Oberseminar:
Diplomarbeit: Erweiterung von *KIEL* um *Stateflow*-Charts

Adrian Posor

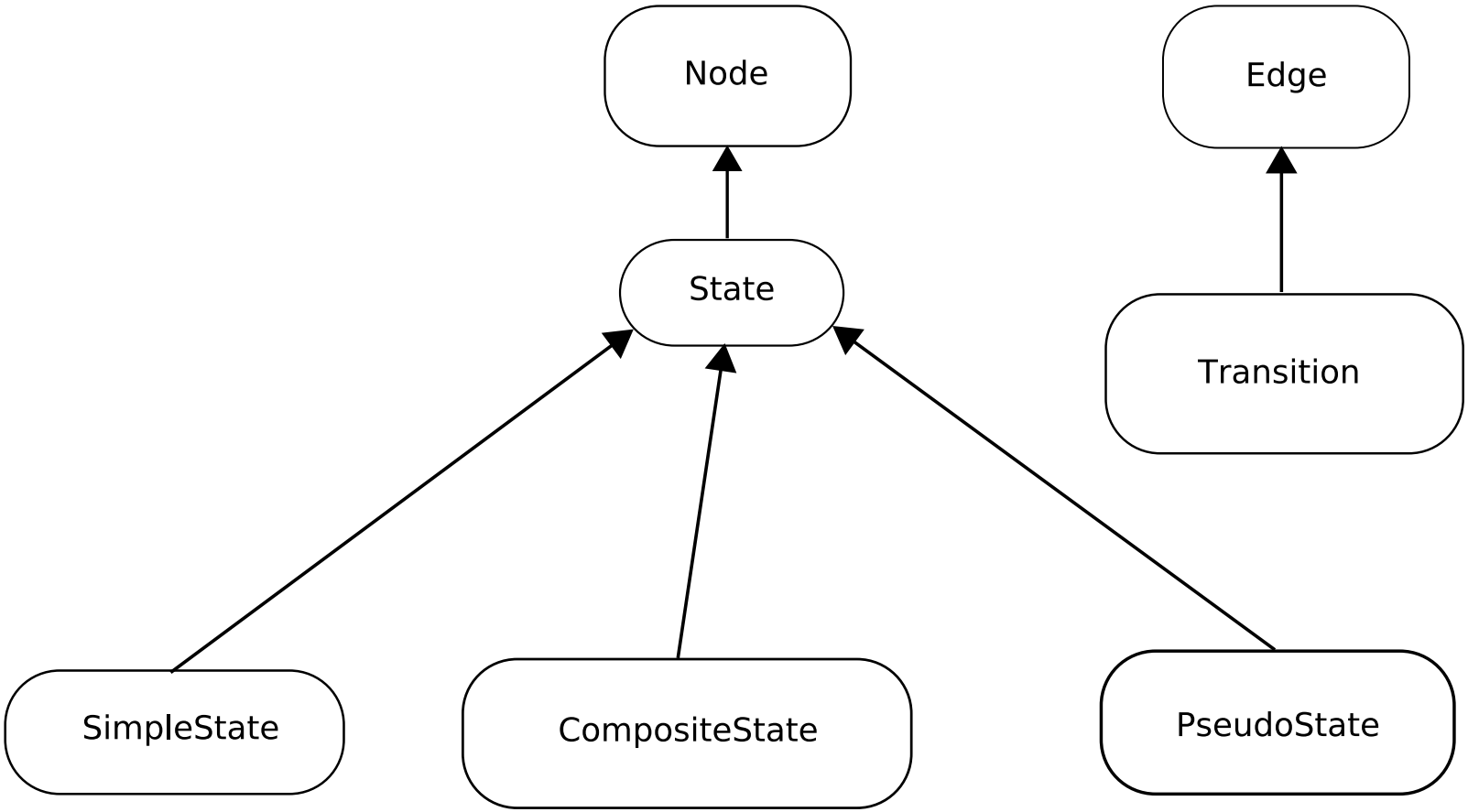
Echtzeitsysteme und Eingebettete Systeme
Institut für Informatik und Angewandte Mathematik
Christian-Albrechts-Universität zu Kiel

13. Dezember 2005

Das *KIEL*-Projekt

- Ein Werkzeug zum Erstellen von Zustandsdiagrammen
- Unterstützt verschiedene Dialekte von Statecharts
- Schwerpunkte: automatisches Layout, dynamische Sichten bei Simulation

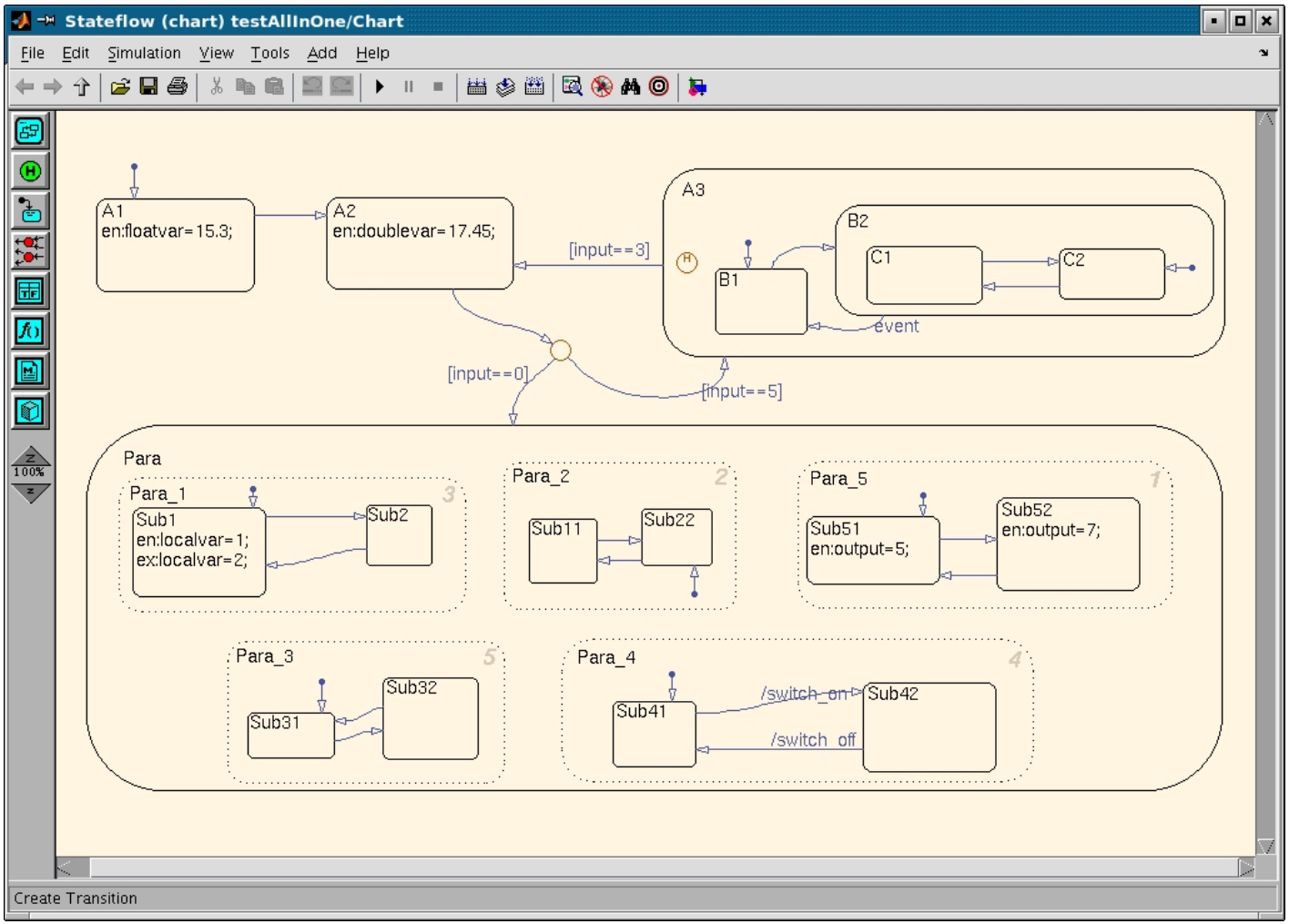
KIEL Datenstruktur



Matlab/Simulink/Stateflow

- Software zur zahlenmäßigen Lösung mathematischer Probleme
- Rechnet mit Matrizen
- Enthält eigene Programmiersprache
- *Simulink* ist Erweiterung von *Matlab*
  - Modellierung, Simulation, und Analyse dynamischer Systeme
- *Stateflow* ist Erweiterung von *Simulink*
  - Werkzeug zur Modellierung des Verhaltens reaktiver Systeme
  - Implementiert Dialekt von Harels Statecharts

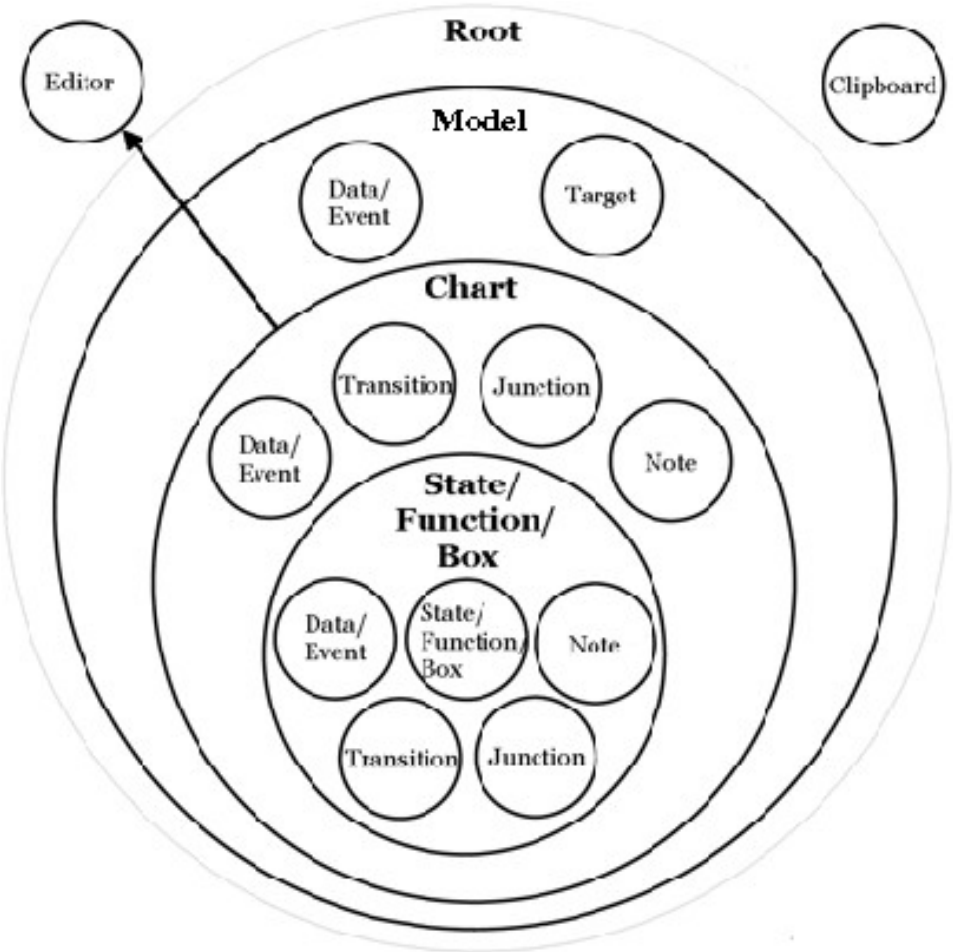
Stateflow



Was ist die Stateflow-API?

- Ermöglicht Erstellen und Ändern von *Stateflow*-Diagrammen von Kommandozeile aus
- Ermöglicht Automatisierung von Diagrammbearbeitung durch Skripte
- Für jedes Objekt in *Stateflow* gibt es ein Objekt in der API
- API-Objekt hat Methoden und Eigenschaften
- Änderungen der Eigenschaften von API-Objekten wirken sich unmittelbar auf *Stateflow*-Objekte aus

Aufbau der Stateflow-API



Methoden und Eigenschaften von API-Objekten

- Zugriff auf Objekte mittels Handles
- Zugriff auf Eigenschaften und Methoden durch sogenannte Punktnotation

Beispiele:

```
label1 = transition.LabelString
sA1.Name = 'A1'
sA1.Position = [80 120 90 60]
```

Finden von Objekten

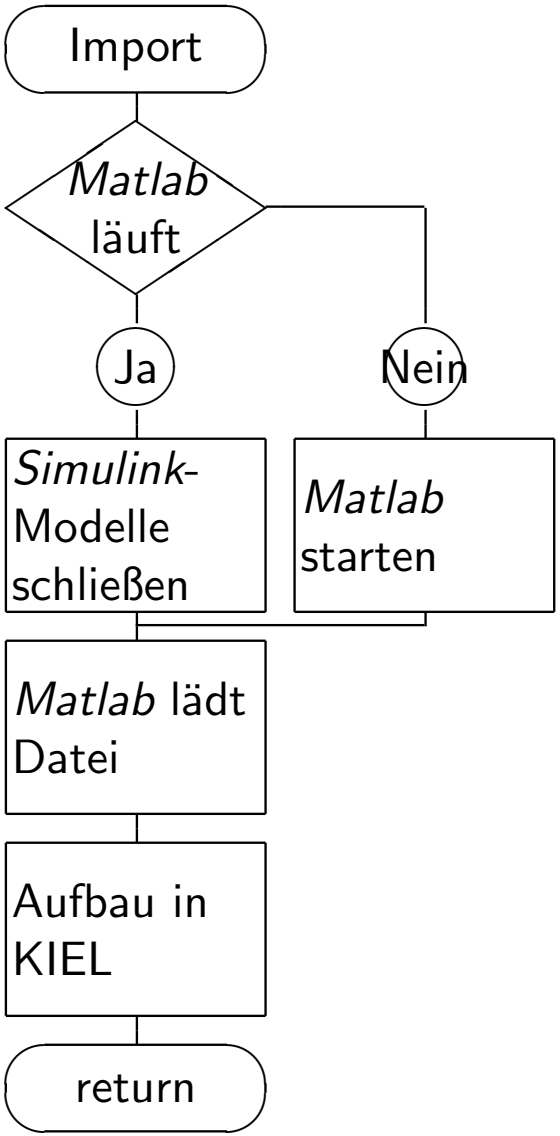
- Jedes Objekt hat eine Methode `find`
- Angabe von
  - Typ des Objektes
  - Optional: Paare von Eigenschaftsnamen und -werten

Beispiel:

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

Liefert alle Zusände im Model `m` deren Name `On` ist.

Generelle Vorgehensweise



Strategie zum Import

- Elemente des Diagramms bilden aufgrund der Hierarchie einen Baum
- Traversieren des Baums beginnend vom Root-Objekt nach der Methode der Breitensuche
- Erzeugen aller Zustände einer Ebene vor Erzeugen aller Transitionen einer Ebene
- Erzeugen von Trennlinien zum Abgrenzen von Regionen
- Umrechnen von Koordinaten (o'clock u.s.w.)

Beispiel


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

```
ans =

     312.6471   247.3765    90.0000    60.0000

>>
```

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

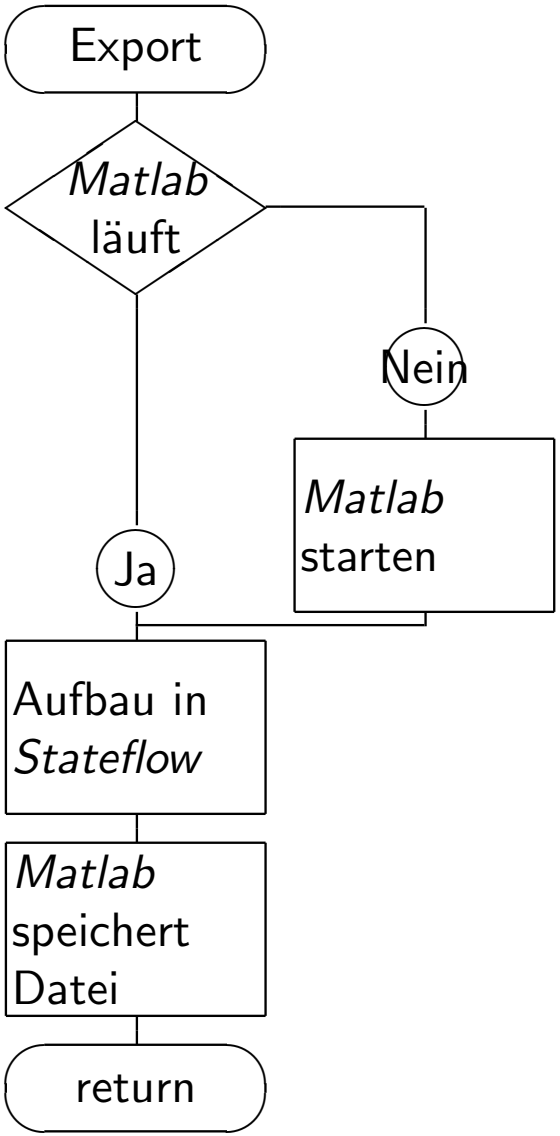
```
ans =

     OR

>>
```

Demonstration

Generelle Vorgehensweise



Strategie zum Export

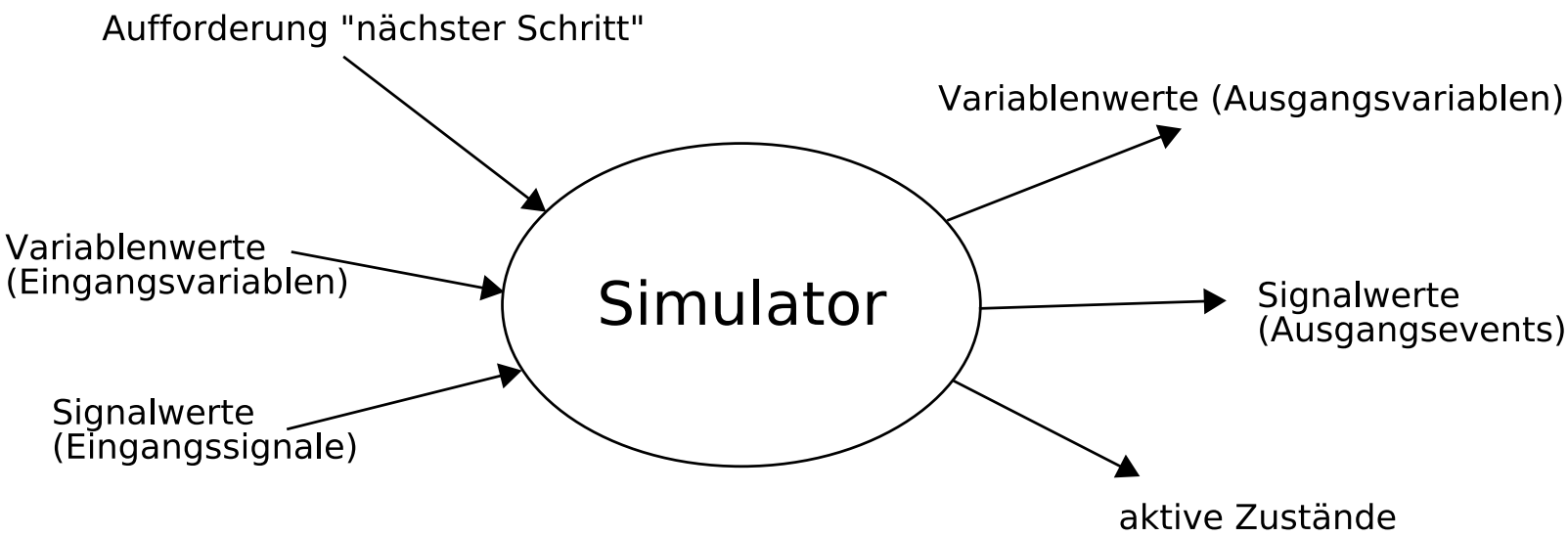
- Traversieren des Baumes beginnend vom Objekt `StateChart` nach der Methode der Breitensuche
- Erzeugen aller Zustände einer Ebene vor Erzeugen aller Transitionen einer Ebene
- Umrechnen von Koordinaten (o'clock u.s.w.)

Demonstration

Anwendung

1. Setzen von Variablen und Signalen/Ereignissen
2. Klicken auf
   - MacroStep
   - Reset
3. Aktive Zustände, Variablenwerte, Signale/Ereignisse werden angezeigt

Prinzip des Simulators



Probleme

- Matlab simuliert von Zeitpunkt 0 bis Zeitpunkt n
- Diagramm wird ausgewertet wenn
  - sich Werte von Eingangsvariablen ändern oder
  - mindestens ein Ereignis auftritt oder
  - immer zur festgelegten Sampletime
- KIEL simuliert einzelne Schritte

Lösung

- Setze Sampletime auf 1.0
- Sei n die aktuelle Schrittnummer
- Simuliere immer von Zeitpunkt 0 bis n
- Zeige nur Ergebnis von Schritt (n-1) nach Schritt n an.

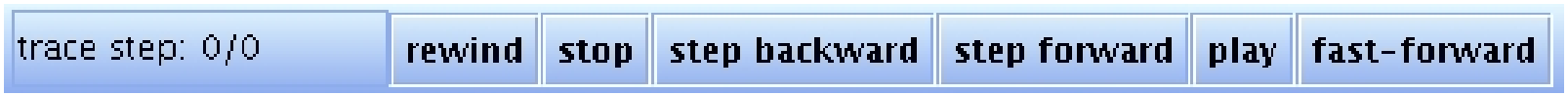
Ereignisse und Signale

- Ereignisse sind ein Konzept von *Stateflow*
- Signale sind ein Konzept von *Simulink*
- KIEL simuliert wahlweise mit Ereignissen oder Signalen als Eingabe (aus Benutzersicht)

Ereignisse als Eingabe

- Eingangssignale lösen Ereignisse aus
  - Trigger: Either, Falling, Rising
- Keine direkte Eingabe von Ereignissen möglich
- Lösung:
  - Setze alle Trigger auf "Either"
  - Löse Ereignis durch Wechsel des Signalwertes zwischen -1 und 1 aus

Anwendung



- Aufzeichnen aller Simulationseingaben
- Abspielen aller bisherigen Simulationsschritte

Demonstration

Anpassungen

- Angepasste Werkzeugleiste
- Angepasste Kontextmenüs
- Neue Dialogfenster zum Bearbeiten von Ereignissen und Variablen

Demonstration

Ende

Danke für Ihre Aufmerksamkeit!

Aufbau der Stateflow-API

Root:

- Umfaßt alle anderen Objekte
- Dient zur Abgrenzung von Objekten aus anderen Werkzeugen

Editor:

- Kein Gegenstück in *Stateflow*
- Bietet Zugriff auf rein graphische Aspekte des Diagramms
- Ein Editor-Objekt pro Diagramm

Clipboard:

- Kein Gegenstück in *Stateflow*
- Stellt Funktionalität zum Kopieren und Einfügen bereit

Aufbau der Stateflow-API

Model:

- Repräsentiert *Simulink*-Modell aus der Sicht von *Stateflow*
- Kann mehrere *Stateflow*-Diagramme enthalten

Chart:

- Repräsentiert ein *Stateflow*-Diagramm

State/Function/Box:

- Elemente, aus denen ein *Stateflow*-Diagramm besteht
- Enthält Objekte vom Typ State, Function, Box, Note, Junction, Transition, Data, Event

Erzeugung von Stateflow-Objekten

Erzeugung von Stateflow-Objekten mittels Konstruktoren

- Stateflow.Chart
- Stateflow.State
- Stateflow.Transition
- Stateflow.Junction
- Stateflow.Data
- Stateflow.Event
- Stateflow.Box
- Stateflow.Note

Generelle Vorgehensweise

1. Starten von *Matlab* oder schließen aller geöffneten *Simulink*-Modelle
2. *Matlab* die gewünschte Datei laden lassen mit dem Befehl `open_system('filename')`
3. Abfragen der relevanten Eigenschaften aller *Stateflow*-Objekte und Aufbauen eines equivalenten Diagramms mithilfe der *KIEL*-Datenstruktur

Generelle Vorgehensweise

1. Starten von *Matlab* falls noch nicht geschehen
2. Abfragen der relevanten Eigenschaften aus den Objekten der *KIEL*-Datenstruktur und Aufbauen eines equivalenten Diagramms mithilfe der *Stateflow*-API
3. *Matlab* das Diagramm abspeichern lassen mit dem Befehl `sfsave('untitled','filename')`

Vorbereitung der Simulation

1. Löschen aller Blöcke und Linien im *Simulink*-Diagram mit Ausnahme des Chart-Blockes
2. Auswählen des diskreten Lösers mit fester Schrittweite 1.0
3. Setzen der Eigenschaft `HasOutputData` auf `true` für alle Zustände
4. Hinzufügen eines To Workspace-Blockes für alle booleschen Signale (inklusive Mux)
5. Hinzufügen eines From Workspace-Blockes für alle in den Trigger-Port führenden Signale
6. Hinzufügen eines From Workspace-Blockes pro Eingangs-Variable
7. Setzen der Eigenschaft `SaveToWorkspace` auf `true` für alle Ausgangs-Variablen

Simulationsschritt I

1. Generieren einer $(m \times n)$-Matrix mit Schrittnummer in $(i, 1)$ und Signalwert in $(i, j), 1 < i <= m, 1 <= j <= n$ für n Schritte und $(n - 1)$ Signalen
2. Generieren einer $(m \times 2)$-Matrix mit Schrittnummer in $(i, 1)$ und Variablenwert in $(i, 2), 1 <= i <= m$
3. Setzen der Stoppzeit auf die aktuelle Schrittnummer
4. Simulieren mit dem Befehl `sim`
5. Erzeugen eines Objektes vom Typ
   - `StateActivated` für jeden im letzten Schritt aktivierten Zustand
   - `StateDeactivated` für jeden im letzten Schritt deaktivierten Zustand

Simulationsschritt II

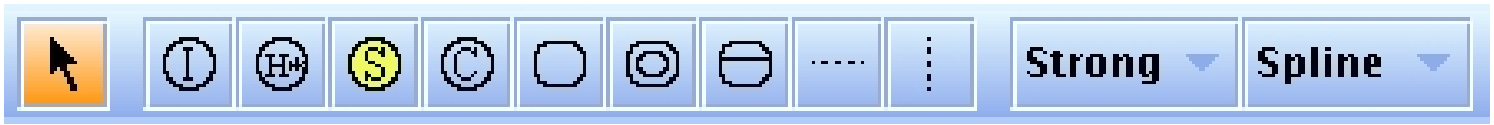
6. Erzeugen eines Objektes vom Typ
   - `SignalPresent` für jedes vorhandene Event
   - `SignalAbsent` für jedes nicht vorhandene Event
7. Erzeugen eines Objektes vom Typ `VariableValue` für jede Variable deren Wert sich im letzten Schritt geändert hat

Implementierung

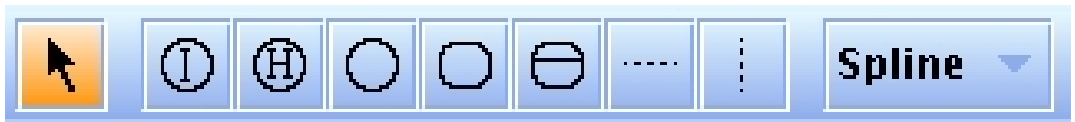
- Klasse `TraceData` verwaltet einzelne Simulationsschritte
- Instanz der Klasse `TraceStep` beinhaltet einen Simulationsschritt
- `TraceStep` enthält Objekte vom Typ
  - `Signal`/`IntegerSignal`/`Event` für Events/Signale
  - `VariableValue` für Änderungen von Variablenwerten

Werkzeugleisten

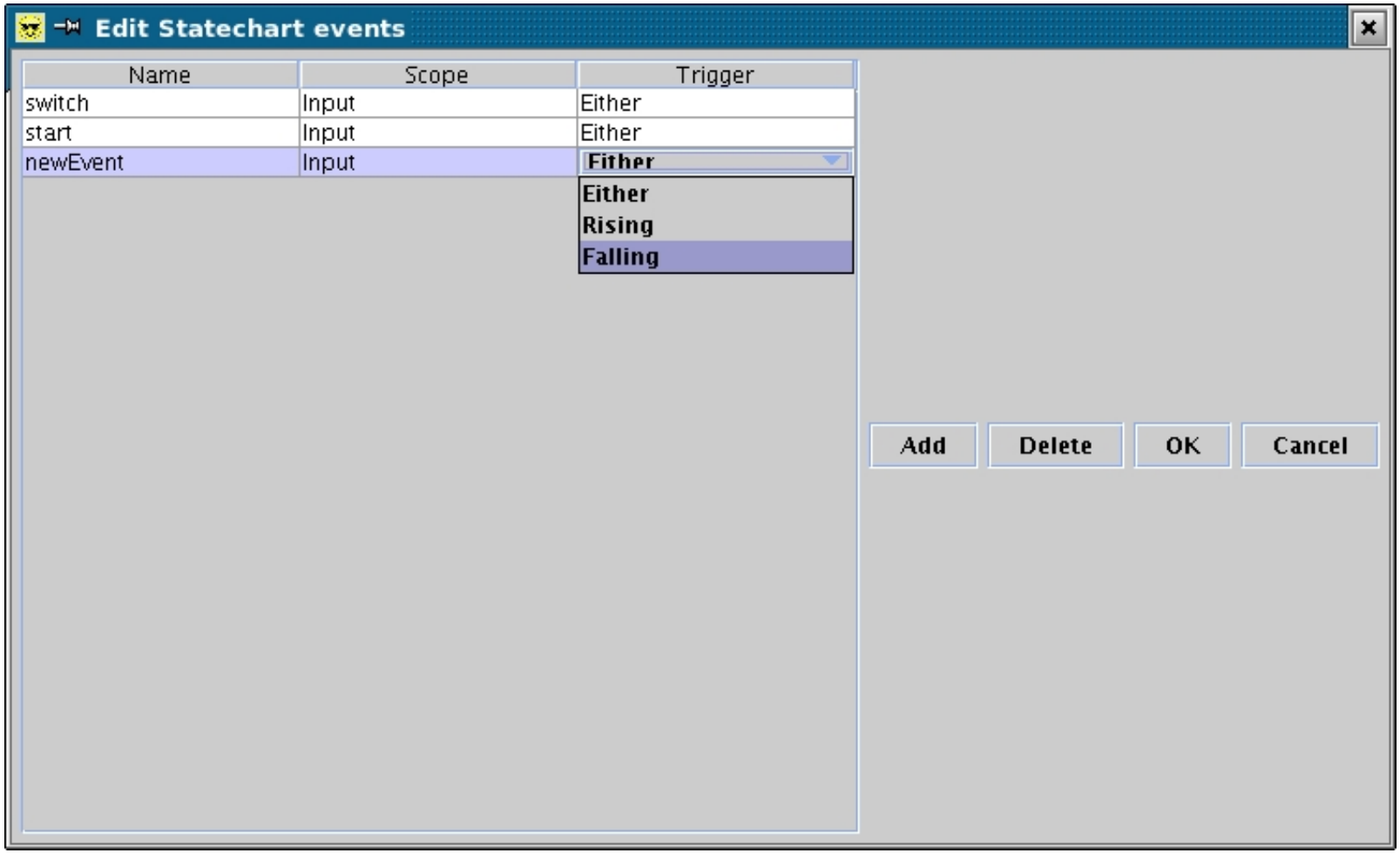
Esterel Studio:



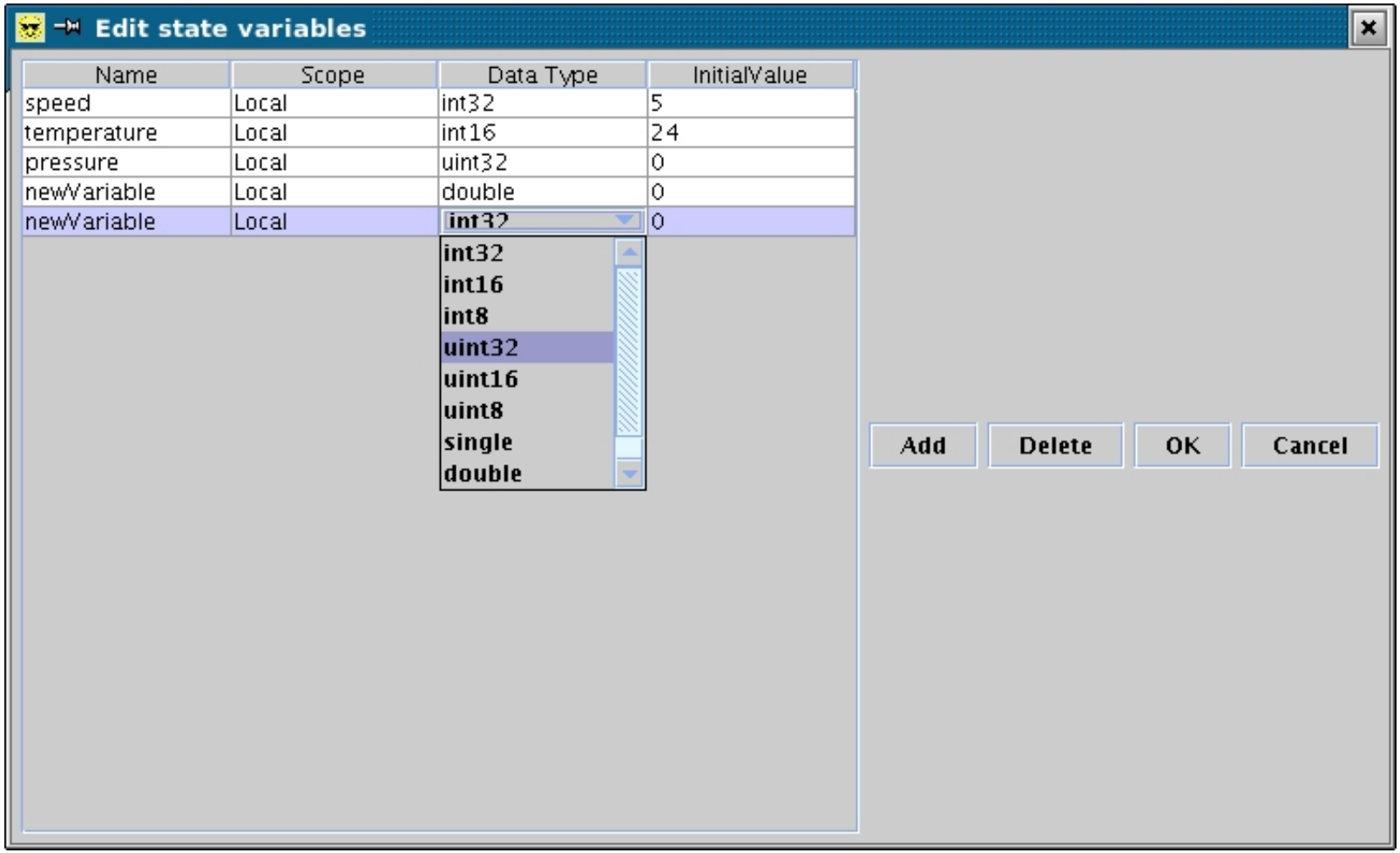
Stateflow:



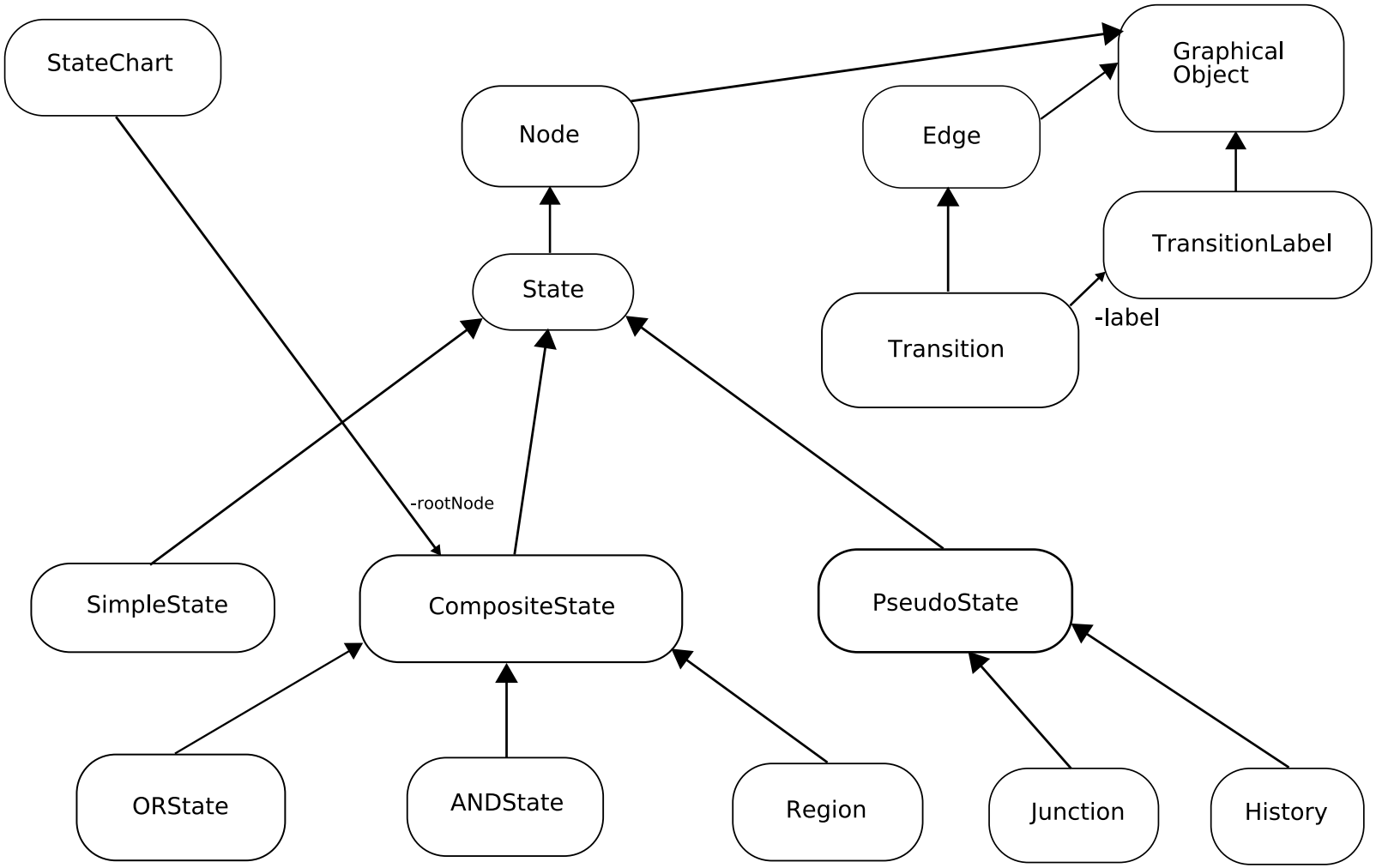
Neue Dialogfenster

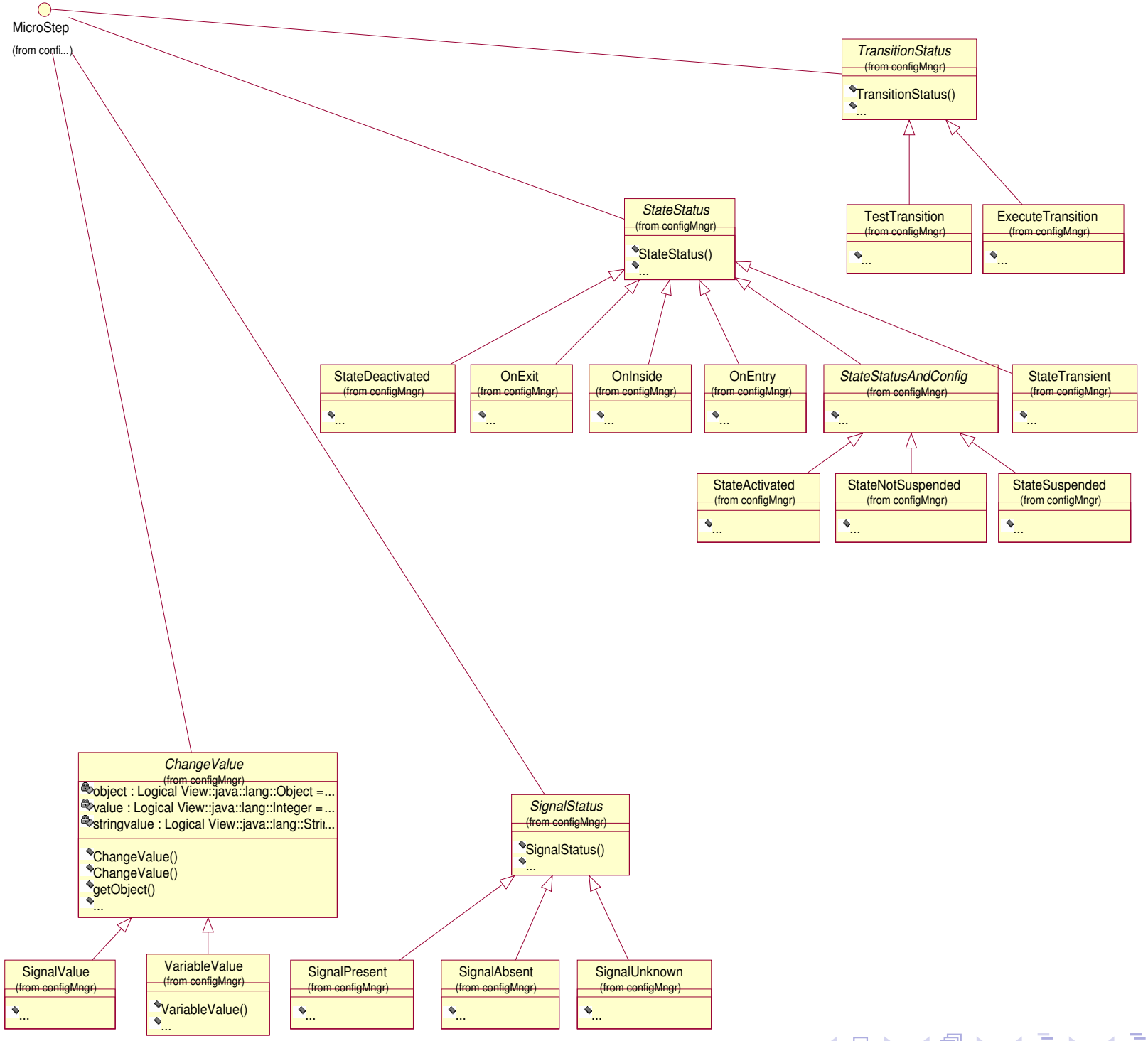


Neue Dialogfenster



KIEL Datenstruktur




MicroStep
(from confi...)
TransitionStatus
(from configMngr)
TransitionStatus()
...
StateStatus
(from configMngr)
StateStatus()
...
TestTransition
(from configMngr)
...
ExecuteTransition
(from configMngr)
...
StateDeactivated
(from configMngr)
...
OnExit
(from configMngr)
...
OnInside
(from configMngr)
...
OnEntry
(from configMngr)
...
StateStatusAndConfig
(from configMngr)
...
StateTransient
(from configMngr)
...
StateActivated
(from configMngr)
...
StateNotSuspended
(from configMngr)
...
StateSuspended
(from configMngr)
...
ChangeValue
(from configMngr)
object : Logical View::java::lang::Object =...
value : Logical View::java::lang::Integer =...
stringvalue : Logical View::java::lang::Stri...
ChangeValue()
ChangeValue()
getObject()
...
SignalStatus
(from configMngr)
SignalStatus()
...
SignalValue
(from configMngr)
...
VariableValue
(from configMngr)
VariableValue()
...
SignalPresent
(from configMngr)
...
SignalAbsent
(from configMngr)
...
SignalUnknown
(from configMngr)
...