
                    } else if ((oclockRad <= alpha3) && (oclockRad > alpha2)) {
                        line = new Line(corner3, corner4);
                    } else {
                        line = new Line(corner4, corner1);
                    }
                    try {
                        pIntersect = direction.intersect(line);
                        catch (LineDotIntersectionException lndi) {
                            log.log(0, lndi.toString());
                        }
                        intersection[0] = pIntersect.getX();
                        intersection[1] = pIntersect.getY();
                        return intersection;
                    }
                    730
                    /**
                     * Obtains the destination o'clock value of the transition specified
                     * by <code>trans</code>
                     * param trans - the transition for which one wants to know the
                     * destination o'clock value.
                     * returns the destination o'clock value of <code>trans</code>
                     * throws IOException if an error occurs in the input or output stream.
                     * throws MatlabStateflowGrabberException if an error is encountered
                     * by Matlab.
                     */
                    private static float getDestinationOClock(final String trans)
                        throws IOException, MatlabStateflowGrabberException {
                            740
                            String[] answer = handleLabCommand(trans + DEST_OCLOCK);
                            return Float.parseFloat(answer[2]);
                        }
                        /**
                         * Obtains the number for transition <code>trans</code>
                         * in the execution order for its source.
                         * param trans - the transition for which one wants to know
                         * the number in the execution order.
                         * returns the number in the execution order.
                         * throws IOException if an error occurs in the input or output stream.
                         * throws MatlabStateflowGrabberException if an error is encountered
                         * by Matlab.
                         */
                        private static int getTransitionExecutionOrder(final String trans)
                            throws IOException, MatlabStateflowGrabberException {
                                750
                                String[] answer = handleLabCommand(trans + EXEC_ORDER);
                                return Integer.parseInt(answer[2]);
                            }
                            /**
                             * Determines whether the state specified by the argument
                             * <code>state</code> is an OR state.
                             * param state - the state for which one wants to know whether it is
                             * an OR state.
                             * returns <code>true</code> if <code>state</code> is an OR state,
                             * <code>false</code> otherwise.
                             * throws IOException if an error occurs in the input or output stream.
                             * throws MatlabStateflowGrabberException if an error is encountered
                             * by Matlab.
                             */
                            private static boolean isORState(final String state) throws IOException,
                                MatlabStateflowGrabberException {
                                    760
                                    String[] answer = handleLabCommand(state + DECOMPO);
                                    return answer[2].equals("EXCLUSIVE_OR");
                                }
                                /**
                                 * Determines whether the state specified by the argument
                                 * <code>state</code> is an AND state.
                                 * param state - the state for which one wants to know whether it is
                                 * an AND state.
                                 * returns <code>true</code> if <code>state</code> is an AND state,
                                 * <code>false</code> otherwise.
                                 * throws IOException if an error occurs in the input or output stream.
                                 * throws MatlabStateflowGrabberException if an error is encountered
                                 * by Matlab.
                                 */
                                private static boolean isANDState(final String state) throws IOException,
                                    MatlabStateflowGrabberException {
                                        770
                                        String[] answer = handleLabCommand(state + DECOMPO);
                                        return answer[2].equals("PARALLEL_AND");
                                    }
                                    /**
                                     * Determines whether the state specified by the argument
                                     * <code>state</code> is a region.
                                     */

```

```

    * <param state - the state for which one wants to know whether it is
    * a region.
    * <return <code>true</code> if <code>state</code> is a region,
    * <code>false</code> otherwise.
    * <throws IOException if an error occurs in the input or output stream.
    * <throws MatlabStateflowGrubberException if an error is encountered
    * by Matlab.
    */
830 private static boolean isRegion(final String state) throws IOException {
    MatlabStateflowGrubberException e = handleMatlabCommand(state + TYPE);
    String[] answer = e.getAnswer();
    return answer[2].equals("AND");
}

/**
 * Determines whether the junction specified by the argument <code>
 * Junction</code> is a history junction.
 * <param junction - the Junction for which one wants to know whether
 * it is a history Junction.
 * <return <code>true</code> if <code>junction</code> is a history
 * Junction, <code>false</code> otherwise.
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
 */
840 private static boolean isHistory(final String junction) throws IOException {
    MatlabStateflowGrubberException e = handleMatlabCommand(junction + TYPE);
    String[] answer = e.getAnswer();
    return answer[2].equals("HISTORY");
}

/**
 * Determines whether the state specified by the argument
 * <code>state</code> is a subchart.
 * <param state - the state for which one wants to know whether it
 * is a subchart.
 * <return <code>true</code> if <code>state</code> is a subchart,
 * <code>false</code> otherwise.
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
 */
850 private static boolean isSubchart(final String state) throws IOException {
    MatlabStateflowGrubberException e = handleMatlabCommand(state + SUBCHART);
    String[] answer = e.getAnswer();
    return answer[2].equals("1");
}

/**
 * Obtains the position of the junction specified by the argument
 * <code>junction</code>.
 * <param junction - the Junction for which one wants to know the position.
 * <return the position of the junction specified by the argument.
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
 */
860 private static int[] getJunctionPosition(final String junction)
    throws IOException, MatlabStateflowGrubberException {
    MatlabStateflowGrubberException e = handleMatlabCommand(junction + JUNC_POS);
    String[] answer = e.getAnswer();
    return answer[2];
}

/**
 * Obtains the initial value of the variable specified by the argument
 * <code>var</code> as a double.
 * <param var - the variable of which one wants to know the initial value.
 * <return the initial value of the variable <code>var</code> as a double.
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
 */
870 private static double getInitialValue(final String var) throws IOException {
    MatlabStateflowGrubberException e = handleMatlabCommand(var + INITIAL_VALUE);
    String[] answer = e.getAnswer();
    return Double.parseDouble(answer[2]);
}

/**
 * Obtains the initial value of the variable specified by the argument
 * <code>var</code> as a string.
 * <param var - the variable of which one wants to know the initial value.
 * <return the initial value of the variable <code>var</code> as a string.
 * <throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
 */
880 private static String getInitialValueString(final String var)
    throws IOException, MatlabStateflowGrubberException {
    MatlabStateflowGrubberException e = handleMatlabCommand(var + INITIAL_VALUE);
    String[] answer = e.getAnswer();
    return answer[2];
}

```

B. Java Code

```

* param var - the variable of which one wants to know the initial value.
* returns the initial value of the variable <code>var</code> as a string.
* throws IOException if an error occurs in the input or output stream.
* throws MatlabStateflowGrubberException if an error is encountered
* by Matlab.
950

private static String getInitialValueAsString(final String var)
throws IOException, MatlabStateflowGrubberException {
    String[] answer = handleMatlabCommand(var + INITIAL_VALUE);
    return answer[2];
}

/**
 * obtains the port number of an event or data object.
 * @param eventOrData - the event or data object of which one wants to
 * know the port number.
 * @return the port number of the specified event or data.
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
960

private static int getPortNumber(final String eventOrData)
throws IOException, MatlabStateflowGrubberException {
    String[] answer = handleMatlabCommand(eventOrData + PORT);
    return Integer.parseInt(answer[2]);
}

/**
 * obtains the trigger type of the event specified by the argument
 * <code>event</code>.
 * @param event - the event of which one wants to know the trigger type.
 * @return the trigger type of the specified event
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
970

private static String getTriggerType(final String event)
throws IOException, MatlabStateflowGrubberException {
    String[] answer = handleMatlabCommand(event + TRIGGER);
    return answer[2];
}

/**
 * obtains the actions of the state in Stateflow specified by the argument
 * <code>state</code> and creates equivalent actions using the KIEL
 * data structure. The actions are created in the state in KIEL specified
 * by the argument <code>stateRef</code>,
 * @param state - the Stateflow state from which one wants to
 * obtain the actions.
 * @param stateRef - the KIEL state for which the obtained actions
 * have to be created.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
980

private static void getAndCreateActions(final String state,
final State stateRef)
throws IOException, MatlabStateflowGrubberException {
    String[] answer = handleMatlabCommand(state + " " + stateRef);
    String events = getArraySizeOf("events");
    int noe = getMatlabResponse();
    String eventString;
    for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
        eventString = "events(" + (eventNumber + 1) + ")";
        log.log(0, "Get and create " + events);
        MatlabStateflowGrubberException {
            log.log(0, "Get and create " + events);
            ((CompositeState) stateRef).addLocalEvent(
                new Event(getObjectName(eventString)));
    }
}

/**
 * obtains the actions of the state in Stateflow specified by the argument
 * <code>state</code> and creates equivalent actions using the KIEL
 * data structure. The actions are created in the state in KIEL specified
 * by the argument <code>stateRef</code>,
 * @param state - the Stateflow state from which one wants to
 * obtain the actions.
 * @param stateRef - the KIEL state for which the obtained actions
 * have to be created.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
990

private static void getAndCreateActions(final String state,
final State stateRef)
throws IOException, MatlabStateflowGrubberException {
    String[] answer = handleMatlabCommand(state + " " + stateRef);
    String events = getArraySizeOf("events");
    int noe = getMatlabResponse();
    String eventString;
    for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
        eventString = "events(" + (eventNumber + 1) + ")";
        log.log(0, "Get and create " + events);
        MatlabStateflowGrubberException {
            log.log(0, "Get and create " + events);
            ((CompositeState) stateRef).addLocalEvent(
                new Event(getObjectName(eventString)));
    }
}

/**
 * obtains the actions of the state in Stateflow specified by the argument
 * <code>state</code> and creates equivalent actions using the KIEL
 * data structure. The actions are created in the state in KIEL specified
 * by the argument <code>stateRef</code>,
 * @param state - the Stateflow state from which one wants to
 * obtain the actions.
 * @param stateRef - the KIEL state for which the obtained actions
 * have to be created.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrubberException if an error is encountered
 * by Matlab.
1000

```

```

* by the argument <code>stateRef</code>.
* &gt; param state - the Stateflow state from which one wants to
* obtain the actions.
* &gt; param stateRef - the KIEL state for which the obtained actions
* have to be created.
* &gt; throws IOException if an error occurs in the input or output stream. 1130
* &gt; throws MatlabStateflowGrabberException if an error is encountered
* by Matlab.
*/
private static void getAndCreateLocalVariables(final String state,
final State stateRef)
throws IOException, MatlabStateflowGrabberException {
    log.log(0, "Get and create local variables");
    toMatlab.write("variables=" + state + GET_VARIABLES_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

int nov = getarrySizeOf("variables");
String variableString;
String type;
String scope;
String name;
Variable var;
Constant con;

for (int variableNumber = 0; variableNumber < nov; variableNumber++) { 1150
    var = null;
    con = null;
    variableString = "variables(" + (variableNumber + 1) + ")";
    type = getDataType(variableString);
    scope = getDataScope(variableString);
    name = getObjJchname(variableString);
    if (type.equals("double")) {
        con = new DoubleConstant(name,
            getInitialValue(variableString));
        var = new DoubleVariable(name, (DoubleConstant) con);
    } else if (type.equals("single")) {
        con = new FloatConstant(name,
            (float) getInitialValue(variableString));
        var = new FloatVariable(name, (FloatConstant) con);
    } else if (type.equals("int32")) {
        con = new IntegerConstant(name,
            (int) getInitialValue(variableString));
        var = new IntegerVariable(name, (IntegerConstant) con);
    } else if (type.equals("int16")) {
        con = new Integer16Constant(name,
            (short) getInitialValue(variableString));
        var = new Integer16Variable(name, (Integer16Constant) con);
    } else if (type.equals("int8")) {
        con = new Integer8Constant(name,
            (byte) getInitialValue(variableString));
        var = new Integer8Variable(name,
            (Integer8Constant) con);
    } else if (type.equals("uint32")) {
        con = new UInt32Constant(name,
            (long) getInitialValue(variableString));
        var = new UInt32Variable(name, (UInt32Constant) con);
    } else if (type.equals("uint16")) {
        con = new UInt16Constant(name,
            (short) getInitialValue(variableString));
        var = new UInt16Variable(name, (UInt16Constant) con);
    }
}

1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200
1210
1220
1230
1240
1250
1260
1270
1280
1290
1300
1310
1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500
1510
1520
1530
1540
1550
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
2060
2070
2080
2090
2100
2110
2120
2130
2140
2150
2160
2170
2180
2190
2200
2210
2220
2230
2240
2250
2260
2270
2280
2290
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
2600
2610
2620
2630
2640
2650
2660
2670
2680
2690
2700
2710
2720
2730
2740
2750
2760
2770
2780
2790
2800
2810
2820
2830
2840
2850
2860
2870
2880
2890
2900
2910
2920
2930
2940
2950
2960
2970
2980
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
9999
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135
10140
10145
10150
10155
10160
10165
10170
10175
10180
10185
10190
10195
10200
10205
10210
10215
10220
10225
10230
10235
10240
10245
10250
10255
10260
10265
10270
10275
10280
10285
10290
10295
10300
10305
10310
10315
10320
10325
10330
10335
10340
10345
10350
10355
10360
10365
10370
10375
10380
10385
10390
10395
10400
10405
10410
10415
10420
10425
10430
10435
10440
10445
10450
10455
10460
10465
10470
10475
10480
10485
10490
10495
10500
10505
10510
10515
10520
10525
10530
10535
10540
10545
10550
10555
10560
10565
10570
10575
10580
10585
10590
10595
10600
10605
10610
10615
10620
10625
10630
10635
10640
10645
10650
10655
10660
10665
10670
10675
10680
10685
10690
10695
10700
10705
10710
10715
10720
10725
10730
10735
10740
10745
10750
10755
10760
10765
10770
10775
10780
10785
10790
10795
10800
10805
10810
10815
10820
10825
10830
10835
10840
10845
10850
10855
10860
10865
10870
10875
10880
10885
10890
10895
10900
10905
10910
10915
10920
10925
10930
10935
10940
10945
10950
10955
10960
10965
10970
10975
10980
10985
10990
10995
11000
11005
11010
11015
11020
11025
11030
11035
11040
11045
11050
11055
11060
11065
11070
11075
11080
11085
11090
11095
11100
11105
11110
11115
11120
11125
11130
11135
11140
11145
11150
11155
11160
11165
11170
11175
11180
11185
11190
11195
11200
11205
11210
11215
11220
11225
11230
11235
11240
11245
11250
11255
11260
11265
11270
11275
11280
11285
11290
11295
11300
11305
11310
11315
11320
11325
11330
11335
11340
11345
11350
11355
11360
11365
11370
11375
11380
11385
11390
11395
11400
11405
11410
11415
11420
11425
11430
11435
11440
11445
11450
11455
11460
11465
11470
11475
11480
11485
11490
11495
11500
11505
11510
11515
11520
11525
11530
11535
11540
11545
11550
11555
11560
11565
11570
11575
11580
11585
11590
11595
11600
11605
11610
11615
11620
11625
11630
11635
11640
11645
11650
11655
11660
11665
11670
11675
11680
11685
11690
11695
11700
11705
11710
11715
11720
11725
11730
11735
11740
11745
11750
11755
11760
11765
11770
11775
11780
11785
11790
11795
11800
11805
11810
11815
11820
11825
11830
11835
11840
11845
11850
11855
11860
11865
11870
11875
11880
11885
11890
11895
11900
11905
11910
11915
11920
11925
11930
11935
11940
11945
11950
11955
11960
11965
11970
11975
11980
11985
11990
11995
12000
12005
12010
12015
12020
12025
12030
12035
12040
12045
12050
12055
12060
12065
12070
12075
12080
12085
1209
```

B. Java Code

```

String name;
Variable var;
Constant con;

for (int variableNumber = 0; variableNumber < nov; variableNumber++) { 1190
    var = null;
    con = null;
    variableString = "variables(" + (variableNumber + 1) + ")";
    type = genDataTypee(variableString);
    scope = getDataScope(variableString);
    name = getObjectName(variableString);
    if (type.equals("double")) {
        con = new DoubleConstant(name,
            getInitialValue(variableString));
        var = new DoubleVariable(name, (DoubleConstant) con);
    } else if (type.equals("single")) {
        con = new FloatConstant(name,
            (float) getInitialValue(variableString));
        var = new FloatVariable(name, (FloatConstant) con);
    } else if (type.equals("int32")) {
        con = new IntegerConstant(name,
            (int) getInitialValue(variableString));
        var = new IntegerVariable(name, (IntegerConstant) con);
    } else if (type.equals("int16")) {
        con = new Integer16Constant(name,
            (short) getInitialValue(variableString));
        var = new Integer16Variable(name, (Integer16Constant) con);
    } else if (type.equals("int8")) {
        con = new Integer8Constant(name,
            (byte) getInitialValue(variableString));
        var = new Integer8Variable(name,
            (IntegerConstant) con);
    } else if (type.equals("uint32")) {
        con = new UInt32Constant(name,
            (long) getInitialValue(variableString));
        var = new UInt32Variable(name, (UInt32Constant) con);
    } else if (type.equals("uint16")) {
        con = new UInt16Constant(name,
            (int) getInitialValue(variableString));
        var = new UInt16Variable(name, (UInt16Constant) con);
    } else if (type.equals("uint8")) {
        con = new UInt8Constant(name,
            (short) getInitialValue(variableString));
        var = new UInt8Variable(name, (UInt8Constant) con);
    } else if (type.equals("boolean")) {
        if (getInitialValueString(variableString).equals("true"))
            con = new BooleanTrue(name);
        var = new BooleanVariable(name, (BooleanExpression) con);
    } else {
        con = new BooleanFalse(name);
        var = new BooleanVariable(name, (BooleanExpression) con);
    }
} else if (type.equals("State")) {
    /* We do nothing here because we do not want to treat them as
     * output variables. They are just used to determine the
     * set of currently active states!
    */
    var =
}
new StringVariable(name
    + getInitialValueAsString(variableString));
}

if (var != null) {
    if (scope.equals("Local")) {
        stateful.addField(var);
    } else if (scope.equals("Input")) {
        stateful.addFieldVariable(var);
    } else if (scope.equals("Output")) {
        stateful.addOutputVariable(var);
    } else if (scope.equals("Constant")) {
        stateful.addConstant(con);
    } else {
        throw new MatlabStateflowGrabberException(
            "Chart contains variable of unsupported scope!");
    }
}

/*
 * Tests whether the set of regions specified by the <code>regions</code>
 * argument is horizontally alignable.
 * @param regions - the set of regions which have to be tested
 * @return <code>true</code> if the given set of regions is
 * horizontally alignable, <code>false</code> otherwise.
 */
private static boolean isHorizontallyAlignable(final State[] regions) {
    int[] prePos = new int[4];
    int[] pos = new int[4];
    CompositeStateLayoutInformation cli;
    cli = (CompositeStateLayoutInformation)
        actualView.getLayoutInformation(regions[0]);
    prePos[0] = cli.getPos();
    prePos[1] = cli.getPos();
    prePos[2] = cli.getPos();
    prePos[3] = cli.getPos();
    for (int i = 1; i < regions.length; i++) {
        cli = (CompositeStateLayoutInformation)
            actualView.getLayoutInformation(regions[i]);
        pos[0] = cli.getPos();
        pos[1] = cli.getPos();
        pos[2] = cli.getPos();
        pos[3] = cli.getPos();
        if ((prePos[0] + prePos[2]) < pos[0]
            || (pos[0] + pos[2] < prePos[0])) {
            cli.getHeight();
            if (cli.getHeight() != pos[3])
                return false;
        }
    }
    return true;
}

/*
 * Tests whether the set of regions specified by the <code>regions</code>
 * argument is vertically alignable.
 * @param regions - the set of regions which have to be tested
 */

```

```

    * @return <code>true</code> if the given set of regions is
    * vertically alignable, <code>false</code> otherwise.
    */
    private static boolean isVerticallyAlignable(final State[] regions) {
        1310    int[] prePos = new int[4];
        pos = new int[4];
        cli = CompositeStateLayoutInformation cli;
        cli = (CompositeStateLayoutInformation)
            actualView.getLayoutInformation(regions[0]);
        prevPos[0] = cli.getPosition();
        prevPos[1] = cli.getWidth();
        prevPos[2] = cli.getHeight();
        prevPos[3] = cli.getDepth();
        for (int i = 1; i < regions.length; i++) {
            1320    cli = (CompositeStateLayoutInformation)
                actualView.getLayoutInformation(regions[i]);
            pos[0] = cli.getPosition();
            pos[1] = cli.getWidth();
            pos[2] = cli.getHeight();
            pos[3] = cli.getDepth();
            if ((prevPos[1] + prevPos[3]) < pos[1]) {
                || (pos[1] + pos[3] < prevPos[1])) {
                    ;
                } else {
                    return false;
                }
            }
            return true;
        }
        /**
         * Generates delimiter lines that separate the specified regions.
         * @param regions - the set of regions which have to be separated by
         * delimiter lines
         * @param parent - the state that contains the regions.
         * @param parentPos - the coordinates of the parent state.
         */
        1330    private static void generateDelimiterLines(final State[] regions,
            final CompositeState parent,
            final int[] parentPos) {
            int[] prePos = new int[4];
            pos = new int[4];
            int upperOffset;
            DelimiterLine dell;
            Linkedlist verticalDels = new LinkedList();
            CompositeStateLayoutInformation cli;
            Arrays.sort(regions, new StateComparator(actualView));
            // if (isHorizontallyAlignable(regions)) {
            //     Arrays.sort(regions, new StateComparatorHorizontal(actualView));
            // } else if (isVerticallyAlignable(regions)) {
            //     Arrays.sort(regions, new StateComparatorVertical(actualView));
            // } else {
            //     Arrays.sort(regions, new StateComparator(cli));
            // }
            1340    upperOffset = Properties.getUpperOffset(parent);
            for (int i = 0; i < regions.length; i++) {
                cli = (CompositeStateLayoutInformation)
                    actualView.getLayoutInformation(regions[i]);
                prevPos[0] = pos[0];
                prevPos[1] = pos[1];
                prevPos[2] = pos[2];
                prevPos[3] = pos[3];
                pos[0] = cli.getPosition();
                pos[1] = cli.getWidth();
                pos[2] = cli.getHeight();
                pos[3] = cli.getDepth();
                pos[12] = cli.getDepth();
                pos[13] = cli.getDepth();
                if (i != 0) {
                    /*create a vertical delimiter line*/
                    if ((prevPos[0] + prevPos[2]) < pos[0]) {
                        dell = new DelimiterLine();
                        ((ANDState) parent).addDelimiterLine(dell);
                        int x1 = (pos[0] - prevPos[0])
                            - prevPos[2]) / 2 + prevPos[0] + prevPos[2];
                        DelimiterLineLayoutInformation dellInfo =
                            new DelimiterLineLayoutInformation(
                                new Point(x1, upperOffset),
                                new Point(x1, parentPos[3]));
                        actualView.setLayoutInformation(dell, dellInfo);
                        verticalDels.add(dell);
                    } else
                        //if((pos[0] + pos[2] < prevPos[0])) {
                            // dell = new DelimiterLine();
                            // ((ANDState) parent).addDelimiterLine(dell);
                            int x1 = -(prevPos[0] - pos[0])
                                - pos[2]) / 2 + pos[0] + pos[2];
                            DellimiterLineLayoutInformation dellInfo =
                                new DelimiterLineLayoutInformation(dellInfo =
                                    new Point(x1, upperOffset),
                                    new Point(x1, parentPos[3]));
                            actualView.setLayoutInformation(dell, dellInfo);
                        //}
                    }
                    1350    DelimiterLine dellInfo;
                    Linkedlist verticalDels = new LinkedList();
                    CompositeStateLayoutInformation cli;
                    Arrays.sort(regions, new StateComparator(actualView));
                    // if (isHorizontallyAlignable(regions)) {
                    //     Arrays.sort(regions, new StateComparatorHorizontal(actualView));
                    // } else if (isVerticallyAlignable(regions)) {
                    //     Arrays.sort(regions, new StateComparatorVertical(actualView));
                    // } else {
                    //     Arrays.sort(regions, new StateComparator(cli));
                    // }
                    1360    upperOffset = Properties.getUpperOffset(parent);
                    for (int i = 0; i < regions.length; i++) {
                        cli = (CompositeStateLayoutInformation)
                            actualView.getLayoutInformation(regions[i]);
                        prevPos[0] = pos[0];
                        prevPos[1] = pos[1];
                        prevPos[2] = pos[2];
                        prevPos[3] = pos[3];
                        pos[0] = cli.getPosition();
                        pos[1] = cli.getWidth();
                        pos[2] = cli.getHeight();
                        pos[3] = cli.getDepth();
                        pos[12] = cli.getDepth();
                        pos[13] = cli.getDepth();
                        if (i != 0) {
                            /*create a vertical delimiter line*/
                            if ((prevPos[0] + prevPos[2]) < pos[0]) {
                                dell = new DelimiterLine();
                                ((ANDState) parent).addDelimiterLine(dell);
                                int y1 = (prevPos[1] - pos[1]) -
                                    pos[3]) / 2 + pos[1] + pos[3];
                                DellimiterLineLayoutInformation dellInfo =
                                    new DellimiterLineLayoutInformation(dellInfo =
                                        new Point(0, y1),
                                        new Point(parentPos[2], y1));
                                actualView.setLayoutInformation(dell, dellInfo);
                                upperOffset = y1;
                                if (verticalDels.size() > 0) {
                                    calAndSetCrossingEndPoints(verticalDels, y1);
                                }
                            } else
                                //if(pos[1] + pos[3] < prevPos[1]) {
                                    // dell = new DelimiterLine();
                                    // ((ANDState) parent).addDelimiterLine(dell);
                                    // int y1 = (prevPos[1] - pos[1]) -
                                    // pos[3]) / 2 + pos[1] + pos[3];
                                    DellimiterLineLayoutInformation dellInfo =
                                        new DellimiterLineLayoutInformation(dellInfo =
                                            new Point(0, y1),
                                            new Point(parentPos[2], y1));
                                    actualView.setLayoutInformation(dell, dellInfo);
                                //}
                            }
                        }
                    }
                }
            }
        }
    }
}

```

B. Java Code

```

        actualView.setLayoutInformation(deli, deli);
        // if(verticalDels.size() > 0){
        //     calcAndSetCrossingEndPoints(verticalDels, y1);
        //}
        // j
    }
    cli.setupLeftPoint(new Point(pos[0], upperOffset));
    1430    cli.setWidth(pos[2]);
    cli.setHeight(pos[3]);
}
}

/*
* Calculates and sets the end points of the vertical delimiter lines
* specified by the argument <code>deli</code> to the points of
* intersection between the vertical delimiter lines and the horizontal
* delimiter line that goes through <code>y</code>.
*
* @param deli - the set of vertical delimiter lines for which the
* end points have to be adapted.
* @param y - the y coordinate of the horizontal delimiter line that
* intersects the vertical ones.
*/
private static void calcAndSetCrossingEndPoints(final LinkedList vdel,
final int y) {
    final int[] pos = vdel.listIterator(0);
    DelimiterLine deli;
    DelimiterLine deli;
    DelimiterLineLayoutInformation deli;
    int x;
    while (iter.hasNext()) {
        deli = (DelimiterLine) iter.next();
        deli = actualView.getLayoutInformation(deli);
        x = deli.getEnd().getX();
        deli.setEnd(new Point(x, y));
    }
    1440
}
}

/*
* Grabs all children of the currently processed state.
* Throws IOException if an error occurs in the input or output stream.
* Throws MatlabStateflowGrabberException if an error is encountered
* by Matlab.
*/
private static void grabOneLevel() throws IOException {
    MatlabStateflowGrabberException f;
    log.log(0, "GrabOneLevel");
    /* Get all states out of Matlab level by level of containment */
    log.log(0, "GrabStates:");
    toMatlab.write("states=parent" + GET_STATES_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    toMatlab.write("states=setdiff(states, parent)");
    toMatlab.newLine();
    toMatlab.flush();
    1450
}
}

/*
* Logs all junctions out of Matlab level by level of containment */
1460
}

/*
* Grabs all children of the currently processed state.
* Throws IOException if an error occurs in the input or output stream.
* Throws MatlabStateflowGrabberException if an error is encountered
* by Matlab.
*/
private static void MatlabStateflowGrabberException f;
log.log(0, "GrabOneLevel");
/* Get all states out of Matlab level by level of containment */
log.log(0, "GrabStates:");
toMatlab.write("states=parent" + GET_STATES_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();
toMatlab.write("states=setdiff(states, parent)");
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();
int nos = getArraySize0("states");
int[] parentPos = new int[4];
int[] pos = null;
int[] prevPos = null;
1470
}

/*
* Sets the regions = new Region[nos];
* Delimiterine dell = null;
* String stateString = null;
* int upperOffset = 0;
*/
CompositeState parent = (CompositeState) map.get(getObjectID("parent"));
if (parent.getParent() != null) {
    parentPos = getObjectPosition("parent");
}
for (int stateNumber = 0; stateNumber < nos; stateNumber++) {
    stateString = "states" + (stateNumber + 1) + ",";
    pos = pos;
    prevPos = pos;
    pos = getEObjectPosition(stateString);
    if (issubchart(stateString)) {
        containsSubcharts = true;
    }
    if ((isSimpleState(stateString) && (!isRegion(stateString))) {
        state = new SimpleState();
        state = new SimpleState();
        map.put(getObjectID(stateString), state);
        NoteLayoutInformation nli = new NoteLayoutInformation();
        nli.setLayoutPoint(new Point(pos[0])
            - parentPos[0],
            pos[1] - parentPos[1]);
        nli.setWidth(pos[2]);
        nli.setHeight(pos[3]);
        actualView.setLayoutInformation(state, nli);
    } else {
        if (isORState(stateString)) {
            state = new ORState();
        } else {
            state = new ANDState();
        }
        if (isRegion(stateString)) {
            state = new Region();
            regions[stateNumber] = state;
        }
        else {
            if (isORState(stateString)) {
                state = new ORState();
            } else {
                state = new ANDState();
            }
            if (isRegion(stateString)) {
                state = new Region();
                regions[stateNumber] = state;
            }
            cli.setWidth(pos[2]);
            cli.setHeight(pos[3]);
            actualView.setLayoutInformation(state, cli);
        }
    }
}
state.setName(getObjectName(stateString));
parent.addSubnode(state);
state.setParent(parent);
getAndCreateActions(statString, state);
getAndCreateLocalEvents(statString, state);
getAndCreateLocalVariables(statString, state);
}
if (regions[0] != null) {
    generatedDelimiterLines(regions, parent, parentPos);
}
1480
}

/*
* Get all Junctions out of Matlab level by level of containment */

```

```

log.log(0, "Grab_junctions_and_histories");
toMatLab.write("junctions=parent" + GET_JUNCTIONS_COMMAND);
toMatLab.newLine();
toMatLab.flush();
checkMatLabResponse();
int nojs = getNumberOfJunctions();
Node node = null;
String juncString = null;
for (int junctionNumber = 0; junctionNumber < nojs; junctionNumber++) {
    1550
    juncString = "junctions(" + (junctionNumber + 1) + ")";
    if (isHistory(juncString)) {
        node = new History();
    } else {
        node = new Junction();
    }
    parent.addSubnode(node);
    node.setParent(parent);
    map.put(getObjectID(juncString), node);
    pos = getJunctionPosition(juncString);
    int rad = getJunctionRadius(juncString);
    NodeLayoutInformation nli = new NodeLayoutInformation();
    nli.setLayoutEndPoint(new Point(pos[0] - rad - parentPos[0]),
    nli.setWidth(rad * 2);
    nli.setHeight(rad * 2);
    actualView.setLayoutInformation(node, nli);
}
/*Get all Transitions out of Matlab level by level of containment*/
log.log(0, "Grab_transitions");
toMatLab.write("transitions=parent" + GET_TRANSITIONS_COMMAND);
toMatLab.newLine();
toMatLab.flush();
checkMatLabResponse();
int nojs = getNumberOfTransitions();
Transition trans = null;
String transString = null;
int[] sep;
int[] map;
int[] dep;
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
2060
2070
2080
2090
2100
2110
2120
2130
2140
2150
2160
2170
2180
2190
2200
2210
2220
2230
2240
2250
2260
2270
2280
2290
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
2600
2610
2620
2630
2640
2650
2660
2670
2680
2690
2700
2710
2720
2730
2740
2750
2760
2770
2780
2790
2800
2810
2820
2830
2840
2850
2860
2870
2880
2890
2900
2910
2920
2930
2940
2950
2960
2970
2980
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
9999
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135
10140
10145
10150
10155
10160
10165
10170
10175
10180
10185
10190
10195
10200
10205
10210
10215
10220
10225
10230
10235
10240
10245
10250
10255
10260
10265
10270
10275
10280
10285
10290
10295
10300
10305
10310
10315
10320
10325
10330
10335
10340
10345
10350
10355
10360
10365
10370
10375
10380
10385
10390
10395
10400
10405
10410
10415
10420
10425
10430
10435
10440
10445
10450
10455
10460
10465
10470
10475
10480
10485
10490
10495
10500
10505
10510
10515
10520
10525
10530
10535
10540
10545
10550
10555
10560
10565
10570
10575
10580
10585
10590
10595
10600
10605
10610
10615
10620
10625
10630
10635
10640
10645
10650
10655
10660
10665
10670
10675
10680
10685
10690
10695
10700
10705
10710
10715
10720
10725
10730
10735
10740
10745
10750
10755
10760
10765
10770
10775
10780
10785
10790
10795
10800
10805
10810
10815
10820
10825
10830
10835
10840
10845
10850
10855
10860
10865
10870
10875
10880
10885
10890
10895
10900
10905
10910
10915
10920
10925
10930
10935
10940
10945
10950
10955
10960
10965
10970
10975
10980
10985
10990
10995
11000
11005
11010
11015
11020
11025
11030
11035
11040
11045
11050
11055
11060
11065
11070
11075
11080
11085
11090
11095
11100
11105
11110
11115
11120
11125
11130
11135
11140
11145
11150
11155
11160
11165
11170
11175
11180
11185
11190
11195
11200
11205
11210
11215
11220
11225
11230
11235
11240
11245
11250
11255
11260
11265
11270
11275
11280
11285
11290
11295
11300
11305
11310
11315
11320
11325
11330
11335
11340
11345
11350
11355
11360
11365
11370
11375
11380
11385
11390
11395
11400
11405
11410
11415
11420
11425
11430
11435
11440
11445
11450
11455
11460
11465
11470
11475
11480
11485
11490
11495
11500
11505
11510
11515
11520
11525
11530
11535
11540
11545
11550
11555
11560
11565
11570
11575
11580
11585
11590
11595
11600
11605
11610
11615
11620
11625
11630
11635
11640
11645
11650
11655
11660
11665
11670
11675
11680
11685
11690
11695
11700
11705
11710
11715
11720
11725
11730
11735
11740
11745
11750
11755
11760
11765
11770
11775
11780
11785
11790
11795
11800
11805
11810
11815
11820
11825
11830
11835
11840
11845
11850
11855
11860
11865
11870
11875
11880
11885
11890
11895
11900
11905
11910
11915
11920
11925
11930
11935
11940
11945
11950
11955
11960
11965
11970
11975
11980
11985
11990
11995
12000
12005
12010
12015
12020
12025
12030
12035
12040
12045
12050
12055
12060
12065
12070
12075
12080
12085
12090
12095
12100
12105
12110
12115
12120
12125
12130
12135
12140
12145
12150
12155
12160
12165
12170
12175
12180
12185
12190
12195
12200
12205
12210
12215
12220
12225
12230
12235
12240
12245
12250
12255
12260
12265
12270
12275
12280
12285
12290
12295
12300
12305
12310
12315
12320
12325
12330
12335
12340
12345
12350
12355
12360
12365
12370
12375
12380
12385
12390
12395
12400
12405
12410
12415
12420
12425
12430
12435
12440
12445
12450
12455
12460
12465
12470
12475
12480
12485
12490
12495
12500
12505
12510
12515
12520
12525
12530
12535
12540
12545
12550
12555
12560
12565
12570
12575
12580
12585
12590
12595
12600
12605
12610
12615
12620
12625
12630
12635
12640
12645
12650
12655
12660
12665
12670
12675
12680
12685
12690
12695
12700
12705
12710
12715
12720
12725
12730
12735
12740
12745
12750
12755
12760
12765
12770
12775
12780
12785
12790
12795
12800
12805
12810
12815
12820
12825
12830
12835
12840
12845
12850
12855
12860
12865
12870
12875
12880
12885
12890
12895
12900
12905
12910
12915
12920
12925
12930
12935
12940
12945
12950
12955
12960
12965
12970
12975
12980
12985
12990
12995
13000
13005
13010
13015
13020
13025
13030
13035
13040
13045
13050
13055
13060
13065
13070
13075
13080
13085
13090
13095
13100
13105
13110
13115
13120
13125
13130
13135
13140
13145
13150
13155
13160
13165
13170
13175
13180
13185
13190
13195
13200
13205
13210
13215
13220
13225
13230
13235
13240
13245
13250
13255
13260
13265
13270
13275
13280
13285
13290
13295
13300
13305
13310
13315
13320
13325
13330
13335
13340
13345
13350
13355
13360
13365
13370
13375
13380
13385
13390
13395
13400
13405
13410
13415
13420
13425
13430
13435
13440
13445
13450
13455
13460
13465
13470
13475
13480
13485
13490
13495
13500
13505
13510
13515
13520
13525
13530
13535
13540
13545
13550
13555
13560
13565
13570
```

B. Java Code

```

/*
 * Grabs the complete Stateflow chart.
 * @return Stateflow chart as <code>StateChart</code>
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowGrabbberException if an error is encountered
 * by Matlab.
 */
public static StateChart grabStateflow() throws IOException {
    MatlabstateflowGrabbberException e;
    fromMatlab = MatlabProcessInfo.fromMatlabStream;
    toMatlab = MatlabProcessInfo.toMatlabStream;
    matlabPrompt = MatlabStateflowProperties.getMatlabPromptString();
    matlabErrorIndicator = MatlabStateflowProperties.getMatlabErrorIndicatorString();
    containsSubcharts = false;
    chartBoundary[0] = 0;
    chartBoundary[1] = 0;

    if (log == null) {
        try {
            log = new LogFile(new FileWriter(
                System.getProperty("user.home") + File.separator
                + ".kiel" + File.separator
                + "MatlabStateflowGrabber.log"),
                "MatlabStateflowGrabber");
        } catch (IOException e) {
            log = new LogFile(new PrintWriter(System.out),
                "MatlabStateflowGrabber");
            log.log(0, "Could not open log file!");
        }
        log.setLevel(0);
        log.enableLog();
        log.setModuleName("MatlabStateflowGrabber");
    }
}

1670
1680
1690
1700
1710
1720

    currentCharName = getObjectName("chart");
    parent.setName(currentCharName);
    map.put(getObjectID("chart"), parent);
    //int[] charSize = getchartSize();
    //rootLayout.setwidth(chartSize[0]);
    //rootLayout.setHeight(chartSize[1]);
    //actualView.setLayoutInformation(parent, rootLayout);

    /*Get all Events and Variables
     * which are children of Stateflow.Chart
     */
    log.log(0, "Grabbing output and local events:");
    toMatlab.write("inputEvents=chart" + GET_INPUTEVENTS_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();

    checkMatlabResponse();
    toMatlab.write("outputEvents=chart" + GET_OUTPUTEVENTS_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();

    checkMatlabResponse();
    toMatlab.write("localEvents=chart" + GET_LOCALEVENTS_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();

    checkMatlabResponse();
    toMatlab.write("arraySizeOf=inputEvents");
    toMatlab.newLine();
    toMatlab.flush();

    int noe = getArraySizeOf("inputEvents");
    Event[] events = new Event[noe];
    int port;
    String eventString;
    String trigger;
    for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
        eventString = "inputEvents[" + (eventNumber + 1) + "]";
        port = getPortNumber(eventString);
        trigger = getTriggerType(eventString);
        events[eventNumber] = new IntegerSignal(
            getObjectName(eventString),
            new IntegerConstant(0), trigger, port);
    }
    stc.setInputEvents(events);

    noe = getArraySizeOf("outputEvents");
    events = new Event[noe];
    for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
        eventString = "outputEvents[" + (eventNumber + 1) + "]";
        port = getPortNumber(eventString);
        trigger = getTriggerType(eventString);
        events[eventNumber] = new IntegerSignal(
            getObjectName(eventString),
            new IntegerConstant(0), trigger, port);
    }
    stc.setOutputEvents(events);

    noe = getArraySizeOf("localEvents");
    events = new Event[noe];
    for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
        eventString = "localEvents[" + (eventNumber + 1) + "]";
        port = getPortNumber(eventString);
        trigger = getTriggerType(eventString);
        events[eventNumber] = new IntegerSignal(
            getObjectName(eventString),
            new IntegerConstant(0), trigger, port);
    }
    stc.setLocalEvents(events);

    ORState parent = new ORState();
    //parent.setName("root");
    stc.setRootNode(parent);
    CompositeStateLayoutInformation rootLayout =
        new CompositeStateLayoutInformation();
    rootLayout.setUpLeftPoint(new Point(0, 0));
    toMatlab.write(GET_ROOT_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    toMatlab.write(GET_MODELS_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    toMatlab.write(GET_CHART_COMMAND);
    toMatlab.newLine();
    toMatlab.flush();
}

```

```

for (int eventNumber = 0; eventNumber < noe; eventNumber++) {
    eventsString = "localEvents[" + (eventNumber + 1) + "]";
    events[eventNumber] = new Event(
        getObjectByName(eventString));
}
stc.setLocalEvents(events);

ArrayList<eventsCol> = new ArrayList();
for (int i = 0; i < events.length; i++) {
    eventsCol.add(events[i]);
}
stc.getRootNode().addLocalEvents(eventsCol);
getAndCreateVariables(stc);

/* Get all states out of Matlab level by level of containment */
toMatlab.write("parent_states=chart" + GET_STATES_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();
toMatlab.write("parent=chart");
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();
grabOneLevel();
}

int nos;
String stateString;
boolean first = true;
nos = getArraySizeOf("parent_states");
do {
    toMatlab.write("nextLevelParent_states=[[]");
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();

    for (int stateNumber = 0; stateNumber < nos; stateNumber++) {
        stateString = "parent_states(" + (stateNumber + 1) + ")";
        if (!isSimpleState(stateString)) {
            toMatlab.write("parent=" + stateString);
            toMatlab.newLine();
            toMatlab.flush();
            checkMatlabResponse();
            grabOneLevel();
            if (first) {
                toMatlab.write("nextLevelParent_states=states");
                toMatlab.newLine();
                toMatlab.flush();
                checkMatlabResponse();
                first = false;
            }
        }
    }
} while ((nos = getArraySizeOf("nextLevelParent_states")) > 0);

rootLayout.setWidth(chartBoundary[0] + 15);
rootLayout.setHeight(chartBoundary[1] + 15);
actualView.setLayoutInformation(parent, rootLayout);
//map.clear(); // we need the map in the simulator!!!
closeLogFile();
return stc;
}

/**
 * Returns the previously constructed view. If the grabbed chart contains
 * subcharts, then no view will be returned.
 * @return the view that results from importing the chart, null if the
 * chart contains subcharts.
 */
public static View getView() {
    if (containsSubcharts) {
        return null;
    } else {
        return actualView;
    }
}

public static void closeLogFile() {
    /*
     * Closes the log file.
     */
    public static void closeLogFile() {
        if (log != null) {
            log.close();
            log = null;
        }
    }
}

if (first) {
    toMatlab.write("nextLevelParent_states=states");
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    first = false;
}
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890

```

B.5.2. MatlabStateflowGrabberException

```
//$Id: MatlabStateflowGrabberException.java,v 1.1 2005/06/30 16:22:03 apo Exp $  
package kiel.util;  
/*  
 * @author apo  
 * @version $Revision: 1.1 $ last modified $Date: 2005/06/30 16:22:03 $  
 */  
  
public class MatlabStateflowGrabberException extends Exception {  
    public MatlabStateflowGrabberException(final String message) {  
        super(message);  
    }  
}
```

B. Java Code

B.5.3. MatlabStateflowCreator

```

//$Id: MatlabStateflowCreator.java,v 1.8 2005/12/15 13:31:12 apo Exp $
/* @author apo
 * @version $Revision: 1.8 $ last modified $Date: 2005/12/15 13:31:12 $
 */
public final class MatlabStateflowCreator {
    /**
     * The Matlab prompt, usually '>>'.
     */
    private static String matlabPrompt;

    /**
     * The string that indicates an error in Matlab, usually '???'.
     */
    private static String matlabErrorIndicator;

    /**
     * The buffered reader used to read Matlab's output.
     */
    private static BufferedReader fromMatlab;

    /**
     * The buffered writer used to send commands to Matlab.
     */
    private static BufferedWriter toMatlab;

    /**
     * The view which includes the graphical information (layout) of the
     * currently processed chart.
     */
    private static View actualView;

    /**
     * The last answer of Matlab.
     */
    private static String input = "n";

    /**
     * The file the log is written into.
     */
    private static Logfile log = null;

    /**
     * The currently processed chart.
     */
    private static Statechart chart;

    /**
     * The map that is used to map between Stateflow objects and
     * KIEL objects.
     */
    private static HashMap map = new HashMap();

    /**
     * The command to close all Simulink models.
     */
}

package kiel.util;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import kiel.util.iterator;
import kiel.datastructure.ANDState;
import kiel.datastructure.CompositeState;
import kiel.datastructure.Constant;
import kiel.datastructure.History;
import kiel.datastructure.InitialState;
import kiel.datastructure.Junction;
import kiel.datastructure.Node;
import kiel.datastructure.Region;
import kiel.datastructure.State;
import kiel.datastructure.StateChart;
import kiel.datastructure.StringLabel;
import kiel.datastructure.StringVariable;
import kiel.datastructure.Transition;
import kiel.datastructure.Variable;
import kiel.datastructure.action.Actions;
import kiel.datastructure.booleanp.BooleanFalse;
import kiel.datastructure.booleanp.BooleanTrue;
import kiel.datastructure.booleanp.BooleanVariable;
import kiel.datastructure.doublexp.DoubleConstant;
import kiel.datastructure.doublexp.DoubleVariable;
import kiel.datastructure.eventxp.Event;
import kiel.datastructure.eventxp.IntegerSignal;
import kiel.datastructure.eventxp.FloatConstant;
import kiel.datastructure.floatxp.FloatVariable;
import kiel.datastructure.int16exp.Integer16Constant;
import kiel.datastructure.int16exp.Integer16Variable;
import kiel.datastructure.int8exp.Integer8Constant;
import kiel.datastructure.int8exp.Integer8Variable;
import kiel.datastructure.intexp.IntegerConstant;
import kiel.datastructure.intexp.IntegerVariable;
import kiel.datastructure.uint16exp.UInt16Constant;
import kiel.datastructure.uint16exp.UInt16Variable;
import kiel.datastructure.uint32exp.UInt32Constant;
import kiel.datastructure.uint32exp.UInt32Variable;
import kiel.datastructure.uint8exp.UIntConstant;
import kiel.datastructure.uint8exp.UIntVariable;
import kiel.graphicalinformations.EdgeLayoutInformation;
import kiel.graphicalinformations.LabelLayoutInformation;
import kiel.graphicalinformations.NodeLayoutInformation;
import kiel.graphicalinformations.PathElement;
import kiel.graphicalinformations.Point;
import kiel.graphicalinformations.View;
import kiel.graphicalinformations.View;

```

B. Java Code

```

private static final String CLOSE_OPEN_COMMAND = "sfexit";
* The suffix to access the radius of a junction.
*/
/* The command to create a new Simulink model with one Stateflow chart.
*/
private static final String CREATE_NEW_CHART_COMMAND = "sfnew";
** The command to obtain the Stateflow root object.
*/
120   private static final String GET_ROOT_COMMAND = "rt=sfroot";
** The command to obtain the Simulink model.
*/
130   private static final String GET_MODEL_COMMAND =
      "model=rt.find('-isa','Simulink.BlockDiagram')";
** The command to obtain the Stateflow chart contained in the model.
*/
140   private static final String GET_CHART_COMMAND =
      "chart=model.find('-isa','Stateflow.Chart')";
** The suffix to access the type of a variable.
*/
150   private static final String DATATYPE =
      ".DataType";
** The suffix to access the unique identifier of a Stateflow object.
*/
160   private static final String ID =
      ".Id";
** The suffix to access the name of a Stateflow object.
*/
170   private static final String NAME =
      ".Name";
** The suffix to access the label of a state of transition.
*/
private static final String LABEL =
      ".LabelString";
** The suffix to access the position of a transition label.
*/
180   private static final String POSITION =
      ".Position";
** The suffix to access the position of a graphical Stateflow object.
*/
190   private static final String JUNC_POS =
      ".JUNC_POS";
** The suffix to access the center coordinates of a junction.
*/
200   private static final String CENTER =
      ".Position.Center";
** The suffix to obtain the port of an input or output event.
*/
210   private static final String PORT =
      ".Port";
** The suffix to obtain the trigger type of an input event.
*/
220   private static final String TRIGGER =
      ".Trigger";
** The suffix to access the trigger type of an input event.
*/
230   private static final String JUNC_RADIUS =
      ".Position.Radius";
** The suffix to access the position and size of the window that contains the chart.
*/
240   private static final String CHART_SIZE =
      ".WindowPosition";
** The suffix to access the position and size of a transition.
*/
250   private static final String SRC =
      ".Source";
** The suffix to access the source end point of a transition.
*/
260   private static final String SRC_EP =
      ".SourceEndPoint";
** The suffix to access the destination of a transition.
*/
270   private static final String DEST =
      ".Destination";
** The suffix to access the o'Clock position of a transition source.
*/
280   private static final String SRC_OCLK =
      ".SourceOClock";
** The suffix to access the o'Clock position of a transition destination.
*/
290   private static final String DEST_OCLK =
      ".DestinationOClock";
** The suffix to access the decomposition type of a state.
*/
300   private static final String DECOMP0 =
      ".Decomposition";
** The suffix to access the type of a transition (connective, history).
*/
310   private static final String TYPE =
      ".Type";
** The suffix to obtain the part of an input or output event.
*/
320   private static final String PORT_TYPE =
      ".PortType";
** The suffix to access the trigger type of an input event.
*/
330   private static final String TRIGGER_TYPE =
      ".TriggerType";

```

```

    */
    * The suffix to access the scope of a variable (local, input, or output).
    */
private static final String SCOPE = ".Scope";
    /**
     * The suffix to access the initial value of a variable.
    */
private static final String INITIAL_VALUE = ".props.InitialValue";
    /**
     * The constructor for states.
    */
private static final String CREATE_STATE = "Stateflow.State(";
    /**
     * The constructor for transitions.
    */
private static final String CREATE_TRANS = "Stateflow.Transition(";

    /**
     * The constructor for junctions.
    */
private static final String CREATE_JUNC = "Stateflow.Junction(";
    /**
     * The constructor for events.
    */
private static final String CREATE_EVENT = "Stateflow.Event(";
    /**
     * The suffix to check whether a state intersects with
     * one ore more other states.
    */
private static final String BAD_INTERSECTION = ".BadIntersection";
    /**
     * Size of the input buffer (for receiving Matlab's answer).
    */
private static final int BUFF_SIZE = 20;
    /**
     * Utility classes should not have a public or default constructor.
    */
private MatlabStateflowCreator() { }

    /**
     * Receives output from Matlab. This method uses a buffered reader
     * that is connected to Matlab's standard output. The stream is read
     * until a Matlab prompt is encountered. Matlab's answer is stored
     * in the global variable <code>input</code>. If an error is indicated
     * by Matlab, then an exception is thrown.
     * @throws MatlabStateflowCreatorException if an error occurs in the input or output stream.
     * @throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
    */
private static int[] getObjectPosition(final String obj)
}

```

B. Java Code

```

throws IOException, MatlabStateflowCreatorException {
    String[] answer = handleMatlabCommand(obj + POSITION);
    int[] pos = new int[4];
    pos[0] = (int) Float.parseFloat(answer[2]);
    pos[1] = (int) Float.parseFloat(answer[3]);
    pos[2] = (int) Float.parseFloat(answer[4]);
    pos[3] = (int) Float.parseFloat(answer[5]);
    360    return pos;
}

/**
 * Obtains the port number of an event or data object.
 * @param eventOrData - the event or data object for which one wants to
 * know the port number.
 * @return the port number of the specified event or data.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static int getPortNumber(final String eventOrData)
throws IOException, MatlabStateflowCreatorException {
    String[] answer = handleMatlabCommand(eventOrData + PORT);
    370    return Integer.parseInt(answer[2]);
}

/**
 * Obtains the port number of all signals specified by the parameter
 * <code>col</code>. This method sets the <code>port</code> property
 * of the signals to the obtained values.
 * @param col - the collection of signals for which one wants to
 * know the port numbers.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void getAllPortNumbers(final Collection col)
throws IOException, MatlabStateflowCreatorException {
    Iterator it = col.iterator();
    380    IntegerSignal signal;
    while (it.hasNext()) {
        signal = (IntegerSignal) it.next();
        signal.setPort(getPortNumber(signal.getName()
            + " " + signal.hashCode()));
    }
}

/**
 * Obtains the unique identifier of the Stateflow object
 * specified by <code>obj</code>.
 * @param obj - the object for which one wants to know the id.
 * @return the id of obj.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static Integer getObjectID(final String obj)
throws IOException, MatlabStateflowCreatorException {
    String[] answer = handleMatlabCommand(obj + ID);
    390    return new Integer(answer[2]);
}

/**
 * Sets the name of the Stateflow object specified by
 * the <code>obj</code> argument to <code>name</code>.
 * @param obj - the object one wants to set the name of
 * @param name - the name of the object.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void setObjectName(final String obj, final String name)
throws IOException, MatlabStateflowCreatorException {
    String command = obj + NAME + "=" + name + ";" ;
    logLog(0, "SetObject[" + obj + ":" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Sets the label of a state or of a transition specified by
 * the <code>obj</code> argument to <code>label</code>.
 * @param obj - the state or transition one wants to set the label of.
 * @param label - the label of obj.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void setObjectLabel(final String obj, final String label)
throws IOException, MatlabStateflowCreatorException {
    String command = obj + LABEL + "=%printf(\"%s\" + label + \"\");";
    logLog(0, "SetObject[" + obj + ":" + label + ":" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Sets the position of the label of the object specified by
 * the <code>obj</code> argument to <code>pos</code>.
 * @param obj - the object (state or transition) of which one
 * wants to set the label position [x, y, width, height].
 * @param pos - the coordinates of the label of the object obj.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void setObjectLabelPosition(final String obj,
final int[] pos)
throws IOException, MatlabStateflowCreatorException {
    String command = obj + LABEL_POS
        + "[[" + pos[0] + " " + pos[1] + " " + pos[2] + " " + pos[3] + " " + pos[4]];
    logLog(0, "SetObject[" + obj + ":" + command);
    toMatlab.write(command);
    toMatlab.newLine();
}

private static Integer getObjectId(final String obj)
throws IOException, MatlabStateflowCreatorException {
    400    410    String[] answer = handleMatlabCommand(obj + ID);
}

```

```

        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the position of the object specified by
     * the argument <code><code> to <code><code>.
     * Param obj - the object of which one wants to set the position.
     * Param pos - the coordinates of the object [x, y, width, height].
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setObjectPosition(final String obj, final int[] pos)
        throws IOException, MatlabStateflowCreatorException {
        String command = obj + POSITION
            + "[" + pos[0] + " "
            + pos[1] + " "
            + pos[2] + " "
            + pos[3] + " ";
        log.log(0, "SetObjectPosition:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the scope of the variable or event specified by
     * the <code><code> argument to <code><code>.
     * Param obj - the variable one wants to set the scope of.
     * Param scope - the scope of the variable or event
     * (input, output, local).
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setObjectScope(final String obj, final String scope)
        throws IOException, MatlabStateflowCreatorException {
        String command = obj + SCOPE + "=" + scope + ";";
        log.log(0, "SetObjectScope:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the source of the transition specified by <code><code>
     * to <code><code>.
     * Param trans - the transition for which one wants to set the source.
     * Param src - the source of the transition <code><code>.
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setTransitionSource(final String trans,
                                         final String src)
        throws IOException, MatlabStateflowCreatorException {
        String command = trans + SRC + "=" + src + ";";
        log.log(0, "SetTransitionSource:" + command);
        toMatlab.write(command);
    }

    /**
     * Sets the destination of the transition specified
     * by <code><code> to <code><code>.
     * Param trans - the transition of which one wants to set
     * the destination.
     * Param dest - the destination of the transition <code><code>.
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setTransitionDestination(final String trans,
                                                final String dest)
        throws IOException, MatlabStateflowCreatorException {
        String command = trans + DEST + "=" + dest + ";";
        log.log(0, "SetTransitionDestination:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Calculates and sets the o'clock value according to the source end
     * point of the transition specified by <code><code> and
     * <code><code>.
     * Param trans - the transition of which one wants to set
     * the source end point.
     * Param sep - the source end point [x, y].
     * Param srcState - the state that is the source of the transition.
     * Param parentPos - the position of the parent state of <code><code>srcState
     * </code>.
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setTransitionSourceEndPoint(final String trans,
                                                 final int[] sep,
                                                 final Node srcState,
                                                 final int[] parentPos)
        throws IOException, MatlabStateflowCreatorException {
        NodarLayoutInformation nli = actualView.getLayoutInformation(srcState);
        int[] center = {parentPos[0] + nli.getWidth() / 2,
                      parentPos[1] + nli.getHeight() / 2};
        double c = Math.sqrt((double) ((sep[0] - center[0])
                                       * (sep[0] - center[0]) +
                                       (sep[1] - center[1]) * (sep[1] - center[1])));
        double oclockRad = Math.atan2(sep[1] - center[1], sep[0] - center[0]);
        oclockRad += Math.PI / 2;
        if (oclockRad < 0) {
            oclockRad = (2 * Math.PI) + oclockRad;
        }
        double oclock = oclockRad / (2 * Math.PI / 12);
        String command = trans + SRC_OCLOCK + "=" + oclock + ";";
        log.log(0, "SetTransitionSourceEndPoint:" + trans + SRC_EP + "="
               + oclock);
    }

    /**
     * Sets the source of the transition specified by <code><code>
     * to <code><code>.
     * Param trans - the transition for which one wants to set the source.
     * Param src - the source of the transition <code><code>.
     * Throws IOException if an error occurs in the input or output stream.
     * Throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setTransitionSource(final String trans,
                                         final String src)
        throws IOException, MatlabStateflowCreatorException {
        String command = trans + SRC + "=" + src + ";";
        log.log(0, "SetTransitionSource:" + command);
        toMatlab.write(command);
    }
}

```

B. Java Code

```

    * param parentPos - the position of the parent state of <code><code></code></code>
    * </code>
    * throws IOException if an error occurs in the input or output stream.
    * throws MatlabStateException if an error is encountered
    * by Matlab.
    */
600   private static void setTransitionEndPoint(final String trans,
                                             final int[] dep,
                                             final Node destState,
                                             final int[] parentPos)
                                         throws IOException, MatlabStateflowCreatorException {
    NodalLayoutInformation nli = actualView.getLayoutInformation(destState);
    int[] center = {parentPos[0] + nli.getXPos() + nli.getWidth() / 2,
                    parentPos[1] + nli.getYPos() + nli.getHeight() / 2};
    double c = Math.sqrt((double) ((dep[0] - center[0]) *
                                    (dep[0] - center[0]) + (dep[1] - center[1]) *
                                    (dep[1] - center[1])) * (dep[1] - center[1]));
    double oclockRad = Math.atan2(dep[1] - center[1], dep[0] - center[0]);
    oclockRad += Math.PI / 2;
    if (oclockRad < 0) {
        oclockRad = (2 * Math.PI) + oclockRad;
    }
    double oclockRad = oclockRad / (2 * Math.PI / 12);
}

/**
 * Sets the transition source end point directly without calculating
 * the o'clock value.
 * param trans - the transition of which one wants to set
 * the source end point.
 * param sep - the source end point [x, y].
 * throws IOException if an error occurs in the input or output stream.
 * throws MatlabStateException if an error is encountered
 * by Matlab.
 */
610   private static void setTransitionSourceEndPointDirect(final String trans,
                                                       final int[] sep,
                                                       final String command);
                                         throws IOException, MatlabStateflowCreatorException {
    String command = trans + SRC_EP + "[" + sep[0] + "," + sep[1] + "]";

    log.log(0, "Set transition source end point:" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Sets the midpoint of the transition specified by <code><code></code></code>
 * param trans - the transition of which one wants to set the midpoint.
 * param mip - the midpoint [x, y].
 * throws IOException if an error occurs in the input or output stream.
 * by Matlab.
 */
620   private static void setTransitionMidPoint(final String trans,
                                              final int[] mip)
                                         throws IOException, MatlabStateflowCreatorException {
    String command = trans + TR_MIDPOINT + "=" +
                    "[" + mip[0] + "," + mip[1] + "]";

    log.log(0, "Set transition midpoint:" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Calculates and sets the destination o'clock value of the transition
 * specified by <code><code></code></code>
 * param trans - the transition for which one wants to know the
 * destination end point.
 * param destState - the state that is the destination of the transition
 * specified by <code><code></code></code>
 * param dep - the destination end point [x, y].
 */
630   private static void setTransitionDestinationEndPoint(final String trans,
                                                       final int[] pos)
                                         throws IOException, MatlabStateflowCreatorException {
    String command = trans + TR_DESTPOINT + "=" +
                    "[" + pos[0] + "," + pos[1] + "]";

    log.log(0, "Set transition destination point:" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Sets the radius of the junction specified by the argument
 * <code><code></code></code>
 * param junc - the junction for which one wants to know the position.
 * param pos - the position of the junction specified by the argument.
 * throws IOException if an error occurs in the input or output stream.
 * by Matlab.
 */
640   private static void setJunctionPosition(final String junc,
                                           final int[] pos)
                                         throws IOException, MatlabStateflowCreatorException {
    String command = junc + JUNC_POS + "=" +
                    "[" + pos[0] + "," + pos[1] + "]";

    log.log(0, "Set junction position:" + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
}

/**
 * Sets the radius of the junction specified by the argument
 * <code><code></code></code>
 * param junc - the junction for which one wants to know the radius.
 * param rad - the radius of the specified junction.
 */
650
}

```

```

    * @throws MatlabStateflowCreatorException if an error is encountered
    * by Matlab.
    */
    private static void setJunctionRadius(final String junc, final int rad)
        throws IOException, MatlabStateflowCreatorException {
        String command = junc + JUNC_RADIUS + "==" + rad + ";\n";
        log.log(0, "Setjunction\u2022radius:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the initial value of the variable specified by the argument
     * <code><obj></code> to <code><val></code>
     * @param obj - the variable of which one wants to set the initial value.
     * @param val - the initial value of the variable <code><obj></code>.
     * @throws IOException if an error occurs in the input or output stream.
     * @throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setInitialValue(final String obj, final String val)
        throws IOException, MatlabStateflowCreatorException {
        String command = obj + INITIAL_VALUE + "==" + val + ";\n";
        log.log(0, "Setinitial\u2022valueof\u2022variable:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the trigger type of the event specified by the argument
     * <code><event></code> to <code><type></code>.
     * @param event - the event of which one wants to know the trigger type.
     * @param type - the trigger type of the specified event
     * (rising, falling, either, function).
     * @throws IOException if an error occurs in the input or output stream.
     * @throws MatlabStateflowCreatorException if an error is encountered
     * by Matlab.
     */
    private static void setTriggerType(final String event, final String type)
        throws IOException, MatlabStateflowCreatorException {
        String command = event + TRIGGER + "==" + type + ";\n";
        log.log(0, "Settrigger\u2022type\u2022of\u2022signal:" + command);
        toMatlab.write(command);
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Sets the data type of the variable specified by
     * the <code><data></code> argument to <code><type></code>
     * @param data - the variable one wants to know the data type of
     * @param type - the data type of the variable (int32, int16, int8,
     * single, double, boolean, unsigned integer (32, 16, 8 bit),
     * @throws IOException if an error occurs in the input or output stream.
     */
}

```

B. Java Code

```

private static boolean hasBadIntersection(final String state)
throws IOException, MatlabStateflowCreatorException {
    String[] answer = handleMatlabCommand(state + BAD_INTERSECTION);
    return (answer[2].equals("1"));
}

<**
 * Creates the variables in Stateflow that are specified by the parameter
 * <code><code>
 * <param col - the variables that have to be created.
 * <param parent - the handle of the parent state that
 * is expected to contain the variables.
 * <param scope - the scope of the variables (local, input, output).
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void createVariables(final Collection<Object> col,
final State parent,
final String scope)
throws IOException, MatlabStateflowCreatorException {
    Object item;
    Variable variable;
    String name;
    String identifier;
    String parentIdentifier = parent.getName() + "__" + parent.hashCode();
    Iterator<Object> it = col.iterator();
    while (it.hasNext()) {
        item = it.next();
        variable = (Variable) item;
        name = variable.getName();
        identifier = name + "__" + variable.hashCode();
        createObject(identifier, parentIdentifier, CREATE_VARIABLE);
        setObjectName(identifier, name);
        if (item.getClass().equals(IntegerVariable.class)) {
            setDataType(identifier, "int32");
        } else if (item.getClass().equals(Integer6Variable.class)) {
            setDataType(identifier, "int16");
        } else if (item.getClass().equals(Integer8Variable.class)) {
            setDataType(identifier, "int8");
        } else if (item.getClass().equals(UInt32Variable.class)) {
            setDataType(identifier, "uint32");
        } else if (item.getClass().equals(UInt6Variable.class)) {
            setDataType(identifier, "uint16");
        } else if (item.getClass().equals(UInt8Variable.class)) {
            setDataType(identifier, "uint8");
        } else if (item.getClass().equals(FloatVariable.class)) {
            setDataType(identifier, "single");
        } else if (item.getClass().equals(DoubleVariable.class)) {
            setDataType(identifier, "double");
        } else if (item.getClass().equals(BooleanVariable.class)) {
            setDataType(identifier, "boolean");
            if (variable.toExpString().equals("true")) {
                setInitialValue(identifier, "1");
            } else {
                setInitialValue(identifier, "0");
            }
            setObjectScope(identifier, scope);
        }
    }
}

<**
 * Creates the constants in Stateflow that are specified by the parameter
 * <code><code>
 * <param col - the constants that have to be created.
 * <param parent - the handle of the parent state that
 * is expected to contain the constants.
 * <throws IOException if an error occurs in the input or output stream.
 * <throws MatlabStateflowCreatorException if an error is encountered
 * by Matlab.
 */
private static void createConstants(final Collection<Object> col,
final State parent)
throws IOException, MatlabStateflowCreatorException {
    Object item;
    Constant constant;
    String name;
    String identifier;
    String parentIdentifier = parent.getName() + "__" + parent.hashCode();
    Iterator<Object> it = col.iterator();
    while (it.hasNext()) {
        item = it.next();
        constant = (Constant) item;
        name = constant.getName();
        identifier = name + "__" + constant.hashCode();
        createObject(identifier, parentIdentifier, CREATE_VARIABLE);
        setObjectName(identifier, name);
        if (item.getClass().equals(IntegerConstant.class)) {
            setDataType(identifier, "int32");
        } else if (item.getClass().equals(Integer16Constant.class)) {
            setDataType(identifier, "int16");
        } else if (item.getClass().equals(Integer6Constant.class)) {
            setDataType(identifier, "int6");
        } else if (item.getClass().equals(Integer8Constant.class)) {
            setDataType(identifier, "int8");
        } else if (item.getClass().equals(UInt32Constant.class)) {
            setDataType(identifier, "uint32");
        } else if (item.getClass().equals(UInt16Constant.class)) {
            setDataType(identifier, "uint16");
        } else if (item.getClass().equals(UInt8Constant.class)) {
            setDataType(identifier, "uint8");
        } else if (item.getClass().equals(FloatConstant.class)) {
            setDataType(identifier, "float");
        } else if (item.getClass().equals(DoubleConstant.class)) {
            setDataType(identifier, "double");
        } else if (item.getClass().equals(BooleanConstant.class)) {
            setDataType(identifier, "boolean");
            if (variable.toExpString().equals("true")) {
                setInitialValue(identifier, "1");
            } else {
                setInitialValue(identifier, "0");
            }
            setObjectScope(identifier, scope);
        }
    }
}

```


B. Java Code

```

String sourceIdentifier;
String destinationIdentifier;
State state;
Transition trans;
Node sourceNode;
Node destinationNode;
Transition transition;
Actions actions;
1080

if (parent.getName().length() == 0) {
    label += "\nentry:" + actions.toString() + " ";
}

actions = state.getDoActivity();
if (actions.toString().length() > 0) {
    label += "\nduring:" + actions.toString() + " ";
}

actions = state.getExit();
if (actions.toString().length() > 0) {
    label += "\nexit:" + actions.toString() + " ";
}

actions = state.getInternalTransition();
if ((StringLabel) trans.getLabel()).toString().length() > 0) {
    label += "\n" + ((StringLabel) trans.getLabel()).toString() + " ";
}

trans = state.getBindAction();
if (((StringLabel) trans.getLabel()).toString().length() > 0) {
    label += "\n" + ((StringLabel) trans.getLabel()).toString() + " ";
}

setObjectLabel(identifier, label);

if (state instanceof ANDstate) {
    handleMatlabCommand(identifier + DECOMP0 + "='PARALLEL_AND' ");
}
else if (state instanceof Region) {
    resolveBadIntersection(identifier);
}
getAndCreateLocalEvents(state);
getAndCreateLocalVariables(state);
}

int noj = junctions.size();
1170
int pos = new int[2];
Object item;
for (int i = 0; i < noj; i++) {
    pos = junctions.get(i);
    nli = actualView.getLayoutInformation((Node) item);
    pos[0] = nli.getPos() + parentPos[0] + nli.getWidth();
    pos[1] = nli.getPos() + parentPos[1] + nli.getWidth();
    createObject("hist" + item.hashCode(),
    parentIdentifier, CREATE_JUNC);
    handleMatlabCommand("hist" + item.hashCode()
    + TYPE + "'HISTORY';");
    setJunctionPosition("hist" + item.hashCode(),
    setJunctionRadius("hist" + item.hashCode(), pos);
}
else {
    createObject("junc" + item.hashCode(),
    parentIdentifier, CREATE_JUNC);
    handleMatlabCommand("junc" + item.hashCode()
    + TYPE + "'CONNECTIVE';");
    setJunctionPosition("junc_" + item.hashCode(), pos);
}

1190
label = name;
actions = state.getEntry();
1190

```

```

setJunctionRadius("junc " + item.getItemId(), sep);
}
}
nii.getWidth() / 2);
}

transitions = getTransitions(parent);
int notes = transitions.size();
Point point;
ArrayList path;
Point point;
int[] sep = new int[2];
int[] mip = new int[2];
int[] dep = new int[2];
int transitionNumber = 0; transitionNumber < notes;
transitionNumber++;
transition = (Transition) transitions.get(transitionNumber);
name = ((StringLabel) transition.getLabel()).toString();
name = name.replaceAll("\n", "\\\n");
identifier = "trans" + transition.hashCode();
createObject(identifier, parentIdentifier, CREATE_TRANS);
setSubjectLabel(identifier, name);
sourceCellode = (Node) transition.getSource();
if (sourceNode instanceof Junction) {
    sourceIdentifier = "junc " + sourceNode.getName()
        + sourceCellode.hashCode();
} else {
    if (sourceNode.getName().length() == 0) {
        sourceIdentifier = "state_" + sourceNode.hashCode();
    } else {
        sourceIdentifier = sourceNode.getName() + "_"
            + sourceCellode.hashCode();
    }
}
destinationNode = (Node) transition.getTarget();
if (destinationNode instanceof Junction) {
    destinationIdentifier = "junc_" + destinationNode.getName()
        + destinationNode.hashCode();
} else {
    if (destinationNode.getName().length() == 0) {
        destinationIdentifier = "state_"
            + destinationNode.hashCode();
    } else {
        destinationIdentifier = destinationNode.getName() + "_"
            + destinationNode.hashCode();
    }
}
eli = actualView.getLayoutInformation(transition);
path = eli.getAllPathElements();
point = ((PathElement) path.get(0)).getTargetPoint();
sep[0] = point.getX() + parentPos[0];
sep[1] = point.getY() + parentPos[1];
setTransitionEndPoint(identifier,
    destinationNode,
    parentPos);
setTransitionDestination(identifier, destinationIdentifier);
}

point = ((PathElement) path.get(path.size() - 1)).getTargetPoint();
dp[0] = point.getX() + parentPos[0];
dp[1] = point.getY() + parentPos[1];
setTransitionDestinationEndPoint(identifier,
    dep,
    destinationNode,
    parentPos);
point = ((PathElement) path.get(path.size() / 2)).getTargetPoint();
mip[0] = point.getX() + parentPos[0];
mip[1] = point.getY() + parentPos[1];
//setTransitionMidPoint(identifier, mip);

lli = actualView.getLayoutInformation(transition.getLabel());
if (lli != null) {
    point = lli.getUpperLeft();
    pos = new int[4];
    pos[0] = point.getX() + parentPos[0];
    pos[1] = point.getY() + parentPos[1];
    pos[2] = lli.getWidth() + 10;
    pos[3] = lli.getHeight() + 10;
    setObjectLabelPosition(identifier, pos);
}
}
}
1260
1270
1280
1290
1300
1310
1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500
1510
1520
1530
1540
1550
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
2060
2070
2080
2090
2100
2110
2120
2130
2140
2150
2160
2170
2180
2190
2200
2210
2220
2230
2240
2250
2260
2270
2280
2290
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
2600
2610
2620
2630
2640
2650
2660
2670
2680
2690
2700
2710
2720
2730
2740
2750
2760
2770
2780
2790
2800
2810
2820
2830
2840
2850
2860
2870
2880
2890
2900
2910
2920
2930
2940
2950
2960
2970
2980
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
9999
10000
10001
10002
10003
10004
10005
10006
10007
10008
10009
100010
100011
100012
100013
100014
100015
100016
100017
100018
100019
100020
100021
100022
100023
100024
100025
100026
100027
100028
100029
100030
100031
100032
100033
100034
100035
100036
100037
100038
100039
100040
100041
100042
100043
100044
100045
100046
100047
100048
100049
100050
100051
100052
100053
100054
100055
100056
100057
100058
100059
100060
100061
100062
100063
100064
100065
100066
100067
100068
100069
100070
100071
100072
100073
100074
100075
100076
100077
100078
100079
100080
100081
100082
100083
100084
100085
100086
100087
100088
100089
100090
100091
100092
100093
100094
100095
100096
100097
100098
100099
1000100
1000101
1000102
1000103
1000104
1000105
1000106
1000107
1000108
1000109
1000110
1000111
1000112
1000113
1000114
1000115
1000116
1000117
1000118
1000119
1000120
1000121
1000122
1000123
1000124
1000125
1000126
1000127
1000128
1000129
1000130
1000131
1000132
1000133
1000134
1000135
1000136
1000137
1000138
1000139
1000140
1000141
1000142
1000143
1000144
1000145
1000146
1000147
1000148
1000149
1000150
1000151
1000152
1000153
1000154
1000155
1000156
1000157
1000158
1000159
1000160
1000161
1000162
1000163
1000164
1000165
1000166
1000167
1000168
1000169
1000170
1000171
1000172
1000173
1000174
1000175
1000176
1000177
1000178
1000179
1000180
1000181
1000182
1000183
1000184
1000185
1000186
1000187
1000188
1000189
1000190
1000191
1000192
1000193
1000194
1000195
1000196
1000197
1000198
1000199
1000200
1000201
1000202
1000203
1000204
1000205
1000206
1000207
1000208
1000209
1000210
1000211
1000212
1000213
1000214
1000215
1000216
1000217
1000218
1000219
1000220
1000221
1000222
1000223
1000224
1000225
1000226
1000227
1000228
1000229
1000230
1000231
1000232
1000233
1000234
1000235
1000236
1000237
1000238
1000239
1000240
1000241
1000242
1000243
1000244
1000245
1000246
1000247
1000248
1000249
1000250
1000251
1000252
1000253
1000254
1000255
1000256
1000257
1000258
1000259
1000260
1000261
1000262
1000263
1000264
1000265
1000266
1000267
1000268
1000269
1000270
1000271
1000272
1000273
1000274
1000275
1000276
1000277
1000278
1000279
1000280
1000281
1000282
1000283
1000284
1000285
1000286
1000287
1000288
1000289
1000290
1000291
1000292
1000293
1000294
1000295
1000296
1000297
1000298
1000299
1000300
1000301
1000302
1000303
1000304
1000305
1000306
1000307
1000308
1000309
1000310
1000311
1000312
1000313
1000314
1000315
1000316
1000317
1000318
1000319
1000320
1000321
1000322
1000323
1000324
1000325
1000326
1000327
1000328
1000329
1000330
1000331
1000332
1000333
1000334
1000335
1000336
1000337
1000338
1000339
1000340
1000341
1000342
1000343
1000344
1000345
1000346
1000347
1000348
1000349
1000350
1000351
1000352
1000353
1000354
1000355
1000356
1000357
1000358
1000359
1000360
1000361
1000362
1000363
1000364
1000365
1000366
1000367
1000368
1000369
1000370
1000371
1000372
1000373
1000374
1000375
1000376
1000377
1000378
1000379
1000380
1000381
1000382
1000383
1000384
1000385
1000386
1000387
1000388
1000389
1000390
1000391
1000392
1000393
1000394
1000395
1000396
1000397
1000398
1000399
1000400
1000401
1000402
1000403
1000404
1000405
1000406
1000407
1000408
1000409
1000410
1000411
1000412
1000413
1000414
1000415
1000416
1000417
1000418
1000419
1000420
1000421
1000422
1000423
1000424
1000425
1000426
1000427
1000428
1000429
1000430
1000431
1000432
1000433
1000434
1000435
1000436
1000437
1000438
1000439
1000440
1000441
1000442
1000443
1000444
1000445
1000446
1000447
1000448
1000449
1000450
1000451
1000452
1000453
1000454
1000455
1000456
1000457
1000458
1000459
1000460
1000461
1000462
1000463
1000464
1000465
1000466
1000467
1000468
1000469
1000470
1000471
1000472
1000473
1000474
1000475
1000476
1000477
1000478
1000479
1000479
1000480
1000481
1000482
1000483
1000484
1000485
1000486
1000487
1000488
1000489
1000489
100
```

B. Java Code

```

try {
    log = new LogFile(new FileWriter(
        System.getProperty("user.home") + File.separator
        + ".kiel" + File.separator
        + "MatlabStateflowCreator.log"));
}

} catch (IOException e) {
    log = new LogFile(new PrintWriter(System.out),
        "MatlabStateflowCreator");
    log.setLevel(0);
    log.info("Could not open log file !");
}

log.log(0, "Creating Stateflow-Chart:");

/*Create new chart in Matlab, close all open ones before*/
toMatlab.write(CLOSE_OPEN_CHARTS_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();
toMatlab.write(CREATE_NEW_CHART_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();

toMatlab.write(GET_ROOT_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();

toMatlab.write(GET_CHART_COMMAND);
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();

/*Create local variables and events*/
Collection col;
Iterator it;
IntegerSignal event;
Variable variable;
Object item;
String name;
String identifier;
String[] names;

while (it.hasNext()) {
    event = (IntegerSignal) it.next();
    name = event.getName();
    identifier = name + " " + event.hashCode();
    createObject(identifier, "chart", CREATE_EVENT);
    setObjectName(identifier, name);
    setObjectType(identifier, "Output");
    setTriggerType(identifier, event.getTrigger());
}

/*Create output events*/
col = stc.getOutputEvents();
it = col.iterator();
while (it.hasNext()) {
    event = (IntegerSignal) it.next();
    name = event.getName();
    identifier = name + " " + event.hashCode();
    createObject(identifier, "chart", CREATE_EVENT);
    setObjectName(identifier, name);
    setObjectType(identifier, "Output");
    setTriggerType(identifier, event.getTrigger());
}

/*Temporarily change root name*/
String chartName = stc.getRootNode().getQualifiedName();
toMatlab.write('RootNode' + stc.getRootNode().getQualifiedName() + "=chart;");
toMatlab.newLine();
toMatlab.flush();
checkMatlabResponse();

stc.getRootNode().setName("RootNode");

/*Create input variables*/
col = stc.getInputVariables();
createVariables(col, stc.getRootNode(), "Input");

/*Create output variables*/
col = stc.getOutputVariables();
createVariables(col, stc.getRootNode(), "Output");

/*Create local variables*/
col = stc.getLocalVariables();
createVariables(col, stc.getRootNode(), "Local");

/*Create constants*/
col = stc.getConstants();
createConstants(col, stc.getRootNode());

/*Get all port numbers. Simulink assigns port numbers to
 * all input and output signals while they are created.*/
getAllPortNumbers(stc.getRootNode());
getAllPortNumbers(stc.getOutputEvents());
int nos = ParentStates.size();
stc.getRootNode().setName(chartName);

do {
    nextLevelParentStates.clear();
    nextLevelParentStates = createOneLevel(stc.getRootNode);
    for (int i = 0; i < nos; i++) {
        nextLevelParentStates.add(new ArrayList());
    }
} 1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500
1510
1520
1530
1540
1550
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
2060
2070
2080
2090
2100
2110
2120
2130
2140
2150
2160
2170
2180
2190
2200
2210
2220
2230
2240
2250
2260
2270
2280
2290
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3798
3799
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3898
3899
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3998
3999
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4098
4099
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4189
4190
4191
4192
4193
4194
4195
4196
4197
4197
4198
4199
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4289
4290
4291
4292
4293
4294
4295
4296
4297
4297
4298
4299
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4398
4399
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4498
4499
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4579
4580
4581
4582
4583
458
```

```

public class MatlabStateflowCreatorException
    extends MatlabStateflowCreatorException {
    public MatlabStateflowCreatorException(
        String message) {
        super(message);
    }
}

public class MatlabStateflowCreatorException
    extends MatlabStateflowCreatorException {
    public MatlabStateflowCreatorException(
        String message) {
        super(message);
    }
}

// $Id: kiel.util/ 1.1 $ last modified $Date: 2005/09/16 10:18:51 $
// $Author: apo $

package kiel.util;

```

B.5.4. MatlabStateflowCreatorException

B. Java Code

B.5.5. MatlabProcessInfo

```

//$$Id: MatlabProcessInfo.java,v 1.7 2005/12/15 13:31:12 apo Exp $      * then this method does the following:
package kiel.util;                                         * First it tells Matlab to quit and waits until Matlab has terminated.
import java.io.BufferedReader;                                * If that does not work it uses Process.destroy();
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.OutputStreamWriter;

10   /**
   * @author apo
   * @version $Revision: 1.7 $ Last modified $Date: 2005/12/15 13:31:12 $
   */
public final class MatlabProcessInfo {
    /**
     * The name of the Matlab executable file name.
     */
    private static String matlabExecutableFileName;
    /**
     * Parameters for the MatlabExecutable.
     */
    private static String options = "";
    /**
     * The Matlab prompt, usually '>>'.
     */
    private static String matlabPrompt;
    /**
     * Holds a reference to the Matlab process.
     */
    public static Process matlabProcess;
    /**
     * Holds a reference to the output stream of Matlab.
     */
    public static BufferedReader fromMatlabStream;
    /**
     * Holds a reference to the input stream of Matlab.
     */
    public static BufferedWriter toMatlabStream;
    /**
     * Size of the input buffer.
     */
    private static final int BUFF_SIZE = 20;
    /**
     * Utility classes should not have a public or default constructor.
     */
    private MatlabProcessInfo() {
        /**
         * If and only if an instance of Matlab had been started by KIEL
         * Runtime.getRuntime().exec(matlabExecutableFileName + options);
         */
    }
}

60   public static void shutdownMatlab() {
    if (matlabProcess != null) {
        try {
            toMatlabStream.write("sfexit");
            toMatlabStream.newLine();
            toMatlabStream.flush();
            fromMatlabStream.skip(3);
            toMatlabStream.write("quit");
            toMatlabStream.newLine();
            toMatlabStream.flush();
        }
        catch (IOException e) {
            matlabProcess.waitFor();
            toMatlabStream.close();
            fromMatlabStream.close();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            matlabProcess.destroy();
        }
        finally {
            matlabProcess = null;
            fromMatlabStream = null;
            toMatlabStream = null;
        }
    }
}
80   /**
     * If no instance of Matlab is running, this method starts a new one.
     * @return Matlab's response.
     * @throws IOException - if something goes wrong while communicating with
     * Matlab.
     */
public static String startMatlab() throws IOException {
    String input = "u";
    char[] inputBuf = new char[BUFF_SIZE];
    int numRead = 0;
    MatlabStateflowProperties.loadProps();
    MatlabStateflowProperties.fileName =
        MatlabStateflowProperties.getMatlabExecutableFileName();
    matlabPrompt = MatlabStateflowProperties.getMatlabPromptString();
    if (MatlabStateflowProperties.startMatlabWithNoDisplay()) {
        options = "-nodisplay";
    } else {
        options = "";
    }
    if (matlabProcess == null) {
        matlabProcess =
            Runtime.getRuntime().exec(matlabExecutableFileName + options);
        fromMatlabStream =
    }
}
100
90
100
40
50
110

```

```

new BufferedHeader(new InputStreamReader(
    new MatlabStream.newLine();
    // enter a new line to force Matlab to show a prompt
    toMatlabStream = new MatlabStream.newLine();
    toMatlabStream.flush();

    input = "";
    while (input.indexOf(matlabPrompt) == -1) {
        numRead = fromMatlabStream.read(
            inputBuf, 0, inputBuf.length);
        input += new String(inputBuf, 0, numRead);
    }
    return inputValue;
}
return new String();
}
}

120
140
130

```

/ This is a workaround due to a bug in Matlab which causes
 * Matlab not to show a prompt after command has
 * finished its output.
 /

toMatlabStream.write("open_system('bla')");

B.5.6. MatlabStateflowProperties

```

//$$Id: MatlabStateflowProperties.java,v 1.3 2005/10/06 14:52:35 $

package kiel.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;

/*
 * @author apo
 * @version $Revision: 1.3 $ last modified $Date: 2005/10/06 14:52:35 $
 */
public class MatlabStateflowProperties {

    /**
     * The property class containing the values.
     */
    private static final java.util.Properties INTERNAL_PROPERTIES = new java.util.
        Properties();

    /**
     * Path to user properties.
     */
    private static final String FILE = System.getProperty("user.home")
        + File.separator + "kiel" + File.separator
        + "matlabStateflow.properties";

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

    /**
     * Loads user properties file or creates default properties
     * if no user file available.
     */
    public static void loadProps() {
        try {
            INTERNAL_PROPERTIES.load(new FileInputStream(new File(FILE)));
        } catch (IOException ex) {
            createDefaultFile();
            loadProps();
        }
    }

    private static void createDefaultFile() {
        InputStream is;
        FileOutputStream fw;
        try {
            is = MatlabStateflowProperties.class.getClassLoader().
                getResourceAsStream("kiel/simulator/matlab/matlabStateflow.properties");
            fw = new FileWriter(FILE);
            int c = is.read();
            while (c >= 0) {
                fw.write(c);
                c = is.read();
            }
            fw.flush();
            is.close();
            fw.close();
        } catch (IOException storeException) {
        }
    }

    public static boolean startMatlabWithNodeisplay() {
        String answer = INTERNAL_PROPERTIES.getProperty(
            "MatlabProcessInfo.nodisplay", "false");
        return answer.equals("true");
    }

    public static float getChartScale() {
        String answer = INTERNAL_PROPERTIES.getProperty(
            "MatlabStateflowGrubber.chartScale", "1.0");
        try {
            return Float.parseFloat(answer);
        } catch (NumberFormatException e) {
            return 1.0f;
        }
    }

    public static String getInputVariableNamePrefix() {
        return INTERNAL_PROPERTIES.getProperty(
            "StateflowSimulator.inputVariableNamePrefix", "VarMat");
    }

    public static String getMatlabPromptString() {
        return INTERNAL_PROPERTIES.getProperty(
            "matlabPrompt", ">>_");
    }

    public static String getMatlabErrorIndicatorString() {
        return INTERNAL_PROPERTIES.getProperty(
            "matlabErrorIndicator", "?");
    }

    public static String getMatlabExecutableFileName() {
        return INTERNAL_PROPERTIES.getProperty(
            "MatlabProcessInfo.matlabExecutableFileName", "matlab");
    }

    private static void createDefaultFile() {
        InputStream is;
        FileOutputStream fw;
        try {

```

B.6. kiel.simulator.matlab

B.6.1. StateflowSimulator

```

//$$Id: StateflowSimulator.java,v 1.22 2005/12/15 16:14:48 apo Exp $

package kiel.simulator.matlab;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import kiel.configMgr.Configuration;
import kiel.configMgr.MacroStep;
import kiel.configMgr.SignalAbsent;
import kiel.configMgr.SignalPresent;
import kiel.configMgr.Stateactivated;
import kiel.configMgr.Stateactivated;
import kiel.configMgr.VariableValue;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Node;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.Variable;
import kiel.dataStructure.eventesp.Event;
import kiel.dataStructure.eventesp.IntegerSignal;
import kiel.simulator.Simulator;
import kiel.simulator.SimulatorException;
import kiel.util.Benchmark;
import kiel.util.LogFile;
import kiel.util.MatlabProcessInfo;
import kiel.util.MatlabStateflowGrabber;
import kiel.util.MatlabStateflowProperties;

/*
 * @author apo
 * @version $Revision: 1.22 $ last modified $Date: 2005/12/15 16:14:48 $
 */
public class StateflowSimulator implements Simulator {
    /*
     * The Matlab prompt, usually '>>'.
     */
    private static String matlabPrompt;

    /*
     * The string that indicates an error in Matlab, usually '??'.
     */
    private static String matlabErrorIndicator;

    /*
     * The prefix used for variables in the Simulink environment.
     */
    private static String matlabPrefix;

    private static String inputVariableNamePrefix;
    /**
     * The chart that is currently simulated.
     */
    private StateChart stc;
    /**
     * The buffered reader used to read Matlab's output.
     */
    private BufferedReader fromMatlab;
    /**
     * The buffered writer used to send commands to Matlab.
     */
    private BufferedWriter toMatlab;
    /**
     * The last answer of Matlab.
     */
    private String input = "";
    /**
     * The file name of the currently processed chart.
     */
    private String fileName = "";
    /**
     * The name of the currently processed chart.
     */
    private String charName = "";
    /**
     * The file the log is written into.
     */
    private LogFile log;
    /**
     * This array associates states with output ports. The state whose
     * port number is i is stored in the array at position (i - 1).
     */
    private State[] stateActivityOutput;
    /**
     * This array holds all input signals.
     */
    private IntegerSignal[] inputSignals;
    /**
     * This array holds all output signals.
     */
    private IntegerSignal[] outputSignals;
}

40   /*
50   * The Matlab prompt, usually '>>'.
   */

```

B. Java Code

```

private int simsize;

/*
 * This array associates states and output signals with output ports.
 * The state or output signal whose port number is i is stored in the array
 * at position (i - 1).
 */
110 private Object[] simontArray;

/*
 * This array stores the actual values of all input signals. The signal
 * whose port number is i is stored at position (i - 1).
 */
115 private int[] signalValues;

private Object[] simontArray;

/*
 * This array stores the actual values of all input variables.
 */
120 private double[] inputVarsValues;

private String[] variableNames;

/*
 * This array stores all variable names.
 */
125 private String[] variableNames;

/*
 * This array stores the previous values of the variables.
 */
130 private Object[] prevVariableValues;

private String[] inputEventStrings;

/*
 * This flag is true if the triggers have been modified by the simulator
 * to support the event simulation mode.
 */
135 private boolean triggerModified = false;

/*
 * The number of output ports (of the chart).
 */
140 private int noops;

/*
 * This array stores the strings used to access the input events in the
 * Matlab workspace.
 */
145 private String[] inputEventStrings;

/*
 * This flag is true if the triggers have been modified by the simulator
 * to support the event simulation mode.
 */
150 private boolean triggerModified = false;

/*
 * The size of the matrix that is the output of the ,simout, block. This
 * matrix contains the simulation results except for output variables.
 */
155
*/

```

```

    log.log(0, input);
    if (input.indexOfMatlabErrorIndicator != -1) {
        return Integer.parseInt(answer[3]);
    }
}

<**
 * Sends a command to Matlab and receives Matlab's response. This method
 * calls <code>checkMatlabResponse</code> to receive Matlab's answer to
 * the command - the command that has to be sent to Matlab.
 * @param command - the command that has to be sent to Matlab.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private String[] handleMatlabCommand(final String command)
throws IOException, SimulatorException {
    log.log(0, "HandleMatlabCommand : " + command);
    toMatlab.write(command);
    toMatlab.newLine();
    toMatlab.flush();
    checkMatlabResponse();
    return input.trim().split("\n\\s+");
}

<**
 * Obtains the position of the chart block in the Simulink diagram.
 * Returns the position (and size) of the chart block in the Simulink
 * diagram [left, top, right, bottom].
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private int[] getChartPosition() throws IOException, SimulatorException {
    String[] answer = handleMatlabCommand("get_param('"
        + fileName
        + "/'" + charName
        + "','Position')");
    int[] pos = new int[4];
    pos[0] = (int) Float.parseFloat(answer[2]); //left
    pos[1] = (int) Float.parseFloat(answer[3]); //top
    pos[2] = (int) Float.parseFloat(answer[4]); //right
    pos[3] = (int) Float.parseFloat(answer[5]); //bottom
    return pos;
}

<**
 * Obtains the number of output ports of the chart.
 * Returns The number of output ports of the chart.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private int getNumberOfOutputPorts() throws IOException,
SimulatorException {
    String[] answer = handleMatlabCommand("get_param('"
        + fileName
        + "/'" + charName
        + "','Ports')");
}

<**
 * Creates an array that contains all state objects. Each state is stored
 * at the position (i - 1) where i is its port number.
 */
private void setPortNumber(final String eventOrData)
throws IOException, SimulatorException {
    String[] answer = handleMatlabCommand(eventOrData + ".Port");
    return Integer.parseInt(answer[2]);
}

<**
 * Sets the trigger of each event to 'Either'.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private void setTriggerToEither() throws IOException,
SimulatorException {
    int port;
    for (int i = 0; i < inputSignals.length; i++) {
        if (!inputSignals[i].getTrigger().equals("Either")) {
            port = inputSignals[i].getPort();
            handleMatlabCommand(inputEventsStrings[port - 1]
                + ".Trigger='Either'");
        }
    }
}

<**
 * Restores the trigger setting of each event to its original setting.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private void restoreTrigger() throws IOException,
SimulatorException {
    int port;
    for (int i = 0; i < inputSignals.length; i++) {
        if (!inputSignals[i].getTrigger().equals("Either")) {
            port = inputSignals[i].getPort();
            handleMatlabCommand(inputEventsStrings[port - 1] +
                ".Trigger=" +
                inputSignals[i].getTrigger() + "'");
        }
    }
}

<**
 * Creates an array that contains all state objects. Each state is stored
 * at the position (i - 1) where i is its port number.
 */
private void createPortNumberArray()
throws IOException, SimulatorException {
    int port;
    for (int i = 0; i < inputEventsStrings.length; i++) {
        port = inputEventsStrings[i];
        handleMatlabCommand("set_param('" + fileName + "'." +
            port + ", PortNumber, " + i + ")");
    }
}
}

```

B. Java Code

```

        * @throws IOException if an error occurs in the input or output stream.
        */
        private void createStateActivityOutputArray() throws IOException {
            350
                SimulatorException {
                    int nos;
                    String[] answer;
                    Integer id;
                    String stateString;
                    handleMatlabCommand("StateActivityOutput=chart.find('-isa',"
                        + "'Stateflow.Data',-depth',1,'Scope','Output','"
                        + "'DataType','State')");
                    answer = handleMatlabCommand("size(StateActivityOutput)");
                    nos = Integer.parseInt(answer[2]);
                    stateActivityOutput = new StateInloops();
                    for (int i = 1; i <= nos; i++) {
                        stateString = "StateActivityOutput(\"" + i + "\")";
                        answer = handleMatlabCommand(stateString + ".OutputState.Id");
                        id = new Integer(answer[2]);
                        answer = handleMatlabCommand(stateString + ".Port");
                        stateActivityOutput.putInt(Integer.parseInt(answer[2]) - 1)
                            .(State) MatlabStateflowGrabber.map.get(id);
                    }
                }
            370
        }

        /**
         * Creates the array <code><inputSignalsArray</code>. This array stores states and
         * output signals. Each signal is stored at position (i - 1) where i is
         * the signals port number.
         * @throws IOException if an error occurs in the input or output stream.
         * @throws SimulatorException if an error is encountered
         * by Matlab.
         */
        private void createInputSignalsArray() throws IOException,
            410
                SimulatorException {
                    ArrayList outputSignalsArray = stc.getOutputEvents();
                    int port;
                    outputSignals = new IntegerSignal[outputSignalsArray.size()];
                    for (int i = 0; i < outputSignalsArray.size(); i++) {
                        port = ((IntegerSignal) outputSignalsArray.get(i)).getPort();
                        outputSignals[port - 1] =
                            (IntegerSignal) outputSignalsArray.get(i);
                    }
                }
            420
        }

        /**
         * Creates the array <code><simoutArray</code>. This array stores states and
         * output signals. The state or output signal whose port number is i is
         * stored in the array at position (i - 1).
         * @throws IOException if an error occurs in the input or output stream.
         * @throws SimulatorException if an error is encountered
         * by Matlab.
         */
        private void createSimoutArray() throws IOException,
            430
                SimulatorException {
                    int nos;
                    String[] answer;
                    Integer id;
                    String stateString;
                    simoutArray = new Object[noops];
                    ArrayList outputSignalsArray = stc.getOutputEvents();
                    int port;
                    handleMatlabCommand("StateActivityOutput=chart.find('-isa',"
                        + "'Stateflow.Data',-depth',1,'Scope','Output','"
                        + "'DataType','State')");
                    answer = handleMatlabCommand("size(StateActivityOutput)");
                    nos = Integer.parseInt(answer[2]);
                    for (int i = 1; i <= nos; i++) {
                        String eventsString;
                        inputSignals = new IntegerSignal[inputSignalsArray.size()];
                        signalValues = new Integer[inputsSignals.length];
                        for (int j = 0; j < inputSignalsArray.size(); j++) {
                            port = ((IntegerSignal) inputSignalsArray.get(j)).getPort();
                            inputSignals[port - 1] = (IntegerSignal) inputSignalsArray.get(j);
                            signalValues[port - 1] = Integer.parseInt(
                                inputSignals[port - 1].getInitialValue()
                                .toIntExpString());
                        }
                    }
                }
            450
        }

        /**
         * Creates the arrays <code><inputEvents</code>, <code><outputEvents</code>.
         * and <code><inputEventStrings</code>.
         * @throws IOException if an error occurs in the input or output stream.
         * @throws SimulatorException if an error is encountered
         * by Matlab.
         */
        private void createEvents() throws IOException,
            470
                SimulatorException {
                    int port;
                    String eventsString;
                    inputEventsStrings = new String[inputSignalsArray.size()];
                    for (int i = 0; i < inputSignalsArray.size(); i++) {
                        port = Integer.parseInt(answer[2]);
                        port = Integer.parseInt(answer[2]);
                        simoutArray[port - 1] =
                            (State) MatlabStateflowGrabber.map.get(id);
                        answer = handleMatlabCommand(stateString + ".Port");
                        answer = handleMatlabCommand(stateString + ".OutputState.Id");
                        id = new Integer(answer[2]);
                        port = Integer.parseInt(answer[2]);
                        eventString = "inputEvents(" + (i + 1) + ")";
                        eventString += chartName + "/" + fileName + " , "
                            + chartName + "/" + fileName + " , "
                            + port + "," + route + "/" + port + " )\" ";
                    }
                }
            490
        }
    
```

```

    }

    for (int i = 0; i < outputSignalsArray.size(); i++) {
        port = ((IntegerSignal) outputSignalsArray.get(i)).getPort();
        port += nos;
        simoutArray[port - 1] = (IntegerSignal) outputSignalsArray.get(1);
        handleMatlabCommand("add_line('" + fileName + "','" +
            chartName + "','" +
            port + "','" +
            route + '/' + port + "')");
    }
    //simsize = nos + outputSignalsArray.size();
    simsize = noops;
}

/** Enables the 'dataSaveToWorkspace' option of all data objects.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private void enableDataSaveToWorkspace() throws IOException,
    SimulatorException {
    String dataString;
    String[] answer = handleMatlabCommand("data=chart.find('-isa','" +
        "Stateflow.Data','-not','datatype','State','" +
        "+','-not','Scope','Input')");
    answer = handleMatlabCommand("size(data)");
    int nod = Integer.parseInt(answer[2]);
    for (int i = 1; i < nod; i++) {
        dataString = "data[" + i + "]";
        handleMatlabCommand("data=" + dataString + " SaveToWorkspace=1");
        //answer = handleMatlabCommand(dataString + ".Name");
        //variableNames[i - 1] = answer[2];
    }
}
500
/** Enables the 'HasOutputData' option of all states. States with this
 * option enabled output their status (activated or not) to Simulink.
 * @throws IOException if an error occurs in the input or output stream.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
private void enableStateOutput() throws IOException,
    SimulatorException {
    String stateString;
    String[] answer = handleMatlabCommand("states=chart.find('-isa','" +
        "Stateflow.State')");
    answer = handleMatlabCommand("size(states)");
    int nos = Integer.parseInt(answer[2]);
    for (int i = 1; i < nos; i++) {
        stateString = "states[" + i + "]";
        handleMatlabCommand(stateString + ".HasOutputData=true");
    }
}
520
/** Modifies the Simulink diagram in such a way that a simulation that
 * collects all variables that are contained in the subchart that
 */

    * is controlled by the KIEL application is possible. 'FromWorkspace',
    * and 'To-Workspace' blocks are created to provide a means to send input
    * to and receive output from the Stateflow chart.
    * @throws IOException if an error occurs in the input or output stream.
    * @throws SimulatorException if an error is encountered
    * by Matlab.
 */
private void createSimEnvForInputVariables() throws IOException,
    SimulatorException {
    String name;
    String inString;
    int Port;
    int[] position; // left top right bottom
    inputVariables = stc.getInputVariables().toArray();
    inputVarValues = new double[inputVariables.length];
    handleMatlabCommand('in=chart.find("-isa","Stateflow.Data","" +
        "-depth',1," +
        "Scope','Input")');
    String[] answer = handleMatlabCommand("size(in)");
    int noi = Integer.parseInt(answer[2]);
    position = getChartPosition();
    for (int i = 1; i < noi; i++) {
        inString = "in[" + i + "]";
        answer = handleMatlabCommand(inString + ".Port");
        port = Integer.parseInt(answer[2]);
        answer = handleMatlabCommand(inString + ".Name");
        name = answer[2];
        handleMatlabCommand("add_block('simulink/Sources/FronWSpace','" +
            + fileName + "/in" + i + ")");
        handleMatlabCommand("set_param('" + fileName +
            + "/in" + i + "','Position',["
            + position[0] + " " +
            + position[3] + " " +
            + (position[0] + 75) + " " +
            + (position[0] + 50 + i * 45) + " " +
            + (position[0] + 50 + i * 45) + " " +
            + (position[3] + 50 + i * 45) + " " +
            + fileName + "/in" + i + "']");
        handleMatlabCommand("set_param('" + fileName +
            + "/in" + i + "','VariableName','" + name +
            + inputVariableNamePrefix + ")");
        handleMatlabCommand("add_block(" +
            + "simulink/SignalAttributes/Data_Type_Conversion','" +
            + fileName + "/in" + "DTC" + i + ")");
        handleMatlabCommand("set_param('" + fileName +
            + "/DTC" + i + "','Position',["
            + (position[0] + 100) + " " +
            + (position[3] + 25 + i * 45) + " " +
            + (position[0] + 150) + " " +
            + (position[3] + 50 + i * 45) + " " +
            + fileName + "/DTC" + i + "']");
        handleMatlabCommand("add_line('" + fileName + "','" +
            + inString + "/1','" +
            + "DTC" + i + "/1')");
        handleMatlabCommand("add_line('" + fileName + "','" +
            + "DTC" + i + "/1')");
        handleMatlabCommand("add_line('" + fileName + "','" +
            + chartName + " /1','" +
            + chartName + " /" + i + "'"));
    }
}
540
560
570
590
510

```

B. Java Code

```

    * Starts at the state specified by the parameter <code>start</code>.
    * This method operates on the KIEL data structure only!
    * <code>start</code> - the state to start the collection of variables.
    * <code>variables</code> - a collection of all variables that are contained starting
      * at the state <code>start</code>.
    */
    private Collection getAllVariables(final CompositeState start) {
        590     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(getAllVariables((CompositeState) next));
                }
            }
        }
        return list;
    }

    /**
     * Creates the array <code>variables</code> that contains all variables
     * that exist in the chart.
     */
    private void createVariableArray() {
        600     Collection cvars = getAllVariables(stc.getRootNode());
        cvars.addAll(stc.getLocalVariables());
        cvars.addAll(stc.getOutputVariables());
        cvars.addAll(stc.getOutputVariables());
        //if (cvars.size() > 0) {
        //    variables = cvars.toArray();
        //}
        //else {
        //    variables = new Variable[0];
        //}
        variableNames = new String[variables.length];
        prevVariableValues = new Object[variables.length];
    }

    /**
     * Creates the arrays <code>prevVariableValues</code> and
     * <code>variableNames</code>.
     */
    private void createInitialPrevVariableValues() {
        610     for (int i = 0; i < variables.length; i++) {
            //if (variables[i] instanceof IntegerVariable) {
            //    prevVariableValues[i] = new Integer(
            //        ((IntegerVariable)variables[i]));
            //}
            else if (variables[i] instanceof DoubleVariable) {
            //    prevVariableValues[i] = new Double(
            //        ((DoubleVariable)variables[i]));
            //}
            else if (variables[i] instanceof FloatVariable) {
            //    prevVariableValues[i] = new Float(
            //        ((FloatVariable)variables[i]));
            //}
            else if (variables[i] instanceof SetValue) {
            //    prevVariableValues[i] = new SetValue(
            //        ((SetValue)variables[i]));
            //}
            else if (variables[i] instanceof StringVariable) {
            //    prevVariableValues[i] = new String(
            //        ((StringVariable)variables[i]).toString());
            }
        }
    }

    /**
     * Sets the log file to the argument <code>newLog</code>.
     * @param newLog - specifies the log file.
     */
    public final void setLogFile(final LogFile newLog) {
        620     this.log = newLog;
    }

    /**
     * Deletes all blocks in the Simulink diagram specified by the argument
     * <code>blocks</code>.
     * @param blocks - the blocks that have to be deleted.
     * @throws IOException if an error occurs in the input or output stream.
     * @throws SimulatorException if an error is encountered
     * by Matlab.
     */
    private void deleteSimulinkBlocks(final String[] blocks)
        throws IOException, SimulatorException {
        for (int i = 1; i < blocks.length; i++) {
            if (blocks[i].startsWith(fileName + "/"))
                630                660                670                680                690                700
                && !(blocks[i].equals(fileName + "/" + blocks[i] + chartName))) {
                toMatlab.write("delete _block('" + blocks[i] + "');");
                toMatlab.newLine();
                toMatlab.flush();
                checkMatlabResponse();
            }
        }
    }

    /**
     * Deletes all lines in the Simulink diagram.
     * @throws IOException if an error occurs in the input or output stream.
     * @throws SimulatorException if an error is encountered
     * by Matlab.
     */
    private void deleteSimulinkLines()
        throws IOException, SimulatorException {
        toMatlab.write("sys=/get_param('" + fileName + "','Handle')");
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Deletes all lines in the Simulink diagram.
     * @throws IOException if an error occurs in the input or output stream.
     * @throws SimulatorException if an error is encountered
     * by Matlab.
     */
    private void deleteSimulinkLines()
        throws IOException, SimulatorException {
        toMatlab.write("delete line(find_system(sys,'FindAll','"
            + "'on','type','line'))");
        toMatlab.newLine();
        toMatlab.flush();
        checkMatlabResponse();
    }

    /**
     * Resets the simulator.
     */
    public final void reset() {
}

```

```

currentStepNumber = 0;
createInitialPathVariableValues();
simBench.finishBenchmark();

try {
    handleMatlabCommand("simin=[ ]");
    for (int i = 0; i < inputVariables.length; i++) {
        handleMatlabCommand(((Variable) inputVariables[i]).getName() + " =[" + inputVariableNamePrefix + "];");
    }
    catch (Exception e) {
        log(O, e.getMessage());
    }
}

/*
 * Calculates the next simulation step.
 * @return A <code>Macrostep</code> object that contains the
 * simulation results.
 * @throws SimulatorException if an error is encountered
 * by Matlab.
 */
public final MacroStep nextStep() throws SimulatorException {
    if (currentStepNumber == 0) {
        simBench.reset();
    }
    simBench.start();
    String simMatrix = new String("[ ]");
    String[] inputVarsSimMatrices = new String[inputVariables.length];
    String[] result;
    String type;
    String dateString;
    Variable var;
    Integer id;
    int no;
    MacroStep step = new MacroStep(currentStepNumber);
    ArrayList activeStates = new ArrayList();
    //step.addMacroStep(new StateActivated(activeStates));
    //new Configuration(activeStates);
    simMatrix += currentStepNumber;
    for (int j = 0; j < inputSignals.length; j++) {
        simMatrix += " " + signalValues[j];
    }
    simMatrix += " ]";
}

try {
    for (int i = 0; i < inputVariables.length; i++) {
        handleMatlabCommand(((Variable) inputVariables[i]).getName() + " =[" + currentStepNumber + " " +
            inputVarsSimMatrices[i] + "];");
        + inputVarValues[i] + " ]");
        + "cat(1," + ((Variable) inputVariables[i]).getName() + " ");
        + inputVariableNamePrefix + " + (" + ((Variable) inputVariables[i]).getName() + " ");
        + " " + inputVarsSimMatrices[i] + " )");
    }
}

```

B. Java Code

```

        if ((simoutArray[i] instanceof State)
            && !(simoutArray[i] instanceof CompositeState)) {
            if (result[i + 3 + (currentStepNumber - 1)]
                // State was active in previous step
                step.addMicroStep(new StateDeactivated(
                    (State) simoutArray[i]));
                // Event is present if the signal value changes
                if (simoutArray[i] instanceof IntegerSignal) {
                    step.addMicroStep(new SignalAbsent(
                        (IntegerSignal) simoutArray[i]));
                }
            } else {
                // Event is absent if signal value
                // has not changed.
                if (simoutArray[i] instanceof IntegerSignal) {
                    step.addMicroStep(new SignalAbsent(
                        (IntegerSignal) simoutArray[i]));
                }
            }
        }

        /*Get current values of variables*/
        for (int i = 0; i < variableNames.length; i++) {
            result = handleMatlabCommand(variableNames[i]);
            String value = result[2];
            if (!value.equals((String) prevVariableValues[i])) {
                step.addMicroStep(new VariableValue(
                    (Variable) variableNames[i], value));
                prevVariableValues[i] = value;
            }
        }
        currentStepNumber++;
    } catch (IOException e) {
        throw new SimulatorException(e.getMessage());
    }
    simBench.stop();
    return step;
}

/**
 * Sets the statechart to the chart specified by the argument
 * <code>sc</code> and prepares the simulator for the first simulation
 * step.
 * Param sc - the new statechart that has to be simulated.
 * Throws SimulatorException if an error is encountered
 * by Matlab.
 */
public final void setStateChart(final StateChart sc)
throws SimulatorException {
    int[] position; // left top right bottom J
    stc = sc;
    try {
        fileName = (handleMatlabCommand("chart.machine.Name") [2]);
        simBench.printOutputToFile(new File ("SimBench" + fileName + ".txt"));
        handleMatlabCommand("chart.Name");
        chartName = input.trim().split("\n+")[1];
        /*check whether simulink environment is empty or not*/
        toMatlab.write("find_system('SearchDepth',1)");
        toMatlab.newline();
        toMatlab.flush();
        checkMatlabResponse();
        String[] answer = input.trim().replaceAll("\n+", "\u00a0")
            .split("\u00a0*\u00a0*\u00a0");
        if (answer.length <= 4) {
            deleteSimulinkLines();
            deleteSimulinkBlocks(answer);
        }
        enableStatOutput();
        position = getChartPosition();
        /*Set solver type to fixed-step discrete*/
        handleMatlabCommand("acs=setActiveConfigSet(model)");
        handleMatlabCommand("acs.set_param('SolverType','Fixed-step')");
        handleMatlabCommand("acs.set_param('Solver','FixedStepDiscrete')");
        ;
        handleMatlabCommand("acs.set_param('FixedStep','1.0')");

        /*Set Simulink blocks for input and output.*/
        handleMatlabCommand("simulink");
        handleMatlabCommand("add_block('simulink/Sinks/ToWorkspace', '"
            + fileName + "/output')");
        handleMatlabCommand("set_param('" + fileName +
            "/output', 'SaveFormat', 'Array')");
        handleMatlabCommand("set_param('" + fileName +
            "/output', 'Position', ["
            + position[2] + " 140] + "\u00a0"
            + position[1] + "\u00a0"
            + position[2] + " 190] + "\u00a0"
            + position[1] + " 25] + "\u00a0");
        noops = genNumberOfOutputPorts();
        handleMatlabCommand("add_block('simulink/Routing/Mux', '"
            + fileName + "/route')");
        handleMatlabCommand("set_param('"
            + "/route', 'Position', ["
            + position[2] + " 100 + "\u00a0"
            + position[1] + "\u00a0"
            + position[2] + " 105 + "\u00a0"
            + position[1] + " 55 + "\u00a0")
            + "(position[2] + 105 + "\u00a0"
            + position[1] + " 55 + "\u00a0))");
        handleMatlabCommand("set_param('"
            + "/input', 'Position', ["
            + (position[2] + 200) + "\u00a0"
            + position[1] + "\u00a0"
            + noops + "\u00a0"]);
        createInputSignalsArray();
        if (inputSignals.length > 0) {
            handleMatlabCommand("add_block("
                + "'simulink/Sources/FromWorkspace', '"
                + fileName + "/input')");
            handleMatlabCommand("set_param('"
                + "/input', 'Position', ["
                + (position[2] + 200) + "\u00a0"
                + position[1] + "\u00a0"
                + "\u00a0"));
        }
    }
}

```

```

940           + (position[2] + 250) + "u"
941           + (position[1] + 25) + "u";
942           handleMatlabCommand("add_line('" + fileName
943           + "', input/1',_" + chartName
944           + " chartName
945           + "/Trigger')");
946       }
947   }
948
949   /**
950    * Emits the valued event specified by the argument <code>e</code>
951    * with the value specified by <code>i</code>.
952    * @param e - the event.
953    * @param i - the value.
954    * @throws SimulatorException if an error is encountered
955    * by Matlab.
956    */
957   public final void emit(final Event e, final int i)
958   throws SimulatorException {
959       if (triggerModified) {
960           try {
961               restoreTrigger();
962           } catch (IOException ex) {
963               throw new SimulatorException(e.toString());
964           }
965       }
966   }
967
968   /**
969    * Returns all input events.
970    * @return A collection of all input events.
971    */
972   public final Collection<Event> getInput() {
973       return stc.getInputEvents();
974   }
975
976   /**
977    * Returns all output events.
978    * @return A collection of all output events.
979    */
980   public final Collection<Event> getOutput() {
981       return stc.getOutputEvents();
982   }
983
984   /**
985    * Emits the event specified by the argument <code>e</code>.
986    * @param e - the event.
987    * @throws SimulatorException if an error is encountered
988    * by Matlab.
989    */
990   public final void emit(final Event e) throws SimulatorException {
991       for (int i = 0; i < inputSignals.length; i++) {
992           int port;
993           if (!triggerModified) {
994               try {
995                   setTriggerToEither();
996               } catch (IOException ex) {
997                   throw new SimulatorException(e.toString());
998               }
999           }
1000          if ((inputSignals[i].getPort() == e.getPort()) &&
1001              (inputSignals[i].getPort() == e.getPort()))
1002          {
1003              if (signalValues[port - 1] > 0) {
1004                  signalValues[port - 1] = -1;
1005              }
1006          }
1007      }
1008  }
1009
1010 /**
1011  * Sets the variable specified by <code>var</code> to the value
1012  * specified by <code>value</code>.
1013  * @param var - the variable.
1014  * @param value - the new value of the variable.
1015  * @throws SimulatorException if an error is encountered
1016  * by Matlab.
1017  */
1018 public final void setVariable(final Variable var, final String value)
1019 throws SimulatorException {
1020     for (int i = 0; i < inputVariables.length; i++) {
1021         if (var.equals(inputVariables[i])) {
1022             if (value.equals("true")) {
1023                 inputVarsValues[i] = 1;
1024             } else if (value.equals("false")) {
1025                 inputVarsValues[i] = 0;
1026             } else {
1027                 inputVarsValues[i] = Double.parseDouble(value);
1028             }
1029         }
1030     }
1031 }
1032
1033 /**
1034  * Sets the feature specified by <code>feature</code> to either
1035  * <code>true</code> or <code>false</code>. Actually, this function
1036  * does absolutely nothing.
1037  * @param feature - the feature that has to be enabled or disabled
1038  * @param status - specified whether to enable or disable the feature
1039  * @throws SimulatorException if an error is encountered
1040  * by Matlab.
1041  */
1042 public void setFeature(final String feature, final boolean status)
1043 throws SimulatorException {

```

B. Java Code

```

1060      }
1061      /**
1062       * Tells whether the feature <code>feature</code> is supported by this
1063       * simulator or not. Currently, this function returns true for the
1064       * <StringLabel> Simulation, only.
1065       * Param feature - the feature for which one wants to know whether it is
1066       * supported or not.
1067       * @return true if the requested feature is supported, otherwise false.
1068       * @throws SimulatorException if an error is encountered
1069       * by Matlab.
1070     */
1071     public final boolean getFeature(final String feature)

```

B.6.2. SimulatorException

```

/*
 * Created on 20.10.2004
 */
package kiel.simulator;

/**
 * SimulatorException is a basic class for all exception which occur during the
 * simulation.
 * @author Andre Olhaff
 */
public class SimulatorException extends Exception {
    /**
     * Constructs a new SimulatorException with the specific detail message.
     * @param message
     *      the detail message
     */
    public SimulatorException(final String message) {
        super(message);
    }
}

```

10

20

20

B.7. kiel.editor.controller

B.7.1. EditorModeAddJunction

```

package kiel.editor.controller;

import kiel.dataStructure.Junction;

/*
 * The state in which the user can add a Junction.
 */
10  * <p>Copyright: (c) 2005 Adrian Posor</p>
   * <p>Company: Uni Kiel</p>
   * @author apo
   * @version $Revision: 1.1 $ Last modified $Date: 2005/10/20 12:26:22 $
   */
public class EditorModeAddJunction extends EditorModeAddNode {
    /**
     * Creates an object with a Junction.gif image.
     */
    EditorModeAddJunction() {
        super("junction.gif", Junction.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}

```

B. Java Code

B.7.2: ActionEditStateChartEventsForStateflow

B.7. kiel.editor.controller

B. Java Code

B.7.3. ActionEditStateChartVariablesForStateflow

B. Java Code

```

220 deleteButton.setAction(new AbstractAction(ResourceBundle
    .getString("delete")) {
280     public void actionPerformed(ActionEvent e) {
        int[] selectedRows = myTable.getSelectedRows();
        Arrays.sort(selectedRows);
        int offset = 0;
        for (int i = 0; i < selectedRows.length; i++) {
            myTableModel.removeRow(selectedRows[i] - offset);
            offset++;
        }
    }
}
JPanel myContentPane = new JPanel();
myContentPane.setOpaque(true);
myContentpane.add(myScrollPane);
myContentpane.add(addButton);
myContentpane.add(deleteButton);
myContentpane.add(okButton);
myContentpane.add(cancelButton);
myDialog.setContentPane(myContentPane);

protected void doAction(ActionEvent e) {
    getVariables();
    updateText();
    myDialog.pack();
    myDialog.setVisible(true);
}

private void getVariables() {
    Vector row;
    Vector dataVector = myTableModel.getDataVector();
    dataVector.clear();
    StateChart statechart = getEditor().getGraph().getMyGraphModel()
        .getCurrentStatechart();
    Collection variables = statechart.getInputVariables();
    Iterator it = variables.iterator();
    for (int i = 0; it.hasNext(); i++) {
        row = new Vector();
        Variable var = (Variable)it.next();
        row.add(var.getName());
        row.add("Input");
        row.add(getVariableType(var));
        row.add(var.toExpString());
        dataVector.add(row);
    }
}
230
290
300
310
320
330
variables = statechart.getLocalVariables();
it = variables.iterator();
for (int i = 0; it.hasNext(); i++) {
    row = new Vector();
    Variable var = (Variable)it.next();
    row.add(var.getName());
    row.add("Local");
    row.add(getVariableType(var));
    row.add(var.toExpString());
    dataVector.add(row);
}
}
variables = statechart.getConstants();
it = variables.iterator();
for (int i = 0; it.hasNext(); i++) {
    row = new Vector();
    Constant con = (Constant)it.next();
    row.add(con.getName());
    row.add("Constant");
    row.add(getConstantType(con));
    row.add(con.toExpString());
    dataVector.add(row);
}
}
}
private void updateText() {
    okButton.setText(ResourceBundle.getString("OK"));
    cancellation.setText(ResourceBundle.getString("Cancel"));
    addButton.setText(ResourceBundle.getString("add"));
    deleteButton.setText(ResourceBundle.getString("delete"));
    myDialog.setTitle(ResourceBundle.getString("editStatechartVariables"));
}
}
private String getVariableType(Variable var) {
    if (var.getClass().equals(IntegerVariable.class)) {
        return "int32";
    } else if (var.getClass().equals(Integer16Variable.class)) {
        return "int16";
    } else if (var.getClass().equals(Integer8Variable.class)) {
        return "int8";
    } else if (var.getClass().equals(UInt32Variable.class)) {
        return "uint32";
    } else if (var.getClass().equals(UInt16Variable.class)) {
        return "uint16";
    } else if (var.getClass().equals(UInt8Variable.class)) {
        return "uint8";
    } else if (var.getClass().equals(FloatVariable.class)) {
        return "single";
    } else if (var.getClass().equals(DoubleVariable.class)) {
        return "double";
    } else if (var.getClass().equals(BooleanVariable.class)) {
        return "boolean";
    } else return "not_supported_type";
}
}
private String getConstantType(Constant con) {
    if (con.getClass().equals(IntegerConstant.class)) {
        240
        300
        360
        420
        480
        540
        600
        660
        720
        780
        840
        900
        960
        1020
        1080
        1140
        1200
        1260
        1320
        1380
        1440
        1500
        1560
        1620
        1680
        1740
        1800
        1860
        1920
        1980
        2040
        2100
        2160
        2220
        2280
        2340
        2400
        2460
        2520
        2580
        2640
        2700
        2760
        2820
        2880
        2940
        3000
        3060
        3120
        3180
        3240
        3300
        3360
        3420
        3480
        3540
        3600
        3660
        3720
        3780
        3840
        3900
        3960
        4020
        4080
        4140
        4200
        4260
        4320
        4380
        4440
        4500
        4560
        4620
        4680
        4740
        4800
        4860
        4920
        4980
        5040
        5100
        5160
        5220
        5280
        5340
        5400
        5460
        5520
        5580
        5640
        5700
        5760
        5820
        5880
        5940
        6000
        6060
        6120
        6180
        6240
        6300
        6360
        6420
        6480
        6540
        6600
        6660
        6720
        6780
        6840
        6900
        6960
        7020
        7080
        7140
        7200
        7260
        7320
        7380
        7440
        7500
        7560
        7620
        7680
        7740
        7800
        7860
        7920
        7980
        8040
        8100
        8160
        8220
        8280
        8340
        8400
        8460
        8520
        8580
        8640
        8700
        8760
        8820
        8880
        8940
        9000
        9060
        9120
        9180
        9240
        9300
        9360
        9420
        9480
        9540
        9600
        9660
        9720
        9780
        9840
        9900
        9960
        10020
        10080
        10140
        10200
        10260
        10320
        10380
        10440
        10500
        10560
        10620
        10680
        10740
        10800
        10860
        10920
        10980
        11040
        11100
        11160
        11220
        11280
        11340
        11400
        11460
        11520
        11580
        11640
        11700
        11760
        11820
        11880
        11940
        12000
        12060
        12120
        12180
        12240
        12300
        12360
        12420
        12480
        12540
        12600
        12660
        12720
        12780
        12840
        12900
        12960
        13020
        13080
        13140
        13200
        13260
        13320
        13380
        13440
        13500
        13560
        13620
        13680
        13740
        13800
        13860
        13920
        13980
        14040
        14100
        14160
        14220
        14280
        14340
        14400
        14460
        14520
        14580
        14640
        14700
        14760
        14820
        14880
        14940
        15000
        15060
        15120
        15180
        15240
        15300
        15360
        15420
        15480
        15540
        15600
        15660
        15720
        15780
        15840
        15900
        15960
        16020
        16080
        16140
        16200
        16260
        16320
        16380
        16440
        16500
        16560
        16620
        16680
        16740
        16800
        16860
        16920
        16980
        17040
        17100
        17160
        17220
        17280
        17340
        17400
        17460
        17520
        17580
        17640
        17700
        17760
        17820
        17880
        17940
        18000
        18060
        18120
        18180
        18240
        18300
        18360
        18420
        18480
        18540
        18600
        18660
        18720
        18780
        18840
        18900
        18960
        19020
        19080
        19140
        19200
        19260
        19320
        19380
        19440
        19500
        19560
        19620
        19680
        19740
        19800
        19860
        19920
        19980
        20040
        20100
        20160
        20220
        20280
        20340
        20400
        20460
        20520
        20580
        20640
        20700
        20760
        20820
        20880
        20940
        21000
        21060
        21120
        21180
        21240
        21300
        21360
        21420
        21480
        21540
        21600
        21660
        21720
        21780
        21840
        21900
        21960
        22020
        22080
        22140
        22200
        22260
        22320
        22380
        22440
        22500
        22560
        22620
        22680
        22740
        22800
        22860
        22920
        22980
        23040
        23100
        23160
        23220
        23280
        23340
        23400
        23460
        23520
        23580
        23640
        23700
        23760
        23820
        23880
        23940
        24000
        24060
        24120
        24180
        24240
        24300
        24360
        24420
        24480
        24540
        24600
        24660
        24720
        24780
        24840
        24900
        24960
        25020
        25080
        25140
        25200
        25260
        25320
        25380
        25440
        25500
        25560
        25620
        25680
        25740
        25800
        25860
        25920
        25980
        26040
        26100
        26160
        26220
        26280
        26340
        26400
        26460
        26520
        26580
        26640
        26700
        26760
        26820
        26880
        26940
        27000
        27060
        27120
        27180
        27240
        27300
        27360
        27420
        27480
        27540
        27600
        27660
        27720
        27780
        27840
        27900
        27960
        28020
        28080
        28140
        28200
        28260
        28320
        28380
        28440
        28500
        28560
        28620
        28680
        28740
        28800
        28860
        28920
        28980
        29040
        29100
        29160
        29220
        29280
        29340
        29400
        29460
        29520
        29580
        29640
        29700
        29760
        29820
        29880
        29940
        30000
        30060
        30120
        30180
        30240
        30300
        30360
        30420
        30480
        30540
        30600
        30660
        30720
        30780
        30840
        30900
        30960
        31020
        31080
        31140
        31200
        31260
        31320
        31380
        31440
        31500
        31560
        31620
        31680
        31740
        31800
        31860
        31920
        31980
        32040
        32100
        32160
        32220
        32280
        32340
        32400
        32460
        32520
        32580
        32640
        32700
        32760
        32820
        32880
        32940
        33000
        33060
        33120
        33180
        33240
        33300
        33360
        33420
        33480
        33540
        33600
        33660
        33720
        33780
        33840
        33900
        33960
        34020
        34080
        34140
        34200
        34260
        34320
        34380
        34440
        34500
        34560
        34620
        34680
        34740
        34800
        34860
        34920
        34980
        35040
        35100
        35160
        35220
        35280
        35340
        35400
        35460
        35520
        35580
        35640
        35700
        35760
        35820
        35880
        35940
        36000
        36060
        36120
        36180
        36240
        36300
        36360
        36420
        36480
        36540
        36600
        36660
        36720
        36780
        36840
        36900
        36960
        37020
        37080
        37140
        37200
        37260
        37320
        37380
        37440
        37500
        37560
        37620
        37680
        37740
        37800
        37860
        37920
        37980
        38040
        38100
        38160
        38220
        38280
        38340
        38400
        38460
        38520
        38580
        38640
        38700
        38760
        38820
        38880
        38940
        39000
        39060
        39120
        39180
        39240
        39300
        39360
        39420
        39480
        39540
        39600
        39660
        39720
        39780
        39840
        39900
        39960
        40020
        40080
        40140
        40200
        40260
        40320
        40380
        40440
        40500
        40560
        40620
        40680
        40740
        40800
        40860
        40920
        40980
        41040
        41100
        41160
        41220
        41280
        41340
        41400
        41460
        41520
        41580
        41640
        41700
        41760
        41820
        41880
        41940
        42000
        42060
        42120
        42180
        42240
        42300
        42360
        42420
        42480
        42540
        42600
        42660
        42720
        42780
        42840
        42900
        42960
        43020
        43080
        43140
        43200
        43260
        43320
        43380
        43440
        43500
        43560
        43620
        43680
        43740
        43800
        43860
        43920
        43980
        44040
        44100
        44160
        44220
        44280
        44340
        44400
        44460
        44520
        44580
        44640
        44700
        44760
        44820
        44880
        44940
        45000
        45060
        45120
        45180
        45240
        45300
        45360
        45420
        45480
        45540
        45600
        45660
        45720
        45780
        45840
        45900
        45960
        46020
        46080
        46140
        46200
        46260
        46320
        46380
        46440
        46500
        46560
        46620
        46680
        46740
        46800
        46860
        46920
        46980
        47040
        47100
        47160
        47220
        47280
        47340
        47400
        47460
        47520
        47580
        47640
        47700
        47760
        47820
        47880
        47940
        48000
        48060
        48120
        48180
        48240
        48300
        48360
        48420
        48480
        48540
        48600
        48660
        48720
        48780
        48840
        48900
        48960
        49020
        49080
        49140
        49200
        49260
        49320
        49380
        49440
        49500
        49560
        49620
        49680
        49740
        49800
        49860
        49920
        49980
        50040
        50100
        50160
        50220
        50280
        50340
        50400
        50460
        50520
        50580
        50640
        50700
        50760
        50820
        50880
        50940
        51000
        51060
        51120
        51180
        51240
        51300
        51360
        51420
        51480
        51540
        51600
        51660
        51720
        51780
        51840
        51900
        51960
        52020
        52080
        52140
        52200
        52260
        52320
        52380
        52440
        52500
        52560
        52620
        52680
        52740
        52800
        52860
        52920
        52980
        53040
        53100
        53160
        53220
        53280
        53340
        53400
        53460
        53520
        53580
        53640
        53700
        53760
        53820
        53880
        53940
        54000
        54060
        54120
        54180
        54240
        54300
        54360
        54420
        54480
        54540
        54600
        54660
        54720
        54780
        54840
        54900
        54960
        55020
        55080
        55140
        55200
        55260
        55320
        55380
        55440
        55500
        55560
        55620
        55680
        55740
        55800
        55860
        55920
        55980
        56040
        56100
        56160
        56220
        56280
        56340
        56400
        56460
        56520
        56580
        56640
        56700
        56760
        56820
        56880
        56940
        57000
        57060
        57120
        57180
        57240
        57300
        57360
        57420
        57480
        57540
        57600
        57660
        57720
        57780
        57840
        57900
        57960
        58020
        58080
        58140
        58200
        58260
        58320
        58380
        58440
        58500
        58560
        58620
        58680
        58740
        58800
        58860
        58920
        58980
        59040
        59100
        59160
        59220
        59280
        59340
        59400
        59460
        59520
        59580
        59640
        59700
        59760
        59820
        59880
        59940
        60000
        60060
        60120
        60180
        60240
        60300
        60360
        60420
        60480
        60540
        60600
        60660
        60720
        60780
        60840
        60900
        60960
        61020
        61080
        61140
        61200
        61260
        61320
        61380
        61440
        61500
        61560
        61620
        61680
        61740
        61800
        61860
        61920
        61980
        62040
        62100
        62160
        62220
        62280
        62340
        62400
        62460
        62520
        62580
        62640
        62700
        62760
        62820
        62880
        62940
        63000
        63060
        63120
        63180
        63240
        63300
        63360
        63420
        63480
        63540
        63600
        63660
        63720
        63780
        63840
        63900
        63960
        64020
        64080
        64140
        64200
        64260
        64320
        64380
        64440
        64500
        64560
        64620
        64680
        64740
        64800
        64860
        64920
        64980
        65040
        65100
        65160
        65220
        65280
        65340
        65400
        65460
        65520
        65580
        65640
        65700
        65760
        65820
        65880
        65940
        66000
        66060
        66120
        66180
        66240
        66300
        66360
        66420
        66480
        66540
        66600
        66660
        66720
        66780
        66840
        66900
        66960
        67020
        67080
        67140
        67200
        67260
        67320
        67380
        67440
        67500
        67560
        67620
        67680
        67740
        67800
        67860
        67920
        67980
        68040
        68100
        68160
        68220
        68280
        68340
        68400
        68460
        68520
        68580
        68640
        68700
        68760
        68820
        68880
        68940
        69000
        69060
        69120
        69180
        69240
        69300
        69360
        69420
        69480
        69540
        69600
        69660
        69720
        69780
        69840
        69900
        69960
        70020
        70080
        70140
        70200
        70260
        70320
        70380
        70440
        70500
        70560
        70620
        70680
        70740
        70800
        70860
        70920
        70980
        71040
        71100
        71160
        71220
        71280
        71340
        71400
        71460
        71520
        71580
        71640
        71700
        71760
        71820
        71880
        71940
        72000
        72060
        72120
        72180
        72240
        72300
        72360
        72420
        72480
        72540
        72600
        72660
        72720
        72780
        72840
        72900
        72960
        73020
        73080
        73140
        73200
        73260
        73320
        73380
        73440
        73500
        73560
        73620
        73680
        73740
        73800
        73860
        73920
        73980
        74040
        74100
        74160
        74220
        74280
        74340
        74400
        74460
        74520
        74580
        74640
        74700
        74760
        74820
        74880
        74940
        75000
        75060
        75120
        75180
        75240
        75300
        75360
        75420
        75480
        75540
        75600
        75660
        75720
        75780
        75840
        75900
        75960
        76020
        76080
        76140
        76200
        76260
        76320
        76380
        76440
        76500
        76560
        76620
        76680
        76740
        76800
        76860
        76920
        76980
        77040
        77100
        77160
        77220
        77280
        77340
        77400
        77460
        77520
        77580
        77640
        77700
        77760
        77820
        77880
        77940
        78000
        78060
        78120
        78180
        78240
        78300
        78360
        78420
        78480
        78540
        78600
        78660
        78720
        78780
        78840
        78900
        78960
        790
```

B. Java Code

B.7.4. ActionEditStateEventsForStateflow

```

package kiel.editor.controller;

import java.awt.event.ActionEvent;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.JScrollPane;
import javax.swing.UIManager;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.MyGraphConstants;
import java.util.Vector;
import java.util.ArrayList;
import java.util.Arrays;
import kiel.datastructure.eventexp.Event;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;
/*
 * @author ap
 * @version $Revision: 1.1 $ last modified $Date: 2005/10/28 12:03:39 $
 */
public class ActionEditStateEventsForStateflow extends EditorAction {
    /*
     * @param iconName
     */
    public JButton okButton;
    private JDialog myDialog;
    private JPanel myTable;
    private DefaultTableModel myTableModel;
    private JButton addButton;
    private JButton deleteButton;
    private JButton cancelButton;
    private JButton addRow;
    private JButton deleteRow;
    Vector columnNames = new Vector();
    columnNames.add("Name");
    Vector rowNames = new Vector();
    rowNames.add("Edit16.gif");
    Vector columnNames1 = new Vector();
    columnNames1.add("Name");
    Vector rowNames1 = new Vector();
    rowNames1.add("Edit16.gif");
    public ActionEditStateEventsForStateflow() {
        super("Edit16.gif");
        myDialog = new JDialog(getEditor().getFrame());
        myDialog.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        myDialog.setLayout(new GridLayout(2, 2));
        myDialog.add(new JLabel("Edit State Events"), "North");
        myDialog.add(new JButton("OK"), "South");
        myDialog.add(new JButton("Cancel"), "South");
        myDialog.add(new JButton("Delete Row"), "East");
        myDialog.add(new JButton("Add Row"), "East");
        myDialog.pack();
        myDialog.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == okButton) {
            myDialog.setVisible(false);
            myDialog.dispose();
        } else if (e.getSource() == cancelButton) {
            myDialog.setVisible(false);
            myDialog.dispose();
        } else if (e.getSource() == deleteRow) {
            int selectedRow = myTable.getSelectedRow();
            if (selectedRow > -1) {
                myTable.removeRow(selectedRow);
                myTable.repaint();
            }
        } else if (e.getSource() == addButton) {
            int selectedRow = myTable.getSelectedRow();
            if (selectedRow > -1) {
                myTable.insertRow(selectedRow, rowNames);
                myTable.repaint();
            }
        }
    }
}

class AbstractAction extends ActionEditStateEventsForStateflow {
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == okButton) {
            myDialog.setVisible(false);
            myDialog.dispose();
        } else if (e.getSource() == cancelButton) {
            myDialog.setVisible(false);
            myDialog.dispose();
        } else if (e.getSource() == deleteRow) {
            int selectedRow = myTable.getSelectedRow();
            if (selectedRow > -1) {
                myTable.removeRow(selectedRow);
                myTable.repaint();
            }
        } else if (e.getSource() == addButton) {
            int selectedRow = myTable.getSelectedRow();
            if (selectedRow > -1) {
                myTable.insertRow(selectedRow, rowNames);
                myTable.repaint();
            }
        }
    }
}

```

B. Java Code

```
myContentPane.add(cancelButton);
myDialog.setContentPane(myContentPane);

protected void doAction(ActionEvent e) {
    getEvents();
    updateText();
    myDialog.pack();
    myDialog.setVisible(true);
}

private void getEvents() {
    Vector row;
    Vector dataVector = myTableModel.getDataVector();
    dataVector.clear();
    State state = (State) MyGraphConstants
        .getGraphicalObject((DefaultGraphCell) getEditor()
            .getGraph().getSelectionCell()).getAttributes();
    ArrayList events = state.getLocalevents();
    120
    for (int i = 0; i < events.size(); i++) {
        row = new Vector();
        Event event = (Event) events.get(i);
        row.add(event.getName());
        dataVector.add(row);
    }
    myTableModel.fireTableDataChanged();
}

private void updateText() {
    okButton.setText(ResourceBundle.getString("OK"));
    cancelButton.setText(ResourceBundle.getString("Cancel"));
    addButton.setText(ResourceBundle.getString("add"));
    deleteButton.setText(ResourceBundle.getString("delete"));
    myDialog.setTitle(ResourceBundle.getString("editStateEvents"));
}
140
```

B.7.5. ActionEditStateVariablesForStateflow

B. Java Code

```

        .getGraphicalObject((DefaultGraphNode) getEditor()
        .getGraph().getSelectionCell()).getAttributes();
        state.getVariables().clear();
        state.getConstants().clear();

        String name;
        String type;
        String initialValue;
        Variable var = null;
        Constant con = null;

        try {
            for (int i = 0; i < myTableModel.getRowCount(); i++) {
                name = (String) myTableModel.getValueAt(i, 0);
                type = (String) myTableModel.getValueAt(i, 2);
                initialValue = (String) myTableModel.getValueAt(i, 3);

                if (type.equals("double")) {
                    180
                    con = new DoubleConstant(name,
                        Double.parseDouble(initialValue));
                    var = new DoubleVariable(name,
                        (DoubleConstant) con);
                } else if (type.equals("single")) {
                    con = new FloatConstant(name,
                        Float.parseFloat(initialValue));
                    var = new FloatVariable(name, (FloatConstant) con);
                } else if (type.equals("int32")) {
                    190
                    con = new IntegerConstant(name,
                        Integer.parseInt(initialValue));
                    var = new IntegerVariable(name,
                        (IntegerConstant) con);
                } else if (type.equals("int16")) {
                    con = new Integer16Constant(name,
                        Short.parseShort(initialValue));
                    var = new IntegerVariable(name,
                        (Integer16Constant) con);
                } else if (type.equals("uint8")) {
                    200
                    con = new UInt8Constant(name,
                        Byte.parsebyte(initialValue));
                    var = new UInt8Variable(name,
                        (UInt8Constant) con);
                } else if (type.equals("uint32")) {
                    con = new UInt32Constant(name,
                        Long.parseLong(initialValue));
                    var = new UInt32Variable(name,
                        (UInt32Constant) con);
                } else if (type.equals("uint16")) {
                    210
                    con = new UInt16Constant(name,
                        Integer.parseInt(initialValue));
                    var = new UInt16Variable(name,
                        (UInt16Constant) con);
                } else if (type.equals("uint8")) {
                    con = new UInt8Constant(name,
                        Short.parseShort(initialValue));
                    var = new UInt8Variable(name, (UInt8Constant) con);
                } else if (type.equals("boolean")) {
                    220
                    if (initialValue.equals("true")) {
                        con = new BooleanTrue(name);
                        var = new BooleanVariable(name,
                            (BooleanExpression) con);
                    }
                }
            }
        }
    }

    120
    130
    140
    150
    160
    170
}

```

```

myContentpane.add(deletebutton);
myContentpane.add(okbutton);
myContentpane.add(cancelbutton);
mydialog.setContentPane(myContentpane);
}

protected void doAction(ActionEvent e) {
    getVariables();
    updateText();
    myDialog.pack();
    myDialog.setVisible(true);
}

private void getVariables() {
    Vector row = new Vector();
    Vector dataVector = myTableModel.getDataVector();
    dataVector.clear();
    State state = (State) MyrapConstants
        .getGraphicalObject(((DefaultGraphCell) getEditor())
            .getGraph().getSelectionCell()).getAttributes();

    Collection variables = state.getVariables();
    Iterator it = variables.iterator();
    for (int i = 0; it.hasNext(); i++) {
        row = new Vector();
        Variable var = (Variable) it.next();
        row.add(var.getName());
        row.add("Local");
        row.add(getVariableType(var));
        dataVector.add(row);
    }
}

variables = state.getConstants();
it = variables.iterator();
for (int i = 0; it.hasNext(); i++) {
    row = new Vector();
    Constant con = (Constant) it.next();
    row.add(con.getName());
    row.add("Constant");
    row.add(getConstantType(con));
    row.add(con.toExtString());
    dataVector.add(row);
}

myTableModel.fireTableDataChanged();
}

private void updateText() {
    okButton.setText(ResourceBundle.getString("OK"));
    cancelButton.setText(ResourceBundle.getString("Cancel"));
}

private String getVariableType(Variable var) {
    if (var.getClass().equals(IntegerVariable.class)) {
        return "int32";
    } else if (var.getClass().equals(Integer16Variable.class)) {
        return "int16";
    } else if (var.getClass().equals(Integer8Variable.class)) {
        return "int8";
    } else if (var.getClass().equals(UInt32Variable.class)) {
        return "uint32";
    } else if (var.getClass().equals(UInt16Variable.class)) {
        return "uint16";
    } else if (var.getClass().equals(UInt8Variable.class)) {
        return "uint8";
    } else if (var.getClass().equals(FlatVariable.class)) {
        return "single";
    } else if (var.getClass().equals(DoubleVariable.class)) {
        return "double";
    } else if (var.getClass().equals(BooleanVariable.class)) {
        return "boolean";
    } else return "not_supported";
}

private String getConstantType(Constant con) {
    if (con.getClass().equals(IntegerConstant.class)) {
        return "int32";
    } else if (con.getClass().equals(Integer16Constant.class)) {
        return "int16";
    } else if (con.getClass().equals(Integer8Constant.class)) {
        return "int8";
    } else if (con.getClass().equals(UInt32Constant.class)) {
        return "uint32";
    } else if (con.getClass().equals(UInt16Constant.class)) {
        return "uint16";
    } else if (con.getClass().equals(UInt8Constant.class)) {
        return "uint8";
    } else if (con.getClass().equals(FloatConstant.class)) {
        return "single";
    } else if (con.getClass().equals(DoubleConstant.class)) {
        return "double";
    } else if (con.getClass().equals(BooleanTrue.class)) {
        || con.getClass().equals(BooleanFalse.class))
        return "boolean";
    } else return "not_supported";
}
}

```

B.7.6. ActionEditStateEntryActionsForStateflow

```

package kiel.editor.controller;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import org.jerph.graph.DefaultGraphCell;
import kiel.dataStructure.State;
import kiel.dataStructure.action.ActionsStringLabel;
import kiel.editor.GraphMyraphConstants;
import kiel.editor.Resources.ResourceBundle;
import kiel.editor.Resources.ResourceBundle;
30   * java.awt.event.ActionEvent)
31   */
32   protected void doAction(final ActionEvent e) {
33     State state = (State) MyraphConstants
34       .getGraphicalObject(((DefaultGraphCell) getEditor()).getGraph())
35       .getSelectionCell().getAttributes();
36     String actionsText = state.getEntry().toString();
37     final int columns = 50;
38     JTextField textField = new JTextField(actionsText, columns);
39     int answer = JOptionPane.showOptionDialog(
40       getEditor().getContentPane(), showOptionDialog(
41         textField, "editOnEntryActions",
42         ResourceBundle.getString("editOnEntryActions"),
43         JOptionPane.OK_CANCEL_OPTION,
44         JOptionPane.QUESTION_MESSAGE,
45         null, new Object[] {
46           ResourceBundle.getString("OK"),
47           ResourceBundle.getString("Cancel")}),
48         ResourceBundle.getString("OK"));
49     if (answer == 0) {
50       state.setEntry(new ActionsStringLabel(textfield.getText()));
51       getEditor().getGraph().repaint();
52     }
53   }
54 }

55 /**
56  * @author apo
57  * @version $Revision: 1.1 $ last modified $Date: 2005/10/28 12:03:39 $
58  */
59 public class ActionEditStateEntriesForStateflow extends EditorAction {
60   /**
61    * Creates an object with an Edit16.gif image.
62    */
63   ActionEditStateEntriesForStateflow() {
64     super("Edit16.gif");
65   }
66 }

67 /**
68  * @see kiel.editor.controller.EditorAction#doAction(
69  */

```

B.7.7. ActionEditStateDoActionsForStateflow

```

30      * java.awt.event.ActionEvent)
31      */
32      protected void doAction(final ActionEvent e) {
33
34          State state = (State) MyGraphConstants
35              .getGraphicalObject((DefaultGraphCell) getEditor().getGraph())
36
37          String actionText = state.getDoActivity().toString();
38
39          final int columns = 50;
40          JTextField textField = new JTextField(actionsText, columns);
41          JOptionPane.showOptionDialog(
42              getEditor(), getKieLFrame().getJFrame(),
43              ResourceBundle.getString("EditStateDoActivities"),
44              textField,
45              ResourceBundle.getString("EditStateDoActivities"),
46              JOptionPane.OK_CANCEL_OPTION,
47              ResourceBundle.getBundle("Ok"),
48              ResourceBundle.getBundle("Cancel"));
49
50          if (answer == 0) {
51              state.setDoActivity(new ActionsStringLabel(textfield.getText()));
52              getEditor().getGraph().repaint();
53          }
54      }
55
56      /**
57      * @author apo
58      * @version $Revision: 1.1 $ last modified $date: 2005/10/28 12:03:39 $
59      */
60      public class ActionEditStateActionsForStateflow extends EditorAction {
61
62          /**
63          * Creates an object with an Edit16.gif image.
64          */
65          ActionEditStateActionsForStateflow() {
66              super("Edit16.gif");
67          }
68
69      }
70
71      /**
72      * @param kiel editor.controller;
73      * @param java.awt.event.ActionEvent;
74      */
75      public void doAction(ActionEvent e) {
76
77          State state = (State) MyGraphConstants
78              .getGraphicalObject((DefaultGraphCell) getEditor().getGraph())
79
80          String actionText = state.getDoActivity().toString();
81
82          final int columns = 50;
83          JTextField textField = new JTextField(actionsText, columns);
84
85          JOptionPane.showOptionDialog(
86              getEditor(), getKieLFrame().getJFrame(),
87              ResourceBundle.getString("EditStateDoActivities"),
88              textField,
89              ResourceBundle.getString("EditStateDoActivities"),
90              JOptionPane.OK_CANCEL_OPTION,
91              ResourceBundle.getBundle("Ok"),
92              ResourceBundle.getBundle("Cancel"));
93
94          if (answer == 0) {
95              state.setDoActivity(new ActionsStringLabel(textfield.getText()));
96              getEditor().getGraph().repaint();
97          }
98      }
99
100 }

```

B.7.8. ActionEditStateExitActionsForStateflow

```

package kiel.editor.controller;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import kiel.datStructure.State;
import kiel.datStructure.action.ActionsStringLabel;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.Resources.ResourceBundle;
import org.jgraph.graph.DefaultGraphCell;

/*
 * Author: apo
 * Version $Revision: 1.1 $ last modified $Date: 2005/10/28 12:03:39 $
 */
public class ActionEditStateExitActionForStateflow extends EditorAction {
    /*
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateExitActionsForStateflow() {
        super("Edit16.gif");
    }
}

/*
 * @see kiel.editor.controller.EditorAction#doAction(
 *      java.awt.event.ActionEvent)
 */
protected void doAction(final ActionEvent e) {
    State state = (State) MyGraphConstants
        .getGraphicalObject(((DefaultGraphCell) getEditor()).getGraph())
        .getSelectionCell().getAttributes();
    String actionsText = state.getExit().toString();
    final int columns = 50;
    JTextField textField = new JTextField(actionsText, columns);
    int answer = JOptionPane.showOptionDialog(
        getEditor().getOptionPane(), showOptionDialog(),
        textField,
        ResourceBundle.getString("editOnExitActions"),
        JOptionPane.OK_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null, new Object[] {
            ResourceBundle.getString("OK"),
            ResourceBundle.getString("Cancel")});
    if (answer == 0) {
        state.setExit(new ActionsStringLabel(textfield.getText()));
        getEditor().getGraph().repaint();
    }
}
}

```

B.7.9. ActionEditStateBindActionsForStateflow

```

package kiel.editor.controller;

import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import kiel.dataStructure.State;
import kiel.dataStructure.action.ActionsStringLabel;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;
import org.jgraph.graph.DefaultGraphCell;
import org.jdesktop.swingx.JTextPane;
import org.jdesktop.swingx.JTextfield;

public class ActionEditStateBindActionsForStateflow extends EditorAction {
    /**
     * @author apo
     * @version $Revision: 1.1 $ last modified $Date: 2005/10/28 12:03:39 $
     */
    public class ActionEditStateBindActionsForStateflow extends EditorAction {
        /**
         * Creates an object with an Edit16.gif image.
         */
        ActionEditStateBindActionsForStateflow() {
            super("Edit16.gif");
        }
    }
}

30     * java.awt.event.ActionEvent e) {
31     */
32     protected void doAction(final ActionEvent e) {
33
34         State state = (State) MyGraphConstants
35             .getGraphicalObject((DefaultGraphCell) getEditor().getGraph()
36                 .getSelectionCell()).getAttributes();
37
38         String actionsText = state.getBindAction().toString();
39
40         final int columns = 50;
41         JTextField textField = new JTextField(actionsText, columns);
42         int answer = JOptionPane.showOptionDialog(
43             getEditor().getKieLFframe().getJFrame(),
44             textField,
45             ResourceBundle.getString("editOnBindActions"),
46             JOptionPane.OK_CANCEL_OPTION,
47             JOptionPane.QUESTION_MESSAGE,
48             null, new Object[] {
49                 ResourceBundle.getString("OK"),
50                 ResourceBundle.getString("Cancel")
51             });
52
53         if (answer == 0) {
54             state.setBindAction(new ActionsStringLabel(textfield.getText()));
55             getEditor().getGraph().repaint();
56         }
57     }
}

```

B. Java Code

B.8. kiel.editor.graph

B.8.1. JunctionRenderer

```
10   * @version $Revision: 1.1 $ last modified $Date: 2005/10/20 12:26:22 $
11   */
12  public class JunctionRenderer extends PseudoStateRenderer {
13
14      protected void paintSelectionBorder(Final Graphics g) {
15          ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
16
17          Dimension d = getSize(); g.setColor(
18              ((MyJGraph) graph).getViewBlackView());
19
20          g.drawOval(0, 0, d.width - 1, d.height - 1);
21
22      }
23  }
```