CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL

Bachelor-Projekt

# Verwendung dynamisch erzeugter KIEL-Statecharts in LaTeX

Karsten Heymann

April 2007

Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

betreut durch:

Steffen H. Prochnow

## Eidesstattliche Erklärung

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

Kiel,

_____

# Danksagung

Viele Leute haben bei der Entstehung dieser Arbeit geholfen. Zu allererst möchte ich meinem Betreuer Steffen Prochnow für seinen unermüdlichen Einsatz danken. Erst durch seine unzähligen Anregungen wurde diese Arbeit zu dem, was sie geworden ist.

Ganz besonders danken möchte ich meinem Korrekturleser Mike Gabriel, der innerhalb kürzester Zeit die ganze Arbeit gelesen hat und dessen Anregungen die sprachliche Qualität der Arbeit hoffentlich haben steigen lassen.

Ulrich Schwarz, mit dem ich die Begeisterung für LaTeX teile und der mehr über TeX weiß als ich jemals lernen werde, danke ich für das Korrekturlesen und die sehr hilfreichen Diskussionen über das Innenleben des TeX-Compilers.

Heiko Oberdiek und Ulrich Dietz danke ich für ihre Beiträge auf meine Fragen in der Newsgroup `de.comp.text.tex`.

Ich möchte auch meinen Arbeitgebern am Ökolgiezentrum und am Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion sehr dafür danken, dass mir während der heißen Phase der Fertigstellung große Flexibilität bei den Arbeitszeiten eingeräumt wurde.

Ich danke meinem Lehrer Heinz Sander dafür, dass er mir schon in der Schule die Faszination von LaTeX nähergebracht hat, die schlussendlich zu dieser Arbeit führte.

Zuletzt, aber nicht zuwenigst, danke ich meiner wunderbaren Frau Daniela dafür, dass sie mich trotz eigener Belastungen in der Endphase der Arbeit rückhaltslos unterstützt hat!

# Inhaltsverzeichnis


**1 Einleitung** **15**

**2 Grundlagen** **17**
2.1 Statecharts . . . . . 17
2.2 Das Kiel Integrated Environment for Layout . . . . . 21
2.2.1 Funktionalität . . . . . 21
2.2.2 Implementierung . . . . . 23
2.2.3 Datenstruktur . . . . . 24
2.2.4 Module . . . . . 24
2.2.5 Benutzerschnittstellen . . . . . 24
2.3 XML als Speicherformat . . . . . 25
2.3.1 XML Data Binding mit JAXB . . . . . 25
2.3.2 Spezifizierung von XML-Dokumenten mit XML-Schema . . . . 27
2.4 LaTeX als Programmierumgebung . . . . . 28
2.4.1 Verfassen von Texten mit LaTeX . . . . . 29
2.4.2 Begriffsdefinitionen . . . . . 29
2.4.3 Verfassen von Texten mit LaTeX . . . . . 30
2.4.4 Überblick über die Arbeitsweise des TeX-Prozessors . . . . . 31

**3 Entwurf der LaTeX-Schnittstelle für KIEL-Statecharts** **33**
3.1 Analyse der bisherigen Nutzung . . . . . 33
3.2 Resultierende Anforderungen . . . . . 37

**4 Implementierung** **39**
4.1 Das LaTeX-Modul KielTeX . . . . . 39
4.1.1 Funktionaler Überblick . . . . . 39
4.1.2 Einbindung von Statecharts . . . . . 42
4.1.3 Verarbeitung von Befehlsparametern . . . . . 43
4.1.4 Erzeugung von *Jobfiles* . . . . . 45
4.1.5 Aufrufen von `kielcmd`, `ps2eps` und `epstopdf` . . . . . 45
4.1.6 Darstellung der Statecharts . . . . . 46
4.2 Erweiterung von KIELs Kommandozeilenschnittstelle KielCmd . . . . . 47
4.2.1 Setzen von KIEL-Programmeinstellungen . . . . . 47
4.2.2 Verarbeitung von Job-Files . . . . . 47
4.2.3 Adressierung von Statechart-Konfigurationen . . . . . 48

4.2.4 Markierung ausgewählter Zustände . . . 49
4.2.5 Zwischenspeicherung von erzeugten Statecharts . . . 49
4.3 Das Modul kiel.preferences . . . 50
4.3.1 Die `XMLPreferences`-Klasse . . . 51
4.3.2 Modulweiser Zugriff auf KIELs Properties . . . 52
4.3.3 Generisches Setzen von Optionen . . . 53
4.3.4 Der Konfigurationsdialog . . . 54

**5 Ergebnisse** **57**
5.1 Laufzeitanalyse . . . 57
5.1.1 Durchführung . . . 57
5.1.2 Ergebnis der Laufzeitmessung . . . 58
5.2 Demonstration von KielTEX . . . 62

**6 Zusammenfassung und Ausblick** **71**

**7 Literaturverzeichnis** **73**

**A Tabeller aller Properties von KIEL** **77**

**B Implementierungshinweise für Autoren neuer KIEL-Module** **87**
B.1 Erstellung der XML-Dateien . . . 87
B.2 Anpassung des Build-Skriptes . . . 88
B.3 Die Property-Klasse des Moduls . . . 90

**C Quellcode** **93**
C.1 Handbuch und Quellcode von KielTEX . . . 93
C.2 Das Modul kiel.preferences . . . 110
C.2.1 XMLPreferences.java . . . 111
C.2.2 XMLConfigDialog.java . . . 119
C.2.3 XMLEditPane.java . . . 122
C.2.4 XColorChooser.java . . . 124
C.2.5 FixedComboBox.java . . . 125
C.2.6 PreferencesMessage.java . . . 126
C.2.7 JAXPValidator.java . . . 127
C.2.8 TypeConverters.java . . . 128
C.3 XML-Converter-Skripte . . . 130
C.3.1 generate-properties.sh . . . 131
C.3.2 csv2xml.py . . . 132

# Tabellenverzeichnis

4.1 Aus dem Beispieldokument erzeugte *Jobfiles* . . . . . 41
4.2 Setzen und Überschreiben von KIEL-Einstellungen in KielTEX . . . . . 45

5.1 Messergebnisse der experimentellen Laufzeituntersuchungen . . . . . 59

# Abbildungsverzeichnis


2.1 Statechart einer Fußgängerampelschaltung . . . . . 18
2.2 Das ABRO-Statechart . . . . . 18
2.3 Verschiedene Layouter von KIEL, angewendet auf das gleiche Statechart 22
2.4 Tastaturbasiertes Bearbeiten von Statecharts in der KIEL-GUI, zum Bearbeiten können Zustände markiert werden . . . . . 23
2.5 Statechart vor und nach der Optimierung . . . . . 23
2.6 Module von KIEL . . . . . 24
2.7 Elemente des XML Data Bindings (angelehnt an [37]) . . . . . 27
2.8 Übermittlung von Informationen zwischen aufeinanderfolgenden TEX-Durchläufen . . . . . 32

3.1 Arbeitsablauf beim Einbinden von KIEL-Statecharts in LATEX . . . . . 34
3.2 Aufbau von KIELs Konfigurationsdateien . . . . . 35
3.3 Statechart-Simulation in KIEL (statische Ansicht) . . . . . 36
3.4 Erzeugen von Statechart-Konfigurationen durch Simulation . . . . . 36
3.5 Statechart-Simulation mit und ohne dynamischer Anzeige . . . . . 37

4.1 Ablauf der Statechart-Erzeugung durch KielTEX (Überblick) . . . . . 40
4.2 Erzeugung eines Statecharts durch KielTEX . . . . . 40
4.3 Aufbau von KIELs neuem Konfigurationssystem . . . . . 50
4.4 Der KIEL-Konfigurationsdialog . . . . . 54

5.1 Übersicht über die Ergebnisse aller Laufzeitmessungen, mit und ohne Einbeziehung der Laufzeiten der nachträglichen Konvertierungen . . . 60
5.2 Laufzeit der `bintree`-Reihe . . . . . 60
5.3 Laufzeit der `quadtree`-Reihe . . . . . 60
5.4 Laufzeit der `Token-Ring`-Reihe . . . . . 61
5.5 Laufzeit der `EstBench`-Statecharts . . . . . 61
5.6 KielTEX-Demonstration: Steuerung der Darstellungsgröße von Statechart-Abbildungen im Text . . . . . 63
5.7 KielTEX-Demonstration: Demonstration des DotLayouters (1), HVLayouters (2) und CircoLayouters (3) . . . . . 64
5.8 KielTEX-Demonstration: Modifikation der Layout-Parameter des DotLayouters (Horizontale und vertikale Orientierung) . . . . . 65

5.9 KielTEX-Demonstration: Auswahl von Konfigurationen über Angabe aktiver Zustände, mit und ohne dynamischer Ausblendung nichtaktiver hierarchischer Zustände . . . . . 66
5.10 KielTEX-Demonstration: Einfaches (`bintree-1`) und komplexes (`bintree-10`) Statechart . . . . . 67
5.11 KielTEX-Demonstration: Aus Esterel generiertes Statechart ohne und mit Optimierung . . . . . 68
5.12 KielTEX-Demonstration: Einbettung von Statechart-Quellcode (ABRO) im LATEX-Dokument . . . . . 69
5.13 KielTEX-Demonstration: Eingebettetes ABRO-Esterel-Statechart, dargestellt als Quelltext . . . . . 70

# Verzeichnis der Abkürzungen

**API** *Application Programming Interface*
**CSV** *Comma Separated Value*
**DOM** *Document Object Model*
**DVI** *Device Independant*
**EPS** *Encapsulated PostScript*
**GUI** *Graphical User Interface*
**HTML** *Hyper Text Markup Language*
**JAXB** *Java Architecture for XML Binding*
**KIEL** *Kiel Integrated Environment for Layout*
**KIT** *KIEL statechart extension of doT*
**MVC** *Model-View-Controller*
**MVL** *Main Vertical List*
**OMG** *Object Management Group*
**PDF** *Portable Document Format*
**PS** *PostScript*
**SAX** *Simple API for XML*
**SSM** *Safe State Machine*
**UML** *Unified Modeling Language*
**XML** *Extensible Markup Language*

# 1 Einleitung

Der modellbasierte Entwurf reaktiver Systeme ist seit längerem ein eigenständiges Forschungsthema in der Informatik [31]. Eine inzwischen weitreichend untersuchte Möglichkeit zur Beschreibung reaktiven Verhaltens sind Statecharts [17], für welche auch am Markt eine Reihe von Werkzeuge existieren (z. B. *StateMate* [18], *Rational Rose* [33], *Matlab Simulink/Stateflow* [40] und *Esterel Studio* [11]). Als ursprünglich grafischer Formalismus werden Statecharts üblicherweise mit grafischen Werkzeugen modelliert. Es existieren aber auch Ansätze, Statecharts mit textuellen Verfahren zu beschreiben [**?**].

Im technischen und wissenschaftlichen Umfeld ist das Textsatzprogramm LaTeX [23] weit verbreitet. Wollte man bisher Statecharts in LaTeX-Dokumenten verwenden, so mussten diese in einem grafischen Werkzeug modelliert und anschließend als Bilddateien in den Text eingefügt werden. Dieser Vorgang erfordert viele manuelle Schritte, insbesondere, wenn aus einem Statechart-Modell verschiedene Ansichten mit unterschiedlichen Parametern dargestellt werden sollen.

Diese Arbeit beschreibt ein Verfahren, welches die Verwendung von Statechart-Abbildungen in LaTeX-Texten vereinfacht. Dafür wird ein LaTeX-Paket erstellt, welches die direkte Einbindung von Statecharts in den Dokumenttext ermöglicht und die Erzeugung von Abbildungen aus diesen Statecharts weitestgehend automatisiert.

Für die Erzeugung der Abbildungen wird das am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme der CAU Kiel entstandene Statechart-Werkzeug *Kiel Integrated Environment for Layout* (KIEL) verwendet. KIEL ist in Java [36] geschrieben und wurde mit dem Ziel entwickelt, neue Verfahren zur effizienten Modellierung komplexer Statecharts zu erproben. Ein Schwerpunkt ist das automatische Layouten von Statecharts [21]. Um die Datenbasis für die implementierten Verfahren zu erweitern, verfügt KIEL über die Möglichkeit, Statechart-Modelle aus anderen Werkzeugen zu importieren.

Für KIEL existieren zwei Benutzerschnittstellen: Eine grafische Arbeitsumgebung und eine Kommandozeilenschnittstelle. Da KIEL primär als grafische Arbeitsumgebung konzipiert ist, steht ein Teil der Funktionalität nur hier zur Verfügung. Für die LaTeX-Anbindung soll allerdings die KIEL-Kommandozeilenschnittstelle verwendet werden. Im Rahmen dieser Arbeit wird die Kommandozeilenschnittstelle daher um bisher nur

über die grafische Schnittstelle erreichbare Funktionen erweitert, die einen größeren Funktionsumfang der LaTeX-Schnittsteller ermöglichen, beispielsweise um die Möglichkeit, die aktiven Zustände eines Statecharts zu spezifizieren.

Das Verhalten von KIEL bei der Erzeugung von Statecharts wird durch KIELs Programmeinstellungen beeinflusst. Um die Flexibilität bei der Erzeugung von Statecharts zu erhöhen, sollen die Einstellungen für jedes Statechart getrennt gesetzt werden können. Da die Architektur von KIEL keinen Zugriff über die Kommandozeilenschnittstelle auf die Einstellungen vorsieht, wird im Rahmen dieser Arbeit das Konfigurationssystem von KIEL um einen solchen Zugriff erweitert und im Zuge dieser Erweiterung auf ein *Extensible Markup Language* (XML)-basiertes Format umgestellt. Das zu entwickelnde Konfigurationssystem soll darüber hinaus einen Konfigurationsdialog für die grafische Benutzerumgebung enthalten und die Möglichkeit vorsehen, Syntaxfehler in den Konfigurationsdateien schon beim Starten von KIEL zu erkennen.

Die vorliegende Arbeit ist wie folgt gegliedert:

- In Kapitel 2 werden die für das Verständnis der Arbeit notwendigen Grundlagen vorgestellt. Es wird eine kurze Einführung in Statecharts gegeben, die verwendeten XML-Techniken werden vorgestellt und es wird ein Überblick über die Funktionsweise eines LaTeX-Übersetzers gegeben.
- In Kapitel 3 wird betrachtet, wie KIEL-Statecharts bisher in LaTeX-Dokumenten genutzt wurden. Es wird untersucht, an welchen Stellen sich der Arbeitsablauf vereinfachen lässt und es werden mögliche Umsetzungen der Schlüsse diskutiert.
- Kapitel 4 beschreibt die Implementierung des LaTeX-Pakets KielTeX und der für die Erstellung von KielTeX notwendigen Änderungen an KIEL.
- Gegenstand von Kapitel 5 sind eine Laufzeitanalyse der in Kapitel 4 vorgestellten Implementierung sowie die Demonstration des Funktionsumfangs von KielTeX anhand einer Reihe von Beispielen.
- In Kapitel 6 wird ein Überblick über die Ergebnisse der Arbeit gegeben und es werden weiterführende Ansätze diskutiert. diskutiert.

# 2 Grundlagen

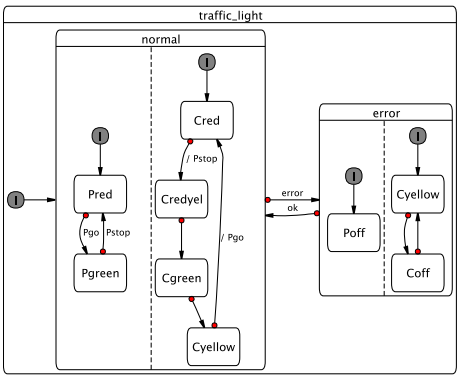
In der vorliegenden Arbeit wird die Entwicklung einer Schnittstelle zur Verwendung von Statecharts in LaTeX-Dokumenten beschrieben. Zum Verständnis der Funktionsweise einer solchen Schnittstelle ist es angeraten, sich zunächst mit den dabei beteiligten Komponenten und Konzepten zu beschäftigen. Begonnen wird mit einer Einführung in Statecharts (Abschnitt 2.1), besonders betrachtet werden dabei die für den Datenaustausch zwischen den Applikationen benötigten Statechart-Dateiformate. Zur Erzeugung von Statechart-Abbildungen werden Werkzeuge des KIEL-Projekts verwendet. Daher wird in Abschnitt 2.2 ein Überblick über den Aufbau und die bereitgestellten Funktionen von KIEL gegeben. Anschließend werden XML-Techniken betrachtet (Abschnitt 2.3), die eine einfache Verwendung von XML als validiertes Speicherformat für Programmeinstellungen ermöglichen. Wesentlich für die Beschreibung einer LaTeX-Schnittstelle ist schlussendlich eine Betrachtung von LaTeX, vor allem im Kontext der LaTeX-Programmierung (Abschnitt 2.4)

## 2.1 Statecharts

Statecharts sind ein ursprünglich von Harel [17] entwickelter syntaktischer Formalismus zur grafischen Modellierung reaktiver Systeme. Ein reaktives System ist nach Harel ein System, das ereignisgesteuert arbeitet und fortwährend auf innere und äußere Einflüsse reagiert. Ausgehend von endlichen Automaten erweiterte er diese um Konzepte wie Parallelität und Hierarchie, um das Problem der Zustandsexplosion bei der Modellierung großer Systeme zu vermeiden. Die ursprüngliche Semantik ist in *StateMate* [19] umgesetzt, daneben existiert eine Vielzahl weiterer Statechart-Dialekte mit zum Teil unterschiedlicher Syntax und Semantik; bereits 1994 stellte von der Beeck [5] 21 unterschiedliche Varianten zusammen.

Harel führte Statecharts als grafischen Formalismus ein. Ein Beispiel für ein einfaches Statechart zeigt Abb. 2.1. Es beschreibt die Schaltung der Straßen- und Fußgängersignale einer Fußgängerampel.

Neben der originären grafischen Notation von Statecharts existieren verschiedene textuelle Verfahren zur Erzeugung und Bearbeitung von Statecharts. Im Folgenden werden drei unterschiedliche Ansätze am Beispiel des ABRO-Statecharts [6] (s. Abb. 2.2) vorgestellt. ABRO demonstriert wichtige Aspekte von Statecharts: Wenn die Signale A




Erläuterung:

- `Poff`, `Pred`, `Pgreen`: Zustände der Fußgängerampel
- `Coff`, `Cred`, `Credyel`, `Cgreen`, `Cyellow`: Zustände der Straßenampel
- `normal`: Normaler Betrieb
- `error`: Fehlerbetrieb
- `/Pgo`, `/Pstop`: Beim Ausführen der Transition wird das Signal entsprechende Sigal emittiert
- `Pgo`, `Pstop`: Die Transition kann nur betreten werden, wenn das entsprechende Signal gesetzt ist.

**Abbildung 2.1:** Statechart einer Fußgängerampelschaltung

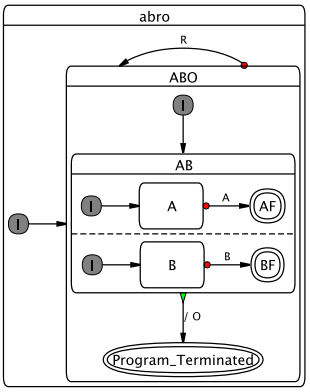



**Abbildung 2.2:** Das ABRO-Statechart

und B mindestens einmal aktiv waren, wird das Signal O emittiert. Wenn das Signal R aktiv ist, wird das Verhalten von ABRO zurückgesetzt.

Das von *Esterel Studio* [11] zum Speichern von Statecharts verwendete SCG-Format ist ein textbasiertes Format, in welchem alle Parameter bezüglich des Layouts und der Struktur eines Statecharts hinterlegt sind. Es handelt sich um ein sehr verboses Dateiformat, welches primär zur maschinellen Verarbeitung ausgelegt ist. Neben der Struktur des Statecharts enthält es beispielsweise auch Informationen über die Anordnung der Elemente sowie die verwendeten Schriften. Das vergleichsweise einfach aufgebaute ABRO-Statechart hat in SCG eine Länge von mehr als 1000 Textzeilen (s. Listing 2.1).

Auf den ersten Blick unerwartet mag erscheinen, dass auch Programmiersprachen zur Erzeugung von Statecharts verwendet werden können. Schon in [35] wird ein

**Listing 2.1:** ABRO-Statechart im SCG-Format (gekürzt)

```
#       Model of type Document saved by /home/esterel/EsterelStudio
 -5.0/bin/estudio.exe[02/09/2006 16:08:59]


GRAPH   Document
ATTRIB
{""
[]
""
0
0
""
[]
""

{
{"."
"Simulation"
"Embedded"
}
.
.
.
.


END # of model.1
END # of state.0









END # of model.0
```

Verfahren vorgestellt, Statecharts in semantisch gleichwertigen Code der synchronen Programmiersprache Esterel [10] zu überführen. Der umgekehrte Weg wird in [32] beschrieben. Hier spezifizieren die Autoren ein Verfahren, um aus Esterelv5-Code *Safe State Machine* (SSM)s zu generieren. Das vorgestellte Verfahren arbeitet zweistufig: in der ersten Stufe werden die einzelnen Esterel-Befehlsstrukturen in gleichwertige SSM-Elemente überführt, im zweiten werden durch ein Optimierungsverfahren die dabei entstehenden semantisch überflüssigen Artefakte entfernt. Listing 2.2 zeigt ABRO in Esterel.

**Listing 2.2:** ABRO-Statechart in Esterel

```
module ABRO:
input A, B, R;
output O;
loop
  [ await A || await B];
  emit O;
each R
end module
```

Das noch relativ junge *KIEL statechart extension of doT* (KIT) [30] ist eine an das DOT-Format des *graphviz*-Paketes [16] angelehnte Sprache zur Beschreibung der Topologie von Statecharts verschiedener Modellierungsansätze. In Listing 2.3 ist das ABRO-Statechart in der KIT-Syntax abgedruckt.

**Listing 2.3:** ABRO-Statechart in KIT

```
statechart abro[model="Esterel Studio";version="5.0"]{
 input A;
 input B;
 input R;
 output O;
 {
  ->ABO;
  ABO{
   AB{
     ->A;
     A->AF[type=sa;label="A"];
     AF[type=final];
    ||
     ->B;
     B->BF[type=sa;label="B"];
     BF[type=final];
   };
   ->AB;
   AB->Program_Terminated[type=nt;label="/ O"];
   Program_Terminated[type=final];
```

```
  };
  ABO->ABO[type=sa;label="R"];
 };
};
```

## 2.2 Das Kiel Integrated Environment for Layout

KIEL [39] ist eine am Lehrstuhl für Echtzeitsysteme der CAU Kiel entwickelte Softwareumgebung für das Arbeiten mit Statecharts (s. Abschnitt 2.1). Im Kontext dieser Arbeit wird es um eine LaTeX-Schnittstelle erweitert. Daher gibt dieser Abschnitt einen Überblick über bereitgestellten Funktionen und den Aufbau von KIEL.

### 2.2.1 Funktionalität

KIEL bietet als prototypische Entwicklungsplatform den Rahmen für eine Vielzahl von Forschungsaktivitäten rund um Statecharts. Einige für diese Arbeit wichtige Funktionen von KIEL werden im Folgenden beschrieben.

#### Simulieren mit dynamischer Ansicht

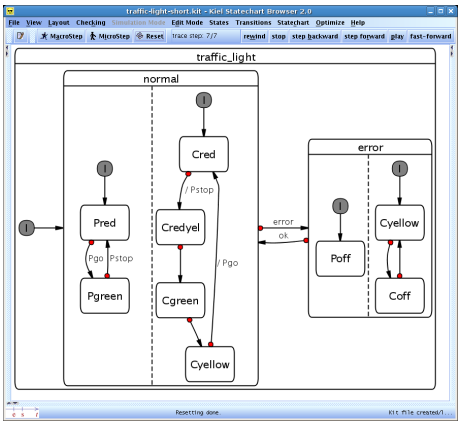
KIEL erlaubt es, Statecharts schrittweise auszuführen und während dieses Simulationsprozesses auf die Parameter Einfluss zu nehmen. Während des Simulierens können sich die angezeigten Zustände dynamisch an die gerade aktiven Zustände anpassen. Dieses, *semantic zooming* genannte Verfahren erhöht die Übersichtlichkeit bei komplexen Statecharts.

#### Automatisches Layout

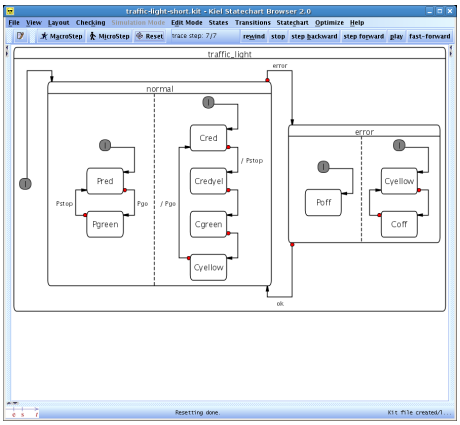
Ein Entwicklungsziel von KIEL ist das automatische Anordnen von Statecharts mit dem Ziel der optimalen Übersichtlichkeit und Verständlichkeit. Verschiedene parametrisierbare Layouter stehen hierfür zur Verfügung (s. Abb. 2.3).

#### Verschiedene Statechart-Formate

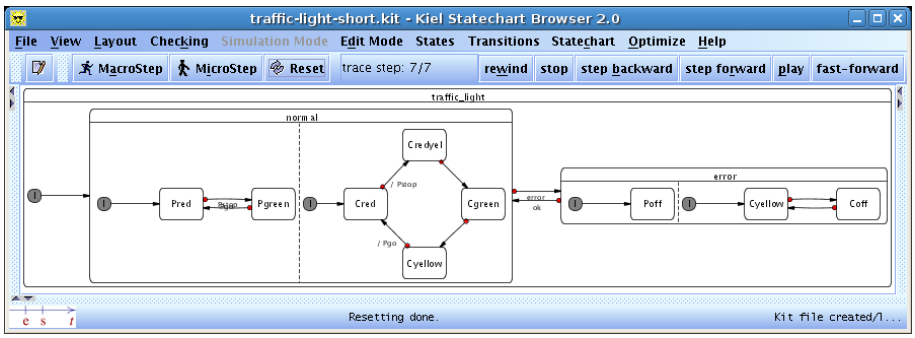
Um für die unterschiedlichen Anwendungsbereiche von KIEL ein möglichst breites Testmaterial zur Verfügung zu stellen, ist es möglich, Modelle aus anderen Werkzeugen zu importieren. Aktuell sind dies beispielsweise Esterel Studio und Matlab Simulink/Stateflow. Außerdem kann KIEL Dateien im eigens dafür entwickelten Format KIT lesen und aus anderen Formaten KIT-Code erzeugen.




**(a)** Dot-Layouter




**(b)** HV-Layouter




**(c)** Circo-Layouter

**Abbildung 2.3:** Verschiedene Layouter von KIEL, angewendet auf das gleiche Statechart

### Bearbeitung von Statecharts

KIEL stellt verschiedene Möglichkeiten zur Bearbeitung von Statecharts in Form von Modulen zur Verfügung (s. Abschnitt 2.2.4). Das *Editor*-Modul stellt eine klassische mausbasierte Arbeitsumgebung bereit. Alternativ bietet das *Browser*-Modul ein tastaturbasiertes Verfahren zur Modifikation von Statecharts (s. Abb. 2.4) sowie die Möglichkeit, Statecharts interaktiv durch Bearbeiten des angezeigten Quellcodes zu verändern. Änderungen hier nehmen direkt auf den Aufbau des Statecharts Einfluss.

### Optimierung von aus Esterel generierten Statecharts

KIEL kann Statecharts nach definierten Regeln vereinfachen (s. Abb. 2.5), ohne dass sich die Semantik des Statecharts ändert. Dies ist vor allem wichtig bei Statecharts,

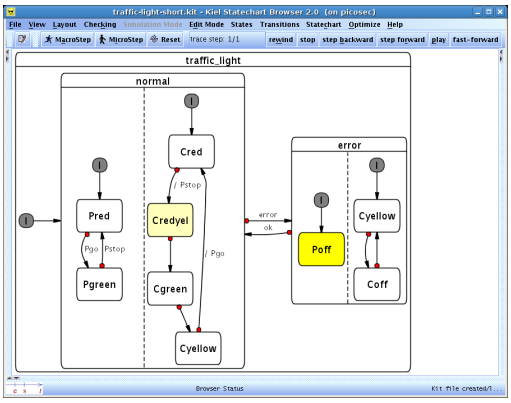
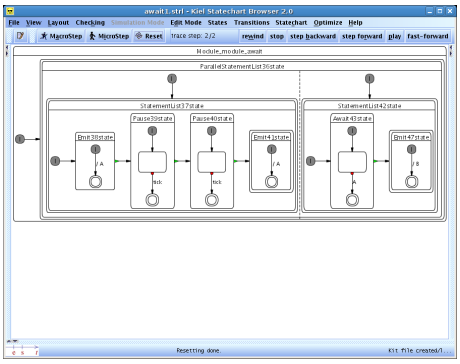
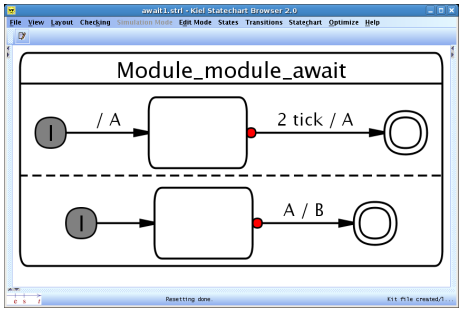



**Abbildung 2.4:** Tastaturbasiertes Bearbeiten von Statecharts in der KIEL-GUI, zum Bearbeiten können Zustände markiert werden

die aus Esterel-Quellcode generiert wurden, um durch maschinelle Konvertierung entstehenden Overhead zu vermeiden.



**(a)** Nicht optimiertes Statechart




**(b)** Optimiertes Statechart

**Abbildung 2.5:** Statechart vor und nach der Optimierung

### 2.2.2 Implementierung

Die Bestandteile von KIEL lassen sich drei großen Bereichen zuordnen:

- der Datenstruktur zur Repräsentierung von Statecharts,
- den einzelnen Modulen, und
- der grafischen Schnittstelle (*KielFrame*) und der Kommandozeilenschnittstelle (KielCmd).

### 2.2.3 Datenstruktur

Die KIEL-Datenstruktur stellt Strukturen zur Repräsentation von Statecharts bereit. Die Datenstruktur ist in drei unterschiedliche Komponenten untergliedert: Modell, Ansichten und Konfigurationen. Das im Aufbau an die UML-Spezifikation für Statecharts [28] angelehnte *Modell* speichert die logische Struktur des Statecharts sowie die Variablen und die Ein- und Ausgangssignale.

Das Modell enthält keine Informationen zum Aussehen des Statecharts, diese sind in einer oder mehrerer mit dem Modell verknüpften *Ansichten* gespeichert. Konstellationen von aktiven Zuständen werden in *Konfigurationen* hinterlegt.

### 2.2.4 Module

Die Funktionen von KIEL sind auf einzelne Module verteilt (s. Abb. 2.6), die größtenteils im Rahmen separater studentischer Projekte entstanden. Großen Wert wurde bei der Entwicklung auf eine größtmögliche Unabhängigkeit der einzelnen Module gelegt. So speichert beispielsweise jedes Modul seine Einstellungen in einer eigenen Konfigurationsdatei.

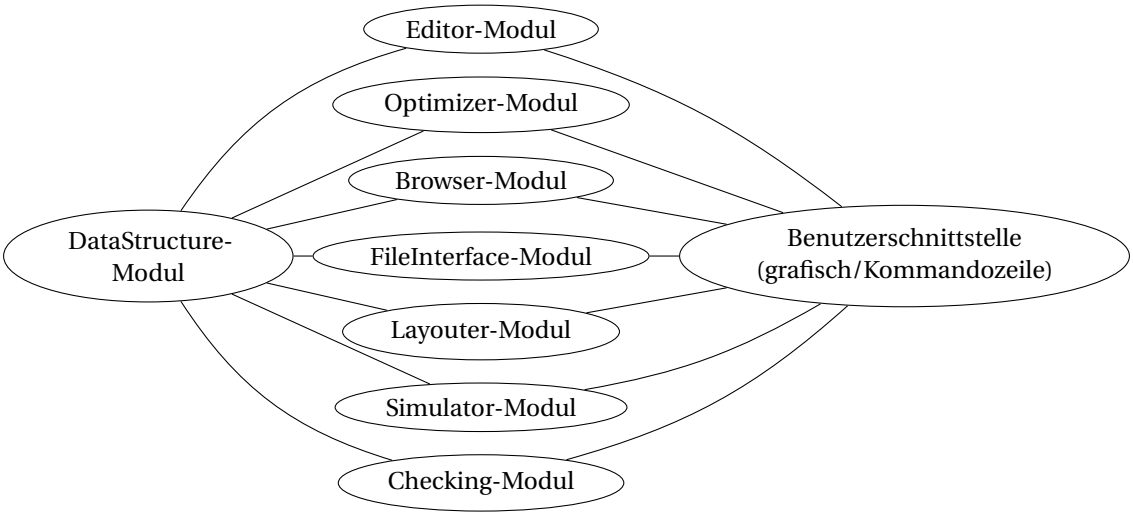



**Abbildung 2.6:** Module von KIEL

### 2.2.5 Benutzerschnittstellen

KIEL verfügt zum Zeitpunkt dieser Arbeit über zwei Benutzerschnittstellen. Die grafische Oberfläche (*KielFrame*) ist die primäre Arbeitsumgebung und stellt alle Funktio-

nen von KIEL in einer interaktiven, visuellen Umgebung bereit.

Die Kommandozeilenschittstelle (*KielCmd*) erlaubt es, Statecharts an der Kommandozeile als Abbildungen zu exportieren, zu überprüfen und zu optimieren. Nicht alle Funktionen, die über die grafische Schnittstelle benutzbar sind, lassen sich auch über die Kommandozeile verwenden.

## 2.3 XML als Speicherformat

XML ist ein universelles Format zur Repräsentation hierarchicher Daten. In dieser Arbeit wird XML als Speicherformat für die Programmeinstellungen von KIEL verwendet. Bei diesem Anwendungsfall ist die Verwendung generischer Zugriffsmethoden wie *Simple API for XML* (SAX) [26] oder *Document Object Model* (DOM) [42] zwar möglich, eine einfachere Handhabund verspricht aber das im Folgenden vorgestellte XML Data Binding. Um XML-Daten auf korrekten Aufbau und Inhalt zu überprüfen, existieren verschiedene Verfahren zur Spezifikation, wie das in Abschnitt 2.3.2 vorgestellte XML-Schema.

### 2.3.1 XML Data Binding mit JAXB

Applikationen operieren oft auf Daten, die aus externen Quellen wie Dateien, Datenbanken oder Netzwerkschnittstellen stammen. Um mit diesen Daten arbeiten zu können, müssen sie zuerst in native Objekte der jeweiligen Programmiersprache übersetzt werden. Dies trifft auch auf Daten zu, die in XML vorliegen. Gebräuchliche Schnittstellen zum Zugriff auf XML wie DOM oder SAX zwingen den Entwickler, sich mit dem strukturellen Aufbau des XML-Dokuments zu befassen. Man kann sie daher als *Low-Level-Schnittstellen* bezeichnen. Ist der Entwickler primär an den *Daten* des XML-Dokuments interessiert, so sind diese Schnittstellen umständlich zu benutzen und liefern keine Informationen über die eigentliche Bedeutung der gelesenen Daten. Diese Probleme lassen sich mit der Entwicklung eigener Zugriffs-*Application Programming Interface*s (APIs) angehen, dies ist aber erfahrungsgemäß aufwändiger.

Ein Ansatz zur Lösung dieses Problems, *XML Data Binding*, wird in [25] definiert als: „[...] der Prozess des Zuordnens von Komponenten eines Datenformats (z. B. SQL Tabellen oder XML-Schema) zu spezifischen Repräsentationen einer Programmiersprache (z. B. Java-Objekte), die die Bedeutung des Dateiformats kodieren.“ *XML Data Binding* ermöglicht es dem Entwickler also, mit den Objekten seiner Programmiersprache zu arbeiten, an Stelle von Objekten, die spezifisch für das verwendete Abfrageverfahren sind.

Verwendet man XML-Dokumente lediglich als Datenspeicher, so ist der lesende und schreibende Zugriff auf den Inhalt über *Low-Level*-APIs wie SAX oder DOM oft unnötig komplex. Wünschenswert wäre es, die Daten direkt als Objekte der Programmiersprache ansprechen zu können. Dieser Übertragungsprozess kann mit *XML Data Binding* realisiert werden. Es gibt verschiedene Toolkits, die Funktionen zum *Data Binding* bereitstellen. Eines der bekanntesten ist *Java Architecture for XML Binding* (JAXB).

Das JAXB-Framework [1] ist eine Spezifikation für *XML-Data-Binding*-Programme. Zugleich ist JAXB aber auch der Name der Referenzimplementierung für diese Spezifikation. Wenn im Folgenden von JAXB die Rede ist, ist damit die Referenzimplementierung gemeint.

Anlehnend an [14] werden beim *XML Data Binding* die folgenden vier Konzepte unterschieden (s. Abb. 2.7):

1. *Klassen-Generierung*

   Hierunter versteht man die automatische Generierung von Klassen und Schnittstellen aus der Dokumenttyp-Definition, z. B. aus XML-Schema. Die Klassen-Generierung wird bei den meisten Frameworks durch einen externen Übersetzer erledigt, bei JAXB ist dies der `xfc`.

2. *Unmarshalling*

   Unter Unmarshalling versteht man das Erzeugen von Objekten aus einem XML-Dokument. Es entsteht ein Objekt, welches das Wurzelelement des XML-Dokuments repräsentiert.

3. *Marshalling*

   Marshalling bezeichnet den Gegenprozess zum Unmarshalling, also die Überführung von Objekten in XML-Daten. Da XML-Schema-Spezifikationen Restriktionen enthalten können, die sich nicht eindeutig auf Konzepte der Programmiersprache abbilden lassen, findet im Anschluss an das Marshalling im Allgemeinen eine Validierung der erzeugten XML-Daten statt. Schlägt die Validierung fehl, so schlägt auch der Marshalling-Prozess fehl.

4. *Anpassung*

   Da es viele Möglichkeiten der Umwandlung von XML-Strukturen in Objekte der Programmiersprache gibt, sehen Data-Binding-Frameworks eine Einflussnahme auf den Umwandlungsprozess vor. So kann z. B. aus einem Feld, welches die textuelle Repräsentation eines Farbwertes enthält, beim Unmarshalling direkt ein Color-Objekt erzeugt werden.

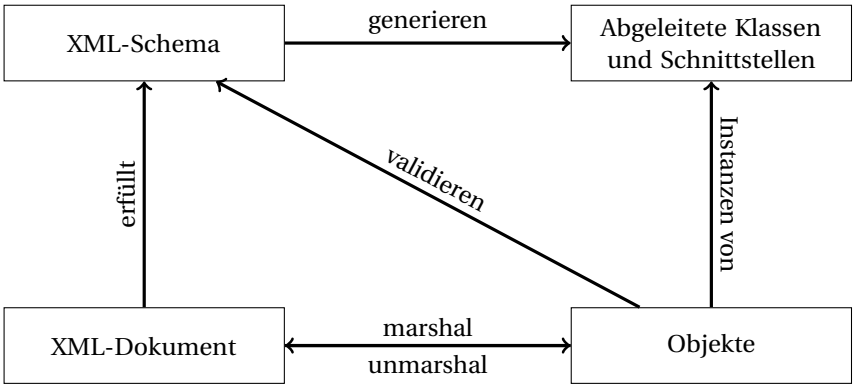



**Abbildung 2.7:** Elemente des XML Data Bindings (angelehnt an [37])

### 2.3.2 Spezifizierung von XML-Dokumenten mit XML-Schema

Wie im letzten Abschnitt erläutert, generiert das JAXB-Framework XML-Zugriffsklassen aus einer Spezifikation der gelesenen XML-Daten. Ein verbreitetes Format zur Spezifikation von XML-Dokumenten ist XML-Schema [41].

Ein syntaxkonformes XML-Dokument wird als *wohlgeformt* bezeichnet. Oft müssen XML-Dokumente aber zusätzlich zur Wohlgeformtheit zusätzliche Restriktionen bezüglich Aufbau und Inhalt erfüllen, um entsprechend der verwendeten Software fehlerfrei bearbeitet werden zu können. Erfüllt ein Dokument diese Restriktionen, so wird es als *valide* (bezüglich einer bestimmten Spezifikation) bezeichnet. Um die Einhaltung der Regeln überprüfen zu können, existieren verschiedene Spezifikationssprachen für XML:

*Document Type Descriptions* (DTDs) sind die Spezifikationssprache des XML-Vorgängers SGML. Sie erlauben Angaben über die Struktur des XML-Dokuments und in eingeschränkter Form der erlaubten Datentypen. Da DTDs nicht in XML formuliert sind, sondern in einer eigenen Beschreibungssprache, wird für die Verarbeitung ein separater Parser benötigt, was die Wiederverwendbarkeit der in der Spezifikation enthaltenen Informationen erschwert.

Bei *XML-Schema* wird der gültige Aufbau eines XML-Dokuments selbst wieder in XML spezifiziert. Dabei gehen die Möglichkeiten von XML-Schema weit über die von DTDs hinaus. So können neben Regeln für den strukturellen Aufbau des Dokuments auch Forderungen an den Inhalt der Datenfelder gestellt werden, beispielsweise kann für ein Feld nur ein numerischer Werte erlaubt sein und keine Buchstaben. Es können aber auch Listen von erlaubten Werten oder reguläre Ausdrücke angegeben werden, denen die Felder gehorchen müssen. Ergänzend ist es möglich, jedem Eintrag über *Annotations* weitere applikationsspezifische Informationen hinzuzufügen. Ein

Beispiel für ein XML-Schema-Dokument ist Listing 2.4, in Listing 2.5 ist ein mögliches XML-Dokument abgedruckt, welches dieses Schema erfüllt.

**Listing 2.4:** Einfaches XML-Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

**Listing 2.5:** Einfache XML-Datei, spezifiziert durch Listing 2.4

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't␣forget␣me␣this␣weekend!</body>
</note>
```

## 2.4 LaTeX als Programmierumgebung

LaTeX wird (nicht nur) im akademischen Umfeld zum Erstellen von von Texten aller Art verwendet. Der folgende Abschnitt erklärt wichtige Grundlagen, beschreibt das Vorgehen beim Verfassen von Texten mit LaTeX und gibt einen Einblick in die Arbeitsweise eines TeX-Compilers.

### 2.4.1 Verfassen von Texten mit LaTeX

Die vorliegende Arbeit betrachtet LaTeX primär aus der Perspektive der Programmierung von LaTeX-Erweiterungen. LaTeX-Dokumente werden – ähnlich wie *Hyper Text Markup Language* (HTML) – in einer einfachen textuellen Beschreibungssprache verfasst, in welcher primär die logische Struktur des Textes, nicht aber das tatsächliche Aussehen beschrieben wird. Dieser Ansatz ist auch unter dem deutsch-englischen Begriff *logisches Markup* bekannt. Eine genaue Beschreibung der LaTeX-Sprache würde an dieser Stelle zu weit führen, stattdessen zeigt Listing 2.6 ein kommentiertes Beispiel eines kurzen, mit LaTeX verfassten Textes, welcher einige Elemente der Sprache demonstriert.

**Listing 2.6:** LaTeX-Beispieldokument

```
% Setzen des Dokumenttyps und globaler Einstellungen
\documentclass[a4paper,12pt]{article}
% Laden von Zusatzmodulen
\usepackage{german}
...
% Beginn des eigentlichen Textes
\begin{document}
...
% Kapitelüberschrift
\subsection{Verfassen von Texten mit \LaTeX}
\label{sec:ltx-intro}

Die vorliegende Arbeit betrachtet \LaTeX\ primär aus der Perspektive
der Programmierung von \LaTeX"=Erweiterungen. \LaTeX"=Dokumente werden
-- ähnlich wie \ac{HTML} -- in einer einfachen textuellen
Beschreibungssprache verfasst, in welcher primär die logische Struktur
des Textes, nicht aber das tatsächliche Aussehen beschrieben
wird. Dieser Ansatz ist auch unter dem deutsch"=englischen Begriff
\emph{logisches Markup} bekannt. Eine genaue Beschreibung der
\LaTeX"=Sprache würde an dieser Stelle zu weit führen, stattdessen
zeigt Listing~\ref{lst:ltx:example} ein kommentiertes Beispiel eines
kurzen, mit \LaTeX\ verfassten Textes, welcher einige Elemente
der Sprache demonstriert.
...
\end{document}
```

### 2.4.2 Begriffsdefinitionen

Während sich dem Anwender LaTeX als ein geschlossenes System darstellt, ist es für die Entwicklung von LaTeX-Modulen von enstscheidender Bedeutung, einen Überblick

über die verschiedenen Komponenten eines LaTeX-Systems zu gewinnen. Daher werden diese an dieser Stelle kurz erläutert.

*TeX* ist eine Satzsystem zum Setzen von Texten, vor allem solchen mit vielen mathematischen Formeln. TeX ist keine reine Markup-Sprache wie beispielsweise HTML, sondern stellt als Makroprozessor auch Kontrollstrukturen und Befehle zur Interaktion mit dem Betriebssystem bereit. Es gibt keine formale Spezifikation der Sprache „TeX“, stattdessen darf sich jedes Programm TeX nennen, welches die von Knuth entwickelte extensive Test-Suite `trip` [22] erfolgreich absolviert. Die Entwicklung von TeX wurde von Knuth eingestellt, lediglich Fehler, welche die Spezifikation von TeX verletzen, werden noch behoben.

*ε-TeX* [12] ist eine Erweiterung von TeX, welche einige der Größenbeschränkungen der originalen TeX-Spezifikation aufhebt und einige weitere Befehle bereitstellt. So wurde z. B. die Anzahl der erlaubten Register von 256 auf 32768 erhöht, was die Realisierung komplexerer Verfahren in TeX ermöglicht. Heutige TeX-Distributionen enthalten fast ausschließlich ε-TeX-kompatible TeX-Programme.

*LaTeX* [24] ist eine Sammlung von Makrobefehlen für TeX mit dem Ziel, strukturierte eingegebene Dokumente größtenteils automatisch ansprechend zu layouten. Die aktuelle Version von LaTeX ist LaTeX $2_\varepsilon$. Bei der Verwendung von LaTeX $2_\varepsilon$ empfehlen die Entwickler den Gebrauch eines ε-TeX-kompatiblen TeX-Programms, einige Pakete fordern dies bereits oder sind bei Nichtvorhandensein in ihrer Funktionalität eingeschränkt.

*pdfTeX* [29] ist ein erweiterter TeX-Prozessor, welcher als alternatives Ausgabeformat neben *Device Independant* (DVI) auch *Portable Document Format* (PDF) erzeugen kann. Der pdfTeX-Compiler ist seit einiger Zeit vollständig ε-TeX-kompatibel[1]. Zusätzlich zum alternativen Ausgabeformat unterstützt pdfTeX auch erweiterte Satzalgorithmen wie den optischen Randausgleich. Bei aktuellen TeX-Distributionen wird pdfTeX sowohl für die die PDF-Erzeugung als auch für die Erzeugung von DVI-Dateien verwendet. Im Gegensatz zu TeX wird pdfTeX aktiv weiterentwickelt, wobei die Abwärtskompatibilität zu TeX einen hohen Stellenwert einnimmt.

### 2.4.3 Verfassen von Texten mit LaTeX

Aus Benutzersicht besteht die Arbeit mit LaTeX aus drei wiederkehrenden Schritten:

1. Verfassen/Ändern des Quelltextes
2. Kompilieren zu DVI/PDF (evtl. mehrmals zur Auflösung von Referenzen)

[1] In den TeX-Distributionen TeXlive 2004 und TeXlive 2005 war ein ε-TeX-kompatibles pdfTeX unter dem Namen `pdfetex` aufrufbar, seit TeXlive 2007 ist ε-TeX standardmäßig aktiviert.

3. Anzeige des DVI/PDF-Dokuments in einem Betrachterprogramm

Für die Erzeugung von Literaturlisten und Indizes können auch noch weitere Programme zum Einstatz kommen, was an dieser Stelle aber vernachlässigt werden kann.

Für die Einbindung von Abbildungen in LaTeX-Dokumente stehen verschiedene Wege zur Verfügung: Zum einen können die Abbildungen innerhalb von LaTeX „programmiert“ werden, dies ist z. B. mit Bibliotheken wie *pstricks* [2] oder *TikZ* [38] möglich. Zum anderen können, je nach verwendetem Ausgabemodus des TeX-Prozessors, unterschiedliche externe Grafikformate eingebunden werden. Für Vektorgrafiken sind dies *Encapsulated PostScript* (EPS)-Dateien für den DVI-/PostScript-Betrieb und PDF-Dateien für den PDF-Modus.

### 2.4.4 Überblick über die Arbeitsweise des TeX-Prozessors

Viele der Besonderheiten, die sich bei der Programmierung mit LaTeX ergeben, erklären sich aus der Funktionsweise von TeX. Daher soll im folgenden ein kurzer Überblick über den Arbeitsablauf von TeX gegeben werden.

TeX verarbeitet die Eingabedatei sequenziell von vorne nach hinten. Die Datei wird gelesen, die Zeichen (gemäß ihres Category Codes) geparst und in Tokens gewandelt, welche dann von TeX bis auf die Ebene der Primitiven, also der vom TeX-Kernel bereitgestellten Befehle, expandiert werden.

Absätze werden komplett gelesen und ausgewertet, dann findet der Umbruch in Textzeilen statt. Diese Zeilen werden an die *Main Vertical List* (MVL) angehängt, eine virtuelle Seite, die beliebig lang werden kann. Ist auf dieser Liste ausreichend Material für eine neue Seite gesammelt, so wird der Algorithmus zum Füllen der Seiten (die *Output-Routine*) aufgerufen, diese füllt aus dem Material der MVL sowie weiteren Elementen wie Fußnoten (die auch noch von vorherigen Seiten stammen können) und Gleitobjekten eine Seite, fügt diese der Ausgabedatei hinzu und entfernt das gesetzte Material von der MVL.

Einmal gesetztes Material ist damit nicht mehr veränderbar, Referenzen im Text wie die Einträge des Inhaltsverzeichnisses, die erst später im Text ermittelt werden, werden daher nicht durch Modifikation bereits gesetzten Materials erzeugt, sondern durch das Lesen und Schreiben in temporäre Dateien, welche Zustandsinformationen von einem LaTeX-Lauf zum nächsten transportieren. Die Funktionsweise von LaTeX ist daher vergleichbar mit der eines *Multipass*-Übersetzers (s. Abb. 2.8).

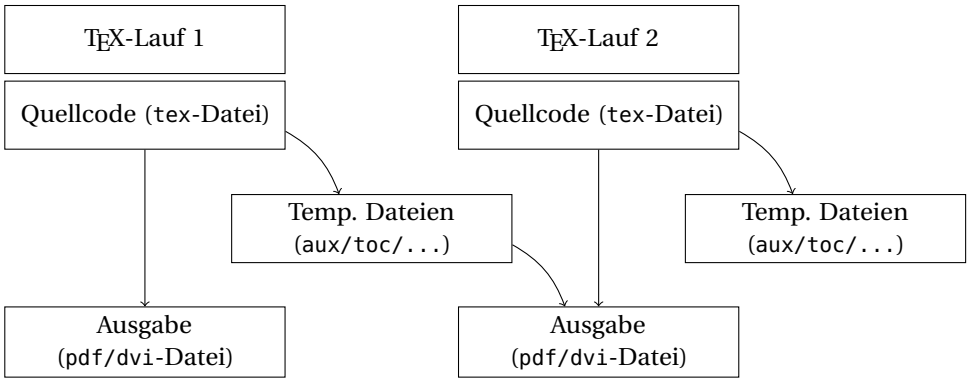



**Abbildung 2.8:** Übermittlung von Informationen zwischen aufeinanderfolgenden TEX-Durchläufen

# 3 Entwurf der LaTeX-Schnittstelle für KIEL-Statecharts

## 3.1 Analyse der bisherigen Nutzung

Bevor ein LaTeX-Interface für KIEL entwickelt werden kann, ist eine Betrachtung der bisherigen Verwendung von KIEL-Statecharts in LaTeX notwendig. Dabei wird eine Reihe von Schritten identifiziert, die für jedes Statechart durchgeführt werden, welches in ein LaTeX-Dokument eingefügt wird (s. Abb. 3.1).

*Arbeitsschritt 1:* In der KIEL-*Graphical User Interface* (GUI) wird das Statechart geladen oder erstellt.

Das Statechart muss dafür in einem von KIEL unterstützten Format vorliegen. Falls ein neues Statechart erstellt werden soll, so kann dies grafisch in der KIEL-GUI oder in einer der textuellen Statechart-Beschreibungssprachen (s. Abschnitt 2.1) erzeugt werden. Das Verhalten von KIEL ist durch die Programmeinstellungen parametrisierbar. Sollen für die zu erstellende Abbildung andere als die aktuell gültigen Programmeinstellungen gelten, so sind diese einzustellen und nach Erzeugung der Abbildung wieder rückgängig zu machen. Dies muss in den Konfigurationsdateien des Benutzers erfolgen (s. Abb. 3.2) und erfordert für viele Einstellungen einen Neustart von KIEL, da die meisten Module die Einstellungen nur beim Programmstart einlesen. Da die Änderungen von Hand in den Konfigurationsdateien vorgenommen werden müssen, kann es dabei zu ungültigen Eintragungen kommen. Der Anwender wird bei der manuellen Änderung von Eintragungen durch Kommentare in der Property-Datei unterstützt.

*Arbeitsschritt 2:* Bearbeitung/Veränderung von Statecharts in KIEL.

Das Statechart kann vom Anwender mit den in Abschnitt 2.2.1 beschriebenen Verfahren bearbeitet werden.

*Arbeitsschritt 3:* Erzeugung einer Statechart-Ansicht.

Zur Erzeugung einer Statechart-Abbildung muss eine Statechart-Ansicht erzeugt werden (s. Abschnitt 2.2.3). Eine Standardansicht wird bereits beim Laden des Statecharts erzeugt. Die Anordnung von Statechart-Elemente kann durch Aufruf eines Autolayouters verändert werden. Ansichten können auch durch Konfigurationen von aktiven

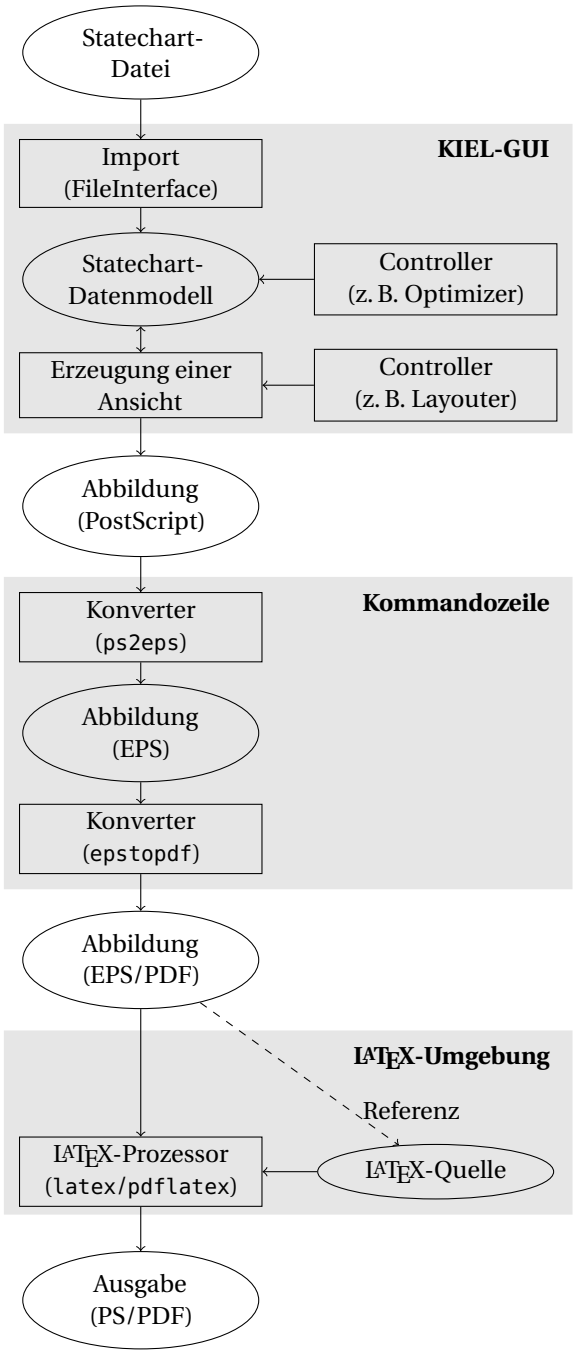



**Abbildung 3.1:** Arbeitsablauf beim Einbinden von KIEL-Statecharts in LaTeX

```
# $Id: cha-konzept.tex,v 1.14 2007/05/13 09:00:01 khe Exp $
# Just change if you know what you are doing

# Browser and its components properties
Browser.Title=Kiel Statechart Browser 2.0
Browser.ShowInterface=false
Browser.ShowTooltip=true
Browser.AlwaysShowPrios=false
Browser.AnimateConfigInMicroSteps=false
Browser.AlwaysStepThrough=true
Browser.Animation=true
Browser.LogLevel=2
Browser.LogCompare=">="
Browser.LayouterBenchFile=LayouterBench_
Browser.DrawingBenchFile=DrawingBench_
Browser.SimulatorBenchFile=SimBench.txt
Browser.doLayouterBench=false
Browser.doDrawingBench=false
Browser.doSimulatorBench=false
Browser.doSimulatorOutput=true
BrowserCanvas.StringLabelColor=#ff0000
BrowserTree.MarkNodeColor=#ffff00
BrowserTree.MarkNodeBrightColor=#ffffbb
...
```

**Abbildung 3.2:** Aufbau von KIELs Konfigurationsdateien

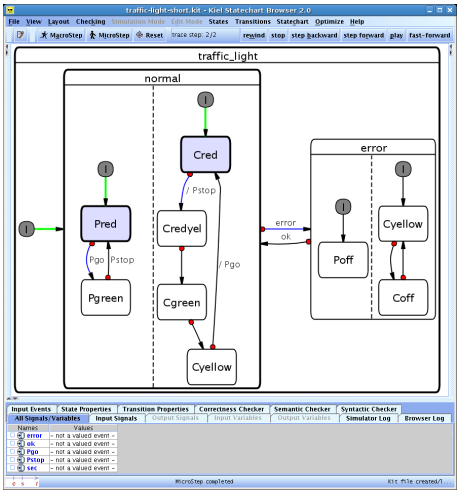
Zuständen generiert werden, wie sie beim Simulieren von Statecharts erzeugt werden (Abb. 3.3 und 3.4). Falls die Funktion „Semantic Zoom“ (s. Abschnitt 2.2.1) aktiv ist, werden Bereiche des Statecharts ohne aktive Zustände ausgeblendet (Abb. 3.5).

*Arbeitsschritt 4:* Export der Statechart-Ansicht in eine Grafikdatei.

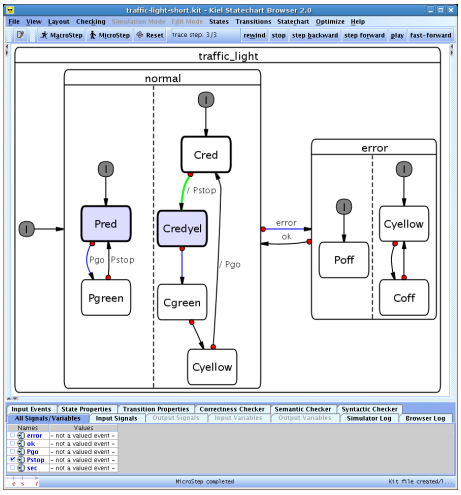
Der Speicherpfad und der Dateiname sind so zu wählen, dass die Grafik passend zum LaTeX-Dokument abgelegt wird. Damit ist die Arbeit in der KIEL-GUI abgeschlossen, die übrigen Schritte finden an der Kommandozeile und im LaTeX-Editor statt.

*Arbeitsschritt 5:* Nachbearbeitung der Grafikdatei an der Kommandozeile.

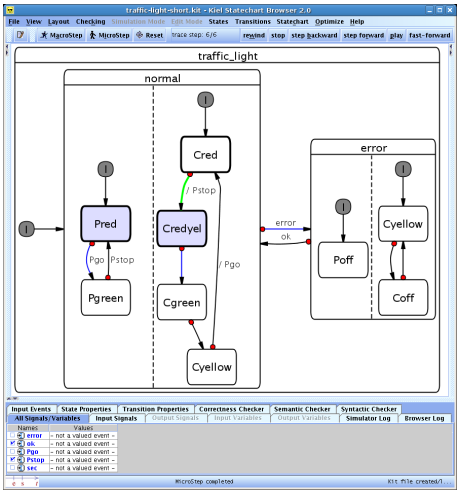
Bei den von KIEL generierten Grafiken handelt es sich um einseitige PostScript-Dateien, bei denen die Seitengröße an die Größe des Statecharts angenähert wurde. Mit einem Programm wie `ps2eps` können die dabei verbleibenden weißen Ränder entfernt werden und es wird sichergestellt, dass die Dateien der EPS-Spezifikation [3] entsprechen. Für die Verarbeitung mit pdfTeX (im PDF-Modus) muss die Grafik im



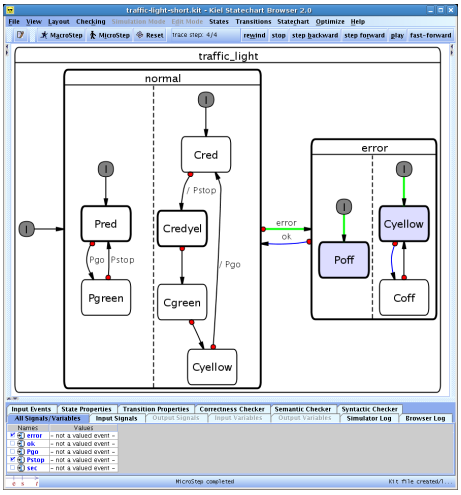

**(a)** Beginn der Simulation




**(b)** Konfiguration nach einem Makrostep




**(c)** Konfiguration nach zwei Makrosteps




**(d)** Konfiguration nach einem Makrostep bei gesetztem Signal *Error*

**Abbildung 3.3:** Statechart-Simulation in KIEL (statische Ansicht)

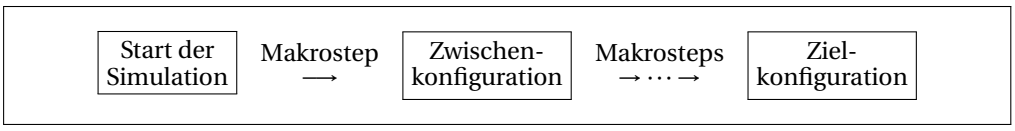
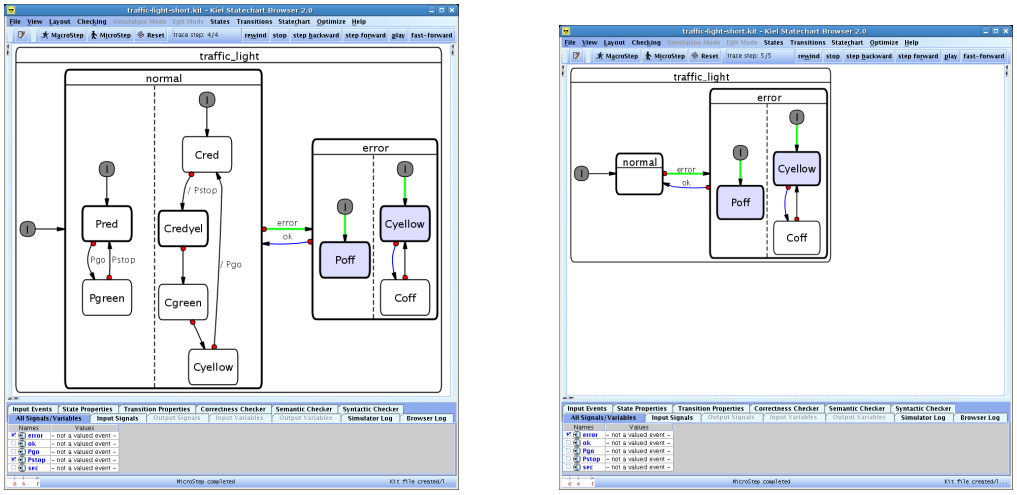



**Abbildung 3.4:** Erzeugen von Statechart-Konfigurationen durch Simulation




**(a)** Statische Ansicht **(b)** Dynamische Ansicht

**Abbildung 3.5:** Statechart-Simulation mit und ohne dynamischer Anzeige

PDF-Format vorliegen. Die PDF-Datei wird mittels eines Konverters wie `epstopdf` aus der EPS-Datei generiert.

*Arbeitsschritt 6:* Grafikdatei in LaTeX einbinden.

Dafür müssen die erzeugten EPS- und ggf. PDF-Dateien in einem für LaTeX erreichbaren Verzeichnis liegen, üblicherweise im Verzeichnis des Textes oder einem Unterverzeichnis. Der Name der Datei muss bekannt sein. Die Anzeigeparameter (z. B. Größe oder Rotation) müssen eingestellt werden. Anschließend wird das Dokument mit LaTeX kompiliert. Wird bei der Kontrolle der LaTeX-Ausgabe festgestellt, dass an dem Statechart Änderungen notwendig sind, so müssen die obigen Schritte, beginnend in der KIEL-GUI, wiederholt werden.

Zur Einbindung von Statecharts in LaTeX-Dokumente muss also eine Vielzahl von Schritten durchgeführt werden. Das Ziel der KIEL-LaTeX-Schnittstelle ist es, die Anzahl dieser Schritte zu minimieren.

## 3.2 Resultierende Anforderungen

Die im letzten Abschnitt gesammelten Daten stellen folgende Forderungen an die KIEL-LaTeX-Schnittstelle:

- Abbildungen von Statecharts sollen erzeugbar sein, ohne dass die grafische Arbeitsumgebung von KIEL aufgerufen werden muss.

- Statecharts sollen aus allen Formaten erzeugbar sein, die KIEL interpretieren kann. Die Statecharts sollen sowohl als Teil des LaTeX-Dokuments, als auch aus extern vorliegenden Dateien erzeugt werden können.
- Alle Parameter von KIEL, die für das Aussehen der Statecharts von Bedeutung sind, sollen für die Generierung einzelner Statecharts anpassbar sein, ohne dass die Konfigurationsdateien geändert werden müssen.
- Die Parameter, die auf das Aussehen eines Statecharts Einfluss nehmen, sollen gesammelt gespeichert werden können, damit bei Grafiken, an denen nur kleine Änderungen nötig sind, nicht alle Einstellungen erneut vorgenommen werden müssen.
- Die generierten Abbildungen sollen automatisch in das benötigte Zielformat konvertiert werden.
- Statecharts sollen in LaTeX wahlweise als Abbildung, als Quellcode oder in einer gemischten Ansicht angezeigt werden.

Neben den an die LaTeX-Anbindung gerichteten Anforderungen werden ergänzend folgende Forderungen formuliert:

- Fehlerhafte Eintragungen in der Benutzerkonfiguration sollen beim Programmstart erkannt werden
- Programmierung eines Konfigurationsdialogs für KIELs grafisches Interface
- Vermeidung der Generierung von Statechart-Abbildungen, deren Daten sich seit dem letzten Aufruf nicht geändert haben.

# 4 Implementierung

Nachdem im letzten Kapitel die Anforderungen an eine LaTeX-Schnittstelle für KIEL-Statecharts zusammengestellt wurden, wird jetzt die Umsetzung der im Rahmen der Arbeit entstandenen Programme beschrieben. Begonnen wird mit dem LaTeX-Modul KielTeX, welches es dem LaTeX-Anwender erlaubt, Statecharts in sein Dokument einzubinden. KielTeX verwendet zur Erzeugung von Statecharts das Programm KielCmd. Die dafür notwendigen Ergänzungen an KielCmd werden in Abschnitt 4.2 vorgestellt. Die Änderungen an dem Konfigurationssystem von KIEL werden in Abschnitt 4.3 beschrieben.

## 4.1 Das LaTeX-Modul KielTeX

### 4.1.1 Funktionaler Überblick

Mit KielTeX können von KIEL erzeugte Statechart-Abbildungen in LaTeX-Dokumente eingebunden werden. Die Erzeugung der Abbildungen durch den LaTeX-Compiler verläuft in mehreren Schritten (Abb. 4.1 und 4.2):

1. Ist ein Statechart in das LaTeX-Dokument eingebettet, so wird das Statechart während des LaTeX-Laufs in eine temporäre Datei extrahiert. Den Namen der temporären Datei bildet KielTeX aus dem Namen des Dokuments und einem Zähler, der für jedes im Dokument enthaltene Statechart um eins erhöht wird. Da die Dateiendung der temporären Statechart-Datei KielCmd darüber informiert, in welcher Sprache das Statechart vorliegt, muss die Endung durch den Benutzer gesetzt werden, entweder mit der `language`-Option für Sprachen, die KielTeX bekannt sind, oder mit der `filetype`-Option für beliebige Endungen.

   **Beispiel:** Ein Dokument `diplomarbeit.tex` enthält drei Statecharts. Das erste ist in *Esterel* geschrieben und liegt eingebettet im Dokument vor. Das zweite ist eine Referenz auf eine Statechart-Datei `statecharts/abro.scg` (im *Esterel Studio*-Format) und das dritte ist wiederum in das Dokument eingebettet und im KIT-Format. LaTeX erzeugt bei der Verarbeitung des Dokuments die temporären Dateien `diplomarbeit-fig1.strl` und `diplomarbeit-fig3.kit`. Für das zweite Statchart wird keine temporäre Datei angelegt, da für die Verarbeitung die existierende, im Text referenzierte Statechart-Datei verwendet wird.

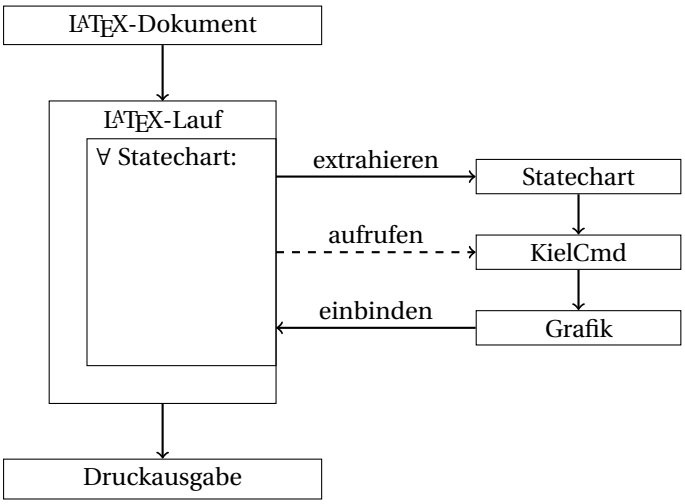



**Abbildung 4.1:** Ablauf der Statechart-Erzeugung durch KielTEX (Überblick)

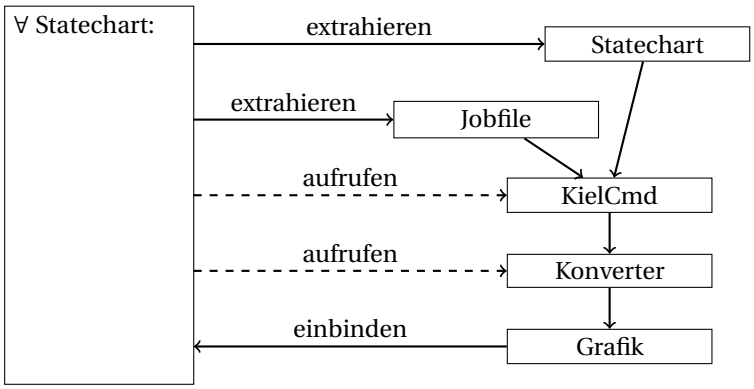



**Abbildung 4.2:** Erzeugung eines Statecharts durch KielTEX

2. Die aktuell gültigen Optionen jedes Statecharts werden in eine weitere temporäre Datei gespeichert, dem *Jobfile*.

   Die aktuell gültigen Optionen können zum einen global gesetzt worden sein, zum anderen auch gezielt für das aktuelle Statechart. Für das Statechart explizit gesetzte Optionen genießen dabei Vorrang. Das *Jobfile* enthält neben den Optionen auch den Dateinamen des Statecharts (temporär erzeugt oder referenziert) und den Namen der Ausgabedatei.

   **Beispiel:** Beim Anwenden von LaTeX auf das im letzten Beispiel beschriebene Dokument `diplomarbeit.tex` werden die in Tabelle 4.1 aufgeführten *Jobfiles* erzeugt.

3. Wurde LaTeX mit der Option zum Ausführen externer Programme gestartet,

**Tabelle 4.1:** Aus dem Beispieldokument erzeugte *Jobfiles*

| | Enthält Referenzen zu | |
|---|---|---|
| **Erzeugtes Jobfile** | **Statechart-Datei** | **Ausgabedatei** |
| `diplomarbeit-fig1.job` | `diplomarbeit-fig1.strl` | `diplomarbeit-fig1.ps` |
| `diplomarbeit-fig2.job` | `statecharts/abro.scg` | `diplomarbeit-fig2.ps` |
| `diplomarbeit-fig3.job` | `diplomarbeit-fig3.kit` | `diplomarbeit-fig3.ps` |

so ruft KielTeX nun das Programm `kielcmd` mit dem *Jobfile* als Parameter auf. Während `kielcmd` läuft, wartet KielTeX, anschließend wird die Bearbeitung des Dokuments fortgesetzt.

Sofern dies nicht durch eine Option unterbunden wurde, werden anschließend die Programme `ps2eps`[1] zur Optimierung der PostScript-Ausgabe und `epstopdf`[2] zur Konvertierung nach PDF aufgerufen.

**Beispiel:** Bezogen auf obiges Beispiel würde KielCmd während der Verarbeitung von `diplomarbeit.tex` nacheinander folgende Programme aufrufen:

- `kielcmd -j=diplomarbeit-fig1.job` (erzeugt `diplomarbeit-fig1.ps`)
- `ps2eps -f diplomarbeit-fig1.ps` (erzeugt `diplomarbeit-fig1.eps`)
- `epstopdf diplomarbeit-fig1.eps` (erzeugt `diplomarbeit-fig1.pdf`)
- `kielcmd -j=diplomarbeit-fig2.job` (erzeugt `diplomarbeit-fig2.ps`)
- ...

4. Im letzten Schritt wird die Ausgabedatei als Grafik in den Text eingefügt, unabhängig davon, ob die vorangegangenen Schritte erfolgreich waren. Dies erlaubt es, Grafiken auch außerhalb des LaTeX-Laufs zu generieren, sie werden dann beim nächsten Lauf eingebunden.

   **Beispiel:** Die Einbindung der erzeugten Abbildungen in `diplomarbeit.tex` entspricht folgenden Befehlen:

```
\includegraphics[⟨Optionen⟩]{diplomarbeit-fig1}
...
\includegraphics[⟨Optionen⟩]{diplomarbeit-fig2}
...
\includegraphics[⟨Optionen⟩]{diplomarbeit-fig3}
```

   Falls einer der alternativen Anzeigemodi aktiv ist (nur Quellcode anzeigen oder Quellcode neben Statechart anzeigen), so wird die Statechart-Datei mit dem *listings*-Paket als Listing angezeigt. Wurde mit der *language*-Option eine Sprache

[1] Getestet mit Version: 1.61
[2] Getestet mit EPSTOPDF 2.9.5gw, 2006/01/29

für das aktive Statechart eingestellt und ist für diese Sprache eine Syntaxdefinition vorhanden, so werden die Schlüsselwörter im Listing entsprechend hervorgehoben.

**Beispiel:** Angenommen, für das erste und zweite Statechart in `diplomarbeit.tex` wäre die Option `show=source` gesetzt. In diesem Fall würde für an Stelle der Abbildungen der Quellcode eingefügt, was folgenden Befehlen entspräche:

```
\lstinputlisting{diplomarbeit-fig1.strl}
...
\lstinputlisting{statecharts/abro.scg}
...
\includegraphics[⟨Optionen⟩]{diplomarbeit-fig3}
```

### 4.1.2 Einbindung von Statecharts

KielTEX stellt zwei Verfahren zur Einbindung von Statecharts in LATEX-Dokumente bereit. Zum einen ist es möglich, den Quellcode des Statecharts in einer `KIELstatechart`-Umgebung direkt in den Text einzubetten. Zum anderen können Statechart-Dateien über ihren Dateinamen referenziert werden.

**Beispiel:** Die im letzten Abschnitt erwähnte Datei `diplomarbeit.tex` könnte folgende Befehle enthalten:

```
...
%% Erstes Statechart
\begin{KIELstatechart}[language=Esterel]
  ...
  ⟨Esterel-Code⟩
  ...
\end{KIELstatechart}
...
%% Zweites Statechart
\includeKIELstatechart{statecharts/abro.scg}
...
%% Drittes Statechart
\begin{KIELstatechart}[language=kit]
  ...
  ⟨KIT-Code⟩
  ...
\end{KIELstatechart}
```

Wie in Abschnitt 4.1.1 erläutert, schreibt KielTEX die in den Text eingebetteten Statecharts in temporäre Dateien, um sie anschließend weiter zu verarbeiten. Dafür ist es wichtig, dass der Text des Statecharts von LATEX nicht interpretiert wird, sondern unverändert in die Datei geschrieben wird. Der LATEX-Kernel stellt hierfür keine

Funkionen bereit. Daher wird in KielTeX zu diesem Zweck das `verbatim`-Paket [34] verwendet. Dieses Paket stellt Methoden bereit, mit denen der Inhalt einer Umgebung unverändert in eine Datei geschrieben werden kann. Das Ende des zu extrahierenden Bereichs wird durch eine Zeile gekennzeichnet, die nur die Zeichenkette

```
\end{KIELstatechart}
```

enthalten darf.

### 4.1.3 Verarbeitung von Befehlsparametern

Es gibt drei Gruppen von Einstellungen, die für KielTeX relevant sind:

1. Einstellungen, welche das Verhalten von KielTeX steuern
2. Einstellungen, welche die Generierung der Statechart-Abbildung durch KIEL beeinflussen. Neben der Wahl des zu verwendenden automatischen Layouters sind hierzu auch alle Konfigurationsoptionen der KIEL-Module zu zählen.
3. Einstellungen, welche die Darstellung des Statecharts auf LaTeX-Ebene steuern, beispielweise in welcher Größe das Statechart angezeigt werden soll.

Das KielTeX-Handbuch (siehe Anhang) enthält eine Liste aller KielTeX-Optionen. Die zum Druckzeitpunkt aktuelle Liste der KIEL-Programmeinstellungen ist in Anhang A aufgeführt.

Eine Anforderung an KielTeX ist, dass die Optionen für jedes eingebundene Statechart einzeln eingestellt werden können. Dafür müssen die Optionen als Parameter an Befehle zur Statechart-Einbindung übergeben werden können. Das Standardverfahren zur Übergabe von Parametern an LaTeX-Befehle ist die Verwendung von in einzelne Klammern für jede Option. Als Beispiel sei der `\frac{⟨Zaehler⟩}{⟨Nenner⟩}`-Befehl zur Angabe von Brüchen genannt. Bei dieser Form der Parameterübergabe ergeben sich zwei Probleme:

- Es muss eindeutig festgelegt sein, wie viele Optionen der Befehl erwartet, und
- es müssen immer alle Optionen angegeben werden.

Da KIEL mehr als 200 Programmeinstellungen hat, erwies sich dieser Ansatz für KielTeX als nicht praktikabel. Stattdessen wird ein erweitertes Optionsformat verwendet, welches das `xkeyval`-Paket [4] für LaTeX bereitstellt. Das Paket ermöglicht es, Parameter in Form von *KV-Listen* (*Key-Value*-Liste) zu übergeben. Eine *KV-Liste* ist eine Folge von null oder mehr `Schlüssel=Wert`-Paaren, die durch Kommata voneinander getrennt werden. Würde man beispielsweise einen `\kvfrac`-Befehl mit Parametern in diesem Format implementieren, so sähen Aufrufe folgendermaßen aus:

```
\kvfrac{zaehler=10,nenner=3}
\kvfrac{zaehler=10,nenner=3,BruchstrichDicke=1pt}
```

Wie zu sehen ist, erlaubt dieses Format eine beliebige Anzahl von Parametern, die auch nicht immer alle angegeben werden müssen.

Jede verfügbare Option muss bei dem `xkeyval`-Paket mit einer Funktion registriert sein, die aufgerufen wird, wenn die Option gesetzt wird. So kann eine *KV-Liste* daraufhin überprüft werden, ob nur gültige Schlüssel angegeben wurden. Die konsequente Verwendung dieses Schemas würde allerdings bedeuten, dass auch die mehr als 200 Programmeinstellungen von KIEL als KielTEX-Optionen registriert und bei Änderungen an KIEL mit synchronisiert werden müssten. Um das zu vermeiden, werden alle nicht bekannten Schlüssel als KIEL-Einstellungen behandelt und an KielCmd weitergereicht. Diese Vorgehensweise führt dazu, dass fehlerhafte Schlüssel (z. B. bei Schreibfehlern) nicht mehr auf LATEX-Ebene erkannt werden. Da KielCmd bei unbekannten Parametern eine Warnung ausgibt, scheint diese Einschränkung aber vertretbar.

Das `xkeyval`-Paket selbst sieht allerdings keine Möglichkeit vor, auf unbekannte Schlüssel anders als mit einer Fehlermeldung zu reagieren. Daher wird in KielTEX zusätzlich das Paket `kvsetkeys` [27] verwendet. Es erweitert `xkeyval` um die Möglichkeit, bei unbekannten Schlüsseln eine durch den Programmierer definierte Funktion aufzurufen, die diese Schlüssel-Wert-Paare gesondert behandelt.

Zur Adressierung von KIELs Programmeinstellungen wird der Name des Moduls, gefolgt von einem Doppelpunkt und dem Namen der Option als Schlüssel angegeben:

```
\KIELsetup{layouter:graphviz.dpi=72}
           -------- ------------ --
            Modul    Schluessel  Wert
```

Damit wird die Option `graphviz.dpi` des KIEL-Moduls `layouter` auf den Wert 72 gesetzt.

Die KIEL-Programmeinstellungen werden in KielTEX intern in einer eigenen Liste gespeichert. Wird in LATEX eine KIEL-Programmeinstellung gesetzt, so wird überprüft, ob diese bereits in der Liste der zu setzenden Einstellungen enthalten ist. Ist dies der Fall, so wird nur der Wert ersetzt, andernfalls wird die Option der Liste hinzugefügt. Beim Schreiben des *Jobfiles* wird die aktuell gültige Liste der Einstellungen formatiert und ausgegeben (Tab. 4.2).

**Tabelle 4.2:** Setzen und Überschreiben von KIEL-Einstellungen in KielTeX

| Befehl | Gültige Einstellungen |
|---|---|
| `\KIELsetup{kielopts={layouter.graphviz.dpi=100}}` | `layouter.graphviz.dpi=100` |
| `\KIELsetup{kielopts={optimizer.DebugMode=false}}` | `layouter.graphviz.dpi=100`<br>`optimizer.DebugMode=false` |
| `\KIELsetup{kielopts={layouter.graphviz.dpi=72}}` | `layouter.graphviz.dpi=72`<br>`optimizer.DebugMode=false` |

### 4.1.4 Erzeugung von *Jobfiles*

In *Jobfiles* werden von KielTeX alle Parameter einer Statechart-Abbildung hinterlegt (s. Abschnitt 4.1.1). Sie dienen der Kommunikation mit KielCmd. Beim Verarbeiten eines Dokumentes mit LaTeX wird von KielTeX für jedes enthaltene Statechart ein *Jobfile* angelegt. Das *Jobfile* enthält alle Optionen, die für das jeweilige Statechart gültig sind. Sie sind so aufgebaut, dass in jeder Zeile eine Kommandozeilenoption für KielCmd steht. Gegeben sei beispielsweise der KielCmd-Aufruf

```
kielcmd -L=HVLayouter -option=layouter.graphviz.dpi=100 -o=abro.ps abro.scg
```

Ein gleichwertiges *Jobfile* hätte dann folgenden Inhalt:

```
-L=HVLayouter
-option=layouter.graphviz.dpi=100
-o=abro.ps
abro.scg
```

### 4.1.5 Aufrufen von `kielcmd`, `ps2eps` und `epstopdf`

Zu den Funktionen, die TeX (und damit auch LaTeX) zur Interaktion mit dem Betriebssystem bereitstellt, gehört das Aufrufen externer Programme mit dem `\write18`-Befehl. Trifft TeX beim Bearbeiten einer Datei auf einen solchen Befehl, so wird das Argument zur Ausführung an das Betriebssystem übergeben. TeX pausiert die Bearbeitung des Dokuments, bis der ausgeführte Befehl terminiert. Eine Rückmeldung über das Ergebnis des Befehls an TeX ist nicht vorgesehen. Auch ist es nicht möglich, die Menge der ausführbaren Befehle einzuschränken.

Da die Möglichkeit, beim Übersetzen von LaTeX-Dokumenten beliebige Befehle auszuführen, ein gewisses Sicherheitsrisiko darstellt, ist der `\write18`-Befehl üblicherweise deaktiviert und muss über einen Kommandozeilenparameter (`-shell-escape`) aktiviert werden. Hinzu kommt, dass in älteren TeX-Versionen keinen Möglichkeit

vorgesehen ist, während eines TEX-Laufs zu erfahren, ob die Ausführung externer Programme erlaubt ist. Mit pdfTEX 1.30 wurde eine nur lesbare TEX-Variable eingeführt, die über die Möglichkeit zur Ausführung externer Programme informiert. Aus diesem Grund setzt KielTEX pdfTEX 1.30 oder neuer voraus.

KielTEX verwendet den `\write18`-Befehl zum Aufruf von `kielcmd` und führt danach auch die Konverter `ps2eps` und `epstopdf` aus (s. Abschnitt 4.1.1).

### 4.1.6 Darstellung der Statecharts

Statecharts können von KielTEX auf drei Arten angezeigt werden, als Abbildung, im Quelltext oder in einer kombinierten Ansicht. Welcher dieser Modi aktiv ist, wird durch die `show`-Option eingestellt.

#### Darstellung als Abbildung (`show=chart`, Vorgabewert)

In diesem Modus wird die von KielCmd erzeugte Abbildung wie eine normale Grafikdatei in das Dokument eingefügt. Zum Einbinden der Grafik verwendet KielTEX den `\includegraphics`-Befehl des `graphicx`-Pakets [7]. Eigenschaften der Abbildung wie Höhe, Breite oder Rotation können über Optionen gesetzt werden, die KielTEX an `\includegraphics` weiterreicht.

#### Darstellung als Quellcode (`show=source`)

Für die Quelltextdarstellung von Statecharts verwendet KielTEX das `listings`-Paket [20]. Mit dem `listings`-Paket kann Programmquelltext dargestellt werden, für viele Programmiersprachen ist eine Hervorhebung von Schlüsselworten und anderen syntaktischen Bestandteilen der Programmiersprache verfügbar. Für die Sprachen *Esterel* und KIT ist eine Syntaxhervorhebung verfügbar und wird von KielTEX aktiviert, wenn die `language`-Option entsprechend gesetzt ist.

Zur Darstellung des Statechart-Quelltextes wird der `\lstinputlisting`-Befehl des `listings`-Pakets verwendet, mit dem extern vorliegende Dateien angezeigt werden können. Bei eingebetteten Statecharts wird die temporäre Statechart-Datei zur Quellcode-Darstellung verwendet, bei mit `\includeKIELstatechart` referenzierten Statechart-Dateien wird diese angezeigt.

#### Abbildung und Quelltext nebeneinander (`show=split`)

Dieser Modus kombiniert die Anzeige von Statecharts und Quelltext miteinander, indem Statechart und Quelltext nebeneinander angezeigt werden. Für die Positionierung wird die LATEX-Umgebung `minipage` verwendet, die es ermöglicht, nicht

sichtbare Kästen mit definierter Breite in das LaTeX-Dokument einzufügen, die nebeneinandergestellt werden. Zur Anzeige des Statecharts und des zugehörigen Quelltextes werden die in den beiden vorigen Abschnitten beschriebenen Verfahren verwendet.

## 4.2 Erweiterung von KIELs Kommandozeilenschnittstelle KielCmd

Für KielTeX sind eine Reihe von Anforderungen definiert (s. Abschnitt 3.2). Ein Teil dieser Anforderungen betrifft KIELs Kommandozeilenschnittstelle KielCmd:

- das temporäre Verändern aller KIEL-Einstellungen ohne Auswirkungen auf die Konfigurationsdateien,
- die Übergabe von Optionen in *Jobfiles*, und
- die Auswahl der Zustände, die in dem zu verarbeitenden Statechart als aktiv markiert sein sollen.

Die Umsetzung dieser Anforderungen ist Gegenstand dieses Abschnitts.

### 4.2.1 Setzen von KIEL-Programmeinstellungen

Um das temporäre Setzen beliebiger KIEL-Programmeinstellungen zu erlauben, wurde eine weitere Kommandozeilenoption zu KielCmd hinzugefügt:

```
-option=⟨module⟩.⟨property⟩=⟨value⟩ : Set ⟨property⟩ for ⟨module⟩ to ⟨value⟩ temporarily
```

Die Syntax entspricht der in Abschnitt 4.1.3 beschriebenen. Der Parameter `-option` kann mehrmals angegeben werden, jedes Vorkommen setzt eine Option. Implementiert ist das Setzen von Optionen über den generischen Konfigurations-Mechanismus des *Preferences*-Moduls (s. Abschnitt 4.3.3).

### 4.2.2 Verarbeitung von Job-Files

KielTeX setzt beim Aufrufen von KielCmd keine Kommandozeilenoptionen, sondern schreibt diese in ein dem jeweiligen Statechart zugeordnetes *Jobfile* (s. Abschnitt 4.1.4). Jede Option wird dabei genau so, wie sie an der Kommandozeile angegeben würde, in eine eigene Zeile geschrieben. So wird verhindert, dass Leer- oder Sonderzeichen maskiert weden müssen.

Um die Jobfiles auszulesen, wurde KielCmd um eine Option `-j=⟨jobfile⟩` ergänzt. KielCmd speichert die an der Kommandozeile übergebenen Parameter in einer Liste und arbeitet sie sequenziell ab. Trifft es beim Bearbeiten der Parameterliste auf einen `-j`-Eintrag, so wird an Stelle des Eintrags der Inhalt der referenzierten Datei in die Liste eingefügt. Zum einen ist es dadurch möglich, Parametersätze für bestimmte Standardfälle in einem Jobfile abzulegen und das Statechart an der Kommandozeile anzugeben (s. Listing 4.1). Zum anderen können auch alle für einen Lauf verwendeten Optionen in ein Jobfile geschrieben werden, so dass der Lauf mit Jobfile und Statechart-Datei vollständig reproduzierbar ist (s. Listing 4.2).

**Listing 4.1:** Teile der Parameter im Jobfile

```
khe@january:~$ cat optimizedhv.job
-Opt
-L=HVLayouter
khe@january:~$ kielcmd -j=optimizedhv.job abro.kit
khe@january:~$ kielcmd -j=optimizedhv.job abcro.kit
```

**Listing 4.2:** Alle Parameter im Jobfile

```
khe@january:~$ cat abro1.job
-Opt
-L=HVLayouter
-o=abro-hv.eps
abro.kit
khe@january:~$ kielcmd -j=abro1.job
```

KielCmd kombiniert beim Start die Kommandozeilenoptionen und den Inhalt der angegebenen *Jobfiles* zu einer Liste, die alle gesetzten Optionen enthält.

### 4.2.3 Adressierung von Statechart-Konfigurationen

KielCmd erlaubt es, für das zu layoutende Statechart eine Statechart-Konfiguration anzugeben (s. Abschnitt 2.1). Dafür können an der Kommandozeile die bei der Simulation aktiven Zustände der Konfiguration mittels der Option `-config=⟨Zustand 1⟩⟨Zustand2⟩...` angegeben werden. An der Kommandozeile muss verhindert werden, dass das Leerzeichen als Parameter-Trennzeichen interpretiert wird, üblicherweise durch Anführungsstriche oder *escaping*:

```
khe@january:~$ kielcmd -config="A B" abro.kit
...
khe@january:~$ kielcmd -config=A\ B abro.kit
```

In Jobfiles muss das Leerzeichen nicht gesondert behandelt werden, hier trennt der Zeilenumbruch die einzelnen Parameter.

Die Adressierung einzelner Zustände über ihre Namen war in KIEL ursprünglich nicht vorgesehen. Daher erzeugt KielCmd nach dem Laden des Statecharts eine Zuordnungstabelle zwischen Zustandsnamen und Zustandsobjekten, indem über alle Zustände des Statecharts iteriert wird und zu jedem Zustandsnamen die Referenz auf das zugehörige Objekt abgespeichert wird. Da Zustandsnamen nicht eindeutig sein müssen, wird bei Zuständen mit gleichem Namen eine Warnung ausgegeben und der erste gefundene Zustand dieses Namens verwendet.

### 4.2.4 Markierung ausgewählter Zustände

Das Browser-Modul von KIEL erlaubt es, Statecharts tastaturbasiert zu bearbeiten (s. Abschnitt 2.2.1). Im Rahmen dieser Bearbeitung ist es möglich, Zustände zu markieren, um beispielsweise ausgehend von diesen Zuständen neue Zustände oder Transitionen einzufügen. Um diese Funktionen dokumentieren zu können, ist es möglich, diese Markierungen auch über KielCmd einzufügen. Dafür ist es möglich, maximal zwei Zustände über ihren Namen anzugeben, die dann hervorgehoben werden. Sind diese Zustände über eine Transition verbunden, so ist es auch möglich, diese Transition zu markieren. Existieren mehrere Transitionen mit unterschiedlichen Prioritäten zwischen den Zuständen, so kann über die Angabe der Priorität gezielt eine Transition ausgewählt werden.

### 4.2.5 Zwischenspeicherung von erzeugten Statecharts

Während der Entwicklung von KielTEX stellte sich heraus, dass es wünschenswert wäre, KielCmd um einen Caching-Mechanismus zu ergänzen, um unveränderte Statecharts nicht neu erzeugen zu müssen. Daher wurde KielCmd um einen einfachen Caching-Mechanismus erweitert, der folgendermaßen arbeitet:

- Soll das Caching verwendet werden, so müssen alle Parameter an KielCmd in einem Jobfile und nicht an der Kommandozeile gesetzt werden.
- Wird KielCmd mit aktivem Caching gestartet, so werdem im ersten Schritt Prüfsummen des Jobfiles und der Statechart-Eingabedatei gebildet.
- Aus diesen Prüfsummen wird ein Dateiname konstruiert.
- Existiert im Cache-Verzeichnis bereits eine Datei mit diesem Namen, so wird die existierende Datei als Ausgabedatei kopiert und KielCmd beendet sich.

- Wird im Cache keine passende Datei gefunden, so erzeugt KielCmd die Statechart-Abbildung wie üblich, legt sie aber zusätzlich unter dem oben gebildeten Dateinamen im Cache ab, so dass sie ab dem nächsten Lauf zur Verfügung steht.

## 4.3 Das Modul kiel.preferences

Zu den Anforderungen an KielTEX gehört, dass alle KIEL-Programmeinstellungen von LATEX aus veränderbar sind (s. Abschnitt 3.2). Das ursprüngliche Konfigurationssystem von KIEL sah zwei Möglichkeiten vor, Programmeinstellungen anzupassen: Dauerhafte Änderungen wurden in den Konfigurationsdateien getätigt, Änderungen, die nur zur Laufzeit wirksam waren, waren über Funktionen möglich, die nach dem Namen der Einstellung benannt waren, z. B. `LayouterProperties.setGraphizDpi(int)`. Die Möglichkeit, Einstellungen anhand ihres Namens temporär zu setzen, existierte nicht.

Um den wahlfreien Zugriff auf die Einstellungen zu ermöglichen, wurde das Konfigurationssystem von KIEL vollständig überarbeitet. Die bisher in allen Modulen einzeln vorhandenen Funktionen zum Laden und Speichern von Einstellungen wurden vereinheitlicht und in dem Modul *kiel.preferences* zusammengefasst. Zusätzlich wurde eine zentrale Instanz geschaffen, die den Zugriff auf alle Einstellungen ermöglicht.

Im Rahmen dieser Umstellung wurde das Speicherformat der Konfigurationsdateien auf ein XML-Format umgestellt (Abb. 4.3). Um die Konfigurationsdateien bezüglich Aufbau und Inhalt validieren zu können, wird für jede Konfiguration ein *XML-Schema* definiert.

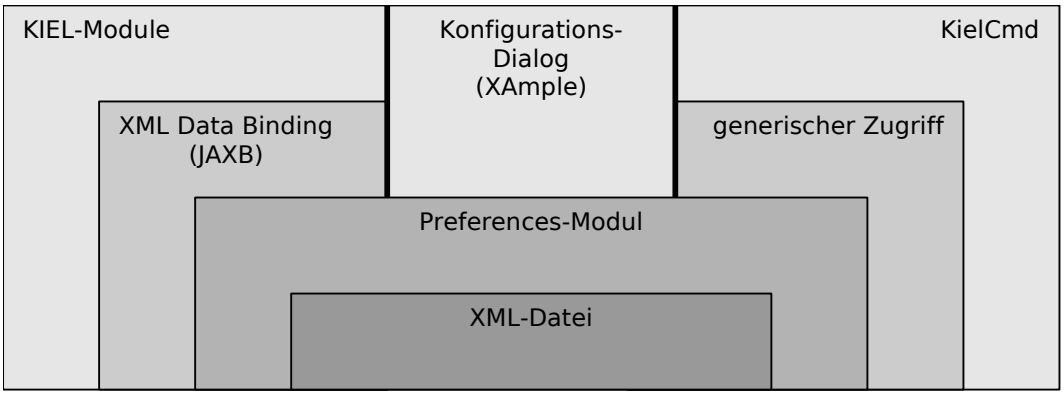



**Abbildung 4.3:** Aufbau von KIELs neuem Konfigurationssystem

### 4.3.1 Die `XMLPreferences`-Klasse

Das zentrale Element von KIELs überarbeitetem Konfigurationssystem ist die Klasse `XMLPreferences`. Sie bildet eine Abstraktionsschicht um die Konfigurationsdateien und stellt ein Benachrichtigungssystem zur Verfügung, mit dem Komponenten von KIEL über geänderte Einstellungen informiert werden können.

Die *XMLPreferences*-Klasse besteht aus zwei Teilen, dem statischen Programmcode und dem Instanz-Programmcode.

#### Instanzmethoden

Die Klasseninstanzen repräsentieren jeweils die Einstellungen eines KIEL-Moduls. Sie enthalten Funktionen zum Laden, Speichern und Validieren der Konfigurationsdateien und erlauben den Zugriff auf die Programmeinstellungen in Form eines DOM-Baums. Die Instanzmethoden der XMLPreferences-Klasse sind in Listing 4.3 aufgeführt.

**Listing 4.3:** Instanzfunktionen der `XMLPreferences`-Klasse

```
// update configuration store with given Data
public void updatePreferences(final Document dom);
public void updatePreferences(final DOMSource source);

// access XML Data
public Document getSchema();
public Document getPreferences();
public URL getXMLSchemaURL();
public URL getXMLDefaultsURL();
public URL getXMLPreferencesURL();

// filesystem interaction
public boolean save();
public void loadDefaults();

// notify Observers about changed settings
public boolean update();

// logging functions
public void setLogger(final LogFile pl);
public LogFile getLogger();
```

Beim Initialisieren einer *XMLPreferences*-Instanz wird die Benutzerkonfiguration gelesen und in einem DOM-Objekt gespeichert. Tritt beim Lesen oder Parsen der Konfiguration ein Fehler auf, so wird die Konfigurationsdatei des Benutzers durch die Vorgabewerte ersetzt und anschließend erneut ausgelesen.

#### Statische Methoden

Um sicherstellen zu können, dass für jedes Modul nur eine `XMLPreferences`-Instanz erstellt wird, verfügt die Klasse über keinen öffentlichen Konstruktor. Stattdessen verfügt die Klasse über statische Methoden zur Erzeugung der Instanzen. Diese stellen sicher, dass für jedes Modul nur eine Instanz erzeugt wird. Weitere Anfragen für das selbe Modul liefern dann eine Referenz auf die bereits erzeugte Instanz zurück. Dieses Verhalten entspricht dem *Singleton*-Pattern [13]. Des weiteren wird eine Liste bereits vorhandener Instanzen gespeichert, die beispielsweise für den Konfigurationsdialog benötigt wird. Die öffentlichen statischen Methoden der XMLPreferences-Klasse sind in Listing 4.4 aufgeführt.

**Listing 4.4:** Statische Methoden der *XMLPreferences*-Klasse

```
// Factory methods for XMLPreferences instances
public static synchronized XMLPreferences createInstance(
  String moduleName, URL schemaURL, URL defaultURL, LogFile logger);
public static XMLPreferences createInstance(
  String moduleName, URL schemaURL, URL defaultURL);

// get an instance without creating it
public static XMLPreferences getInstance(String moduleName);

// generic option setter
public static void setProperty(String module, String property,);

// XML namespace helper function
public static String getModuleNamespace(String module);

// deprecated listener interface
public interface Listener;
public static void addListener(Listener listener);
public static void notifyListeners();

// logging interface
public static int getLoglevel();
public static LogFile getStaticLogger();
public static void log(int level, String message);
```

### 4.3.2 Modulweiser Zugriff auf KIELs Properties

Alle Module in KIEL, die auf persistente Einstellungen zugreifen, verfügen über eine Properties-Klasse (z. B. *BrowserProperties*, *LayouterProperties*, ...), über welche alle Zugriffe auf die Einstellungen des jeweiligen Moduls durchgeführt werden. Zum

Zugriff auf die einzelnen Eigenschaften stellen die Properties-Klassen `get-` und `set-`Methoden bereit.

Wenn ein KIEL-Modul geladen wird, registriert es sich bei dem `XMLPreferences`-Modul und erhält die Referenz auf die dem Modul zugeordnete `XMLPreferences`-Instanz:

```
public final class LayouterProperties {
    static {
        xmlprefs = XMLPreferences.createInstance("layouter",
                LayouterProperties.class);
    }
```

Die `XMLPreferences`-Instanz enthält die Einstellungen als DOM-Baum. Die `get-` und `set`-Methoden operieren allerdings nicht direkt auf dem DOM-Objekt, sondern auf einem Konfigurationsobjekt, welches mit *XML Data Binding* aus dem DOM-Baum gewonnen wird (s. Abschnitt 2.3.1).

### 4.3.3 Generisches Setzen von Optionen

Der Zugriff über JAXB auf die Einstellungen bietet zwar einige Vorteile, insbesodere die inhärente Typkonversion, er erlaubt aber keinen generischen Zugriff auf Einstellungen anhand des Property-Namens. Daher wurd in `XMLPreferences` ein alternativer Zugriffsmechanismus implementiert, welcher direkt die Eintragungen im DOM-Baum ändert. Die Funkion

```
public static void setProperty(final String module, final String property,
        final String value, final boolean validate, final boolean reload);
```

setzt in dem Modul `module` die Einstellung `property` auf den Wert `value`.

Zum Setzen der Werte in dem DOM-Objekten wird XPath verwendet. XPath [43] ist eine Adressierungssprache für Teile von XML-Bäumen. Mit XPath ist es z. B. möglich, Elemente anhand ihres Typs, ihrer Attribute und ihrer hierarchischen Position absolut im Baum oder relativ zu anderen Elementen zu adressieren (Listing 4.5).

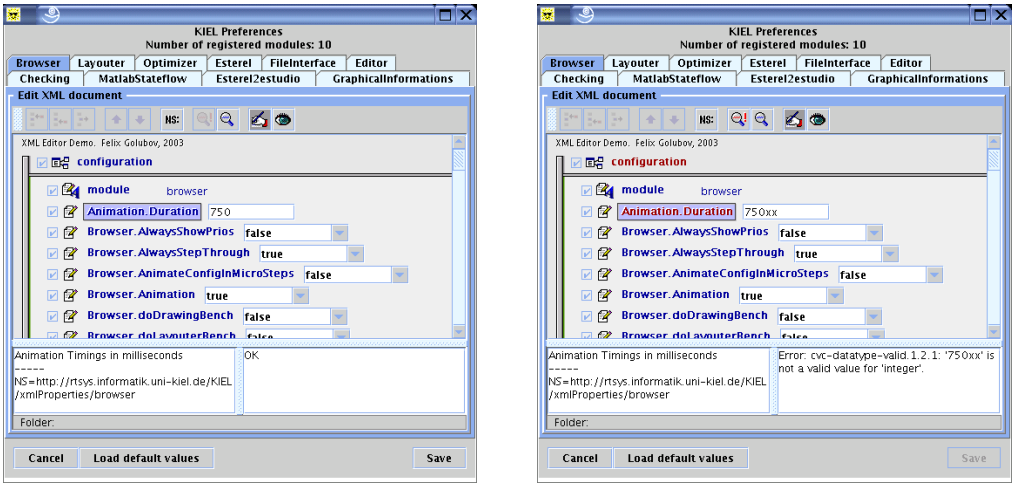
**Listing 4.5:** Setzen von Properties im DOM-Baum mit XPath (ohne Namespaces)

```
JXPathContext context = JXPathContext.newContext(xmldata);
Node node = (Node) context.selectSingleNode("configuration/" + property);
node.getFirstChild().setNodeValue(value);
```

Da KIELs Module zum Teil dynamisch geladen werden, ist es möglich, dass versucht wird, Einstellungen für noch nicht geladene Module zu setzen. Damit diese Änderungen nicht verloren gehen, werden sie in einem Zwischenspeicher hinterlegt und gesetzt, sobald das Modul geladen und initialisiert ist.

### 4.3.4 Der Konfigurationsdialog

Mit der Klasse `kiel.preferences.XMLConfigDialog` wurde ein Konfigurationsdialog (s. Abb. 4.4) implementiert. Der Dialog enthält für jedes Modul eine Registerkarte, in welcher die Einstellungen des Moduls bearbeitet werden können. Die Reiter werden dynamisch aus den vorhanenen *XMLPreferences*-Instanzen generiert, so dass dynamisch geladene Module erst nach ihrer Initialisierung erscheinen.




**(a)** Gültige Eingabe **(b)** Fehlerhafte Eingabe

**Abbildung 4.4:** Der KIEL-Konfigurationsdialog

Wurden Einstellungen geändert und wird der Konfigurationsdialog beendet, so wird das DOM-Objekt der zum Modul gehörigen XMLPreferences-Instanz aktualisiert. Über den Observer-Mechanismus, einen Mechanismus der Java-API zur Verteilung von Nachrichten über geänderte Objekte, wird die Information über die geänderten Einstellungen dann in KIEL bekannt gemacht. Da bei der Entwicklung von KIEL nicht davon ausgegangen wurde, dass sich Einstellungen zur Laufzeit ändern können und die Konfigurationswerte nur einmalig beim Start des Programmes auslesen, werden Änderungen an manchen Einstellungen erst nach einem Neustart von KIEL wirksam.

Die einzelnen Registerkarten des Konfigurationsdialogs sind Instanzen der Klasse `kiel.preferences.XMLEditPane`. Jedes XMLEditPane-Objekt ist dem XMLPreferences-Objekt des zugehörigen Moduls zugeordnet. Um die Einstellungen zu bearbeiten, wird die *XAmple*-Editierkomponente [15] verwendet. *XAmple* erstellt dynamisch aus den Struktur- und Typinformationen eines XML-Schemas einenen Editor, in welchem XML-Daten, die dem Schema folgen, bearbeitet werden können. Die Kopplung

an ein XML-Schema erlaubt es, Fehleingaben schon bei der Eingabe zu erkennen (s. Abb. 4.4).

Da XAmple als Datenmodell DOM-Objekte verwenden kann, kann es direkt auf dem DOM-Feld des XMLPreferences-Objektes des zugehörigen Moduls operieren.

*XAmple* liefert für viele Datentypen des XML-Schemas bereits Editierfelder mit. Darüberhinaus können für Felder im *XML-Schema* auch eigene Editierklassen angegeben werden. Dafür bedient sich *XAmple* der *appInfo*-Felder, mit denen in XML-Schema jedem Eintrag applikationsspezifische Zusatzinformationen hinzugefügt werden können. In Listing 4.6 wird ein Auszug aus einem XML-Schema gezeigt, in dem zur Bearbeitung des Farbeintrages eine spezielle Editorklasse eingetragen wurde. Mit dem gleichen Mechanismus werden die Klartextbeschreibungen, die in Abb. 4.4 links unten zu sehen sind, dem Schema über die *message*-Felder hinzugefügt.

**Listing 4.6:** Eintrag aus einem XML-Schema mit Angabe einer angepassten Editierklasse

```
⟨xs:element name="BrowserCanvas.StringLabelColor" type="xs:string"⟩
  ⟨xs:annotation⟩
    ⟨xs:appinfo⟩
      ⟨xa:node-info editor-class="kiel.preferences.XColorChooser" message="↩
          Color␣of␣labels"⟩
        ⟨xa:message⟩-----⟨/xa:message⟩
      ⟨/xa:node-info⟩
    ⟨/xs:appinfo⟩
  ⟨/xs:annotation⟩
⟨/xs:element⟩
```

# 5 Ergebnisse

## 5.1 Laufzeitanalyse

Ziel von KielTEX ist es, die Einbindung von Statecharts in LATEX zu optimieren. Neben der Reduzierung der für die Einbindung notwendigen manuellen Schritte ist hierfür auch die Laufzeit der Implementierung von Interesse. Um die Laufzeit von KielTEX quantitativ zu erfassen, wurde ein Reihe von Messungen an einer Gruppe von Statecharts durchgeführt. Ziel der durchgeführten Messungen ist es, eine realistische Einschätzung für die Laufzeit von KielTEX in typischen Anwendungsszenarien zu erhalten. Daher werden zur Messung Modelle in verschiedenen Formaten verwendet, die von KIEL teilweise erst noch konvertiert werden müssen.

Die für die Messung verwendeten Modelle sind dem Fundus der Beispielstatecharts von KIEL entnommen. Ein Großteil entstammt dem *EstBench Esterel Benchmark* [8] und dem *Columbia Esterel Compiler*-Paket [9]. Durch die Verwendung etablierter Benchmarks ist eine gewisse Vergleichbarkeit der Ergebnisse mit anderen Messungen möglich. Die `bintree-`, `quadtree-` und `token-ring`-Statecharts wurden in Esterel Studio synthetisch erstellt, um das Skalierungsverhalten von KIEL untersuchen zu können. Bei den übrigen Modellen handelt es sich um von Hand erzeugte Statecharts, die im Rahmen der Entwicklung von KIEL entstanden. Die zur Messung verwendeten Statecharts decken einen weiten Bereich typischer Statecharts ab.

### 5.1.1 Durchführung

Für die Laufzeitmessung wird für jedes Statechart eine minimale LATEX-Datei angelegt (Listing 5.1). Es wurden pro Statechart vier Messungen durchgeführt:

- (M1) Ein LATEX-Lauf ohne den Aufruf externer Programme wie KielCmd. Es wird die reine LATEX-Laufzeit erfasst.

  Kommando: `pdflatex ⟨testdatei⟩.tex`

- (M2) Ein kompletter LATEX-Lauf inklusive der Statechart-Erzeugung mit KielCmd und der anschließenden Nachbearbeitung der Ausgebdatei ins EPS- und PDF-Format.

  Kommando: `pdflatex -shell-escape ⟨testdatei⟩.tex`

- (M3) Ein weiterer LaTeX-Lauf ohne den Aufruf externer Programme. Diesmal werden die aus dem zweiten Lauf noch vorhandenen Statechart-Grafiken eingebunden.

  Kommando: `pdflatex ⟨testdatei⟩.tex`

- (M4) Zum Vergleich wurde im vierten Schritt die reine Laufzeit von KielCmd mit dem von LaTeX erzeugten Jobfile ermittelt.

  Kommando: `kielcmd -j=⟨testdatei⟩-fig1.job`

Als TeX-Compiler wird bei den Messungen pdfTeX[1] verwendet. Für jede Messung werden 20 Einzelmessungen durchgeführt, von denen die ersten fünf verworfen werden, um Einschwing- und Cachingeffekte zu vermeiden. Von den übrigen 15 Einzelmessungen wird das arithmetische Mittel gebildet. Die Messungen wurden auf einem PC mit 2,6 GHz Athlon 64-Prozessor und 2 GB RAM unter Linux durchgeführt.

### 5.1.2 Ergebnis der Laufzeitmessung

Die Messergebnisse zeigt Tabelle 5.1. Eine Reihe von erzeugten Statechart-Abbildungen kann von LaTeX nicht verarbeitet werden (*mca200*, *tokenring50*, *tokenring100*, *tokenring200* und *tokenring300*). Die Untersuchung der Fehlerursache ergibt, dass die von KIEL für diese Statecharts generierten Abbildungen so große Abmessungen haben, dass die Länge oder Breite mit TeXs Größenregistern nicht mehr darstellbar ist (das Maximum für eine von TeX verarbeitbare Länge liegt bei ca. 575cm).

**Listing 5.1:** LaTeX-Beispieldatei für die Laufzeitmessung

```
\documentclass{article}
\usepackage{kieltex}
\begin{document}
\includeKIELstatechart[gfxopts={width=\textwidth,height=\textheight,%
    keepaspectratio=true}]{examples/esterel/ctr/abro.strl}
\end{document}
```

Die Auswertung der Messreihen M1 und M3 zeigt, dass LaTeX ohne den Aufruf externer Programme für die Verarbeitung der Testdateien zwischen 0,26s und 1,24s benötigt. Dieses Ergebnis deutet darauf hin, dass die reine Arbeitszeit des TeX-Compilers bei der Verwendung von KielTeX auf heutigen Computersystemen vernachlässigbar ist.

Tabelle 5.1 zeigt weiter, dass selbst bei Statecharts geringer Komplexität die Erzeugung der Abbildungen durch KielCmd knapp fünf Sekunden benötigt. Dies ist durch die Zeit erklärbar, die für die Initialisierung der Java Virtual Machine und der KIEL-Module benötigt wird.

[1] Version 3.141592-1.40.3-2.2 aus der TeXlive 2007-Distribution.

**Tabelle 5.1:** Messergebnisse der experimentellen Laufzeituntersuchungen; angegeben sind die Anzahl der graphischen Elemente (Zustände, Pseudozustände, Transitionen, etc) der Testmodelle und die Laufzeiten der Messreihen M1–M4.

| Statechart | Format | Grafische Elemente | Messung (s) | | | |
|---|---|---|---|---|---|---|
| | | | M1 | M2 | M3 | M4 |
| cabin | KIT | 27 | 0.67 | 8.26 | 0.70 | 6.74 |
| abro | Esterel | 37 | 0.28 | 6.95 | 0.30 | 6.16 |
| StopWatchv5 | scg | 57 | 0.30 | 6.06 | 0.31 | 5.14 |
| watch | Esterel | 210 | 0.28 | 11.06 | 0.33 | 7.72 |
| schizophrenia | Esterel | 42 | 0.27 | 6.92 | 0.28 | 6.05 |
| reincarnation | scg | 22 | 0.27 | 5.56 | 0.29 | 4.89 |
| reincarnation | Esterel | 86 | 0.29 | 7.69 | 0.31 | 6.53 |
| jacky1 | Esterel | 98 | 0.28 | 7.95 | 0.29 | 6.64 |
| runner | Esterel | 131 | 0.29 | 8.72 | 0.29 | 7.06 |
| abcd | Esterel | 750 | 0.28 | 16.99 | 0.34 | 10.31 |
| greycounter | Esterel | 768 | 0.26 | 15.74 | 0.34 | 10.01 |
| ww | Esterel | 1167 | 0.28 | 29.20 | 0.50 | 14.68 |
| mejia | Esterel | 1317 | 0.69 | 37.11 | 1.09 | 17.86 |
| tcint | Esterel | 1614 | 0.28 | 34.15 | 0.40 | 14.69 |
| atds-100 | Esterel | 1577 | 0.29 | 35.26 | 0.42 | 16.77 |
| mca200 | Esterel | 5468 | —(*) | — (*) | —(*) | 155.95 |
| bintree-1 | scg | 9 | 0.27 | 5.37 | 0.28 | 4.78 |
| bintree-2 | scg | 25 | 0.29 | 5.54 | 0.30 | 4.94 |
| bintree-3 | scg | 57 | 0.27 | 5.79 | 0.29 | 5.11 |
| bintree-4 | scg | 121 | 0.29 | 6.18 | 0.31 | 5.22 |
| bintree-5 | scg | 249 | 0.27 | 7.13 | 0.30 | 5.82 |
| bintree-6 | scg | 505 | 0.26 | 8.85 | 0.31 | 6.69 |
| bintree-7 | scg | 1017 | 0.28 | 12.14 | 0.31 | 8.17 |
| bintree-8 | scg | 2041 | 0.28 | 17.79 | 0.33 | 10.87 |
| bintree-9 | scg | 4089 | 0.27 | 29.25 | 0.37 | 17.11 |
| bintree-10 | scg | 8185 | 0.28 | 52.08 | 0.46 | 25.23 |
| quadtree-0 | scg | 1 | 0.91 | 8.02 | 1.21 | 5.62 |
| quadtree-1 | scg | 21 | 0.94 | 10.91 | 1.12 | 7.71 |
| quadtree-2 | scg | 101 | 0.88 | 10.48 | 1.08 | 7.52 |
| quadtree-3 | scg | 421 | 0.84 | 12.92 | 0.96 | 9.28 |
| quadtree-4 | scg | 1701 | 1.24 | 21.90 | 1.10 | 12.43 |
| quadtree-5 | scg | 6821 | 0.74 | 46.00 | 0.41 | 22.97 |
| quadtree-6 | scg | 27285 | 0.27 | 161.21 | 0.87 | 70.92 |
| token-ring3 | Esterel | 272 | 0.29 | 10.31 | 0.31 | 7.70 |
| token-ring10 | Esterel | 874 | 0.33 | 19.40 | 0.67 | 12.48 |
| token-ring50 | Esterel | 1708 | —(*) | — (*) | —(*) | 13.96 |
| token-ring100 | Esterel | 3408 | —(*) | — (*) | —(*) | 23.10 |
| token-ring200 | Esterel | 6808 | —(*) | — (*) | —(*) | 35.57 |
| token-ring300 | Esterel | 10208 | —(*) | — (*) | —(*) | 53.69 |

(*): Diese Abbildungen konnten von TEX nicht bearbeitet werden, da ihre Abmessungen größer als die maximal erlaubte Längeneinheit von TEX sind.

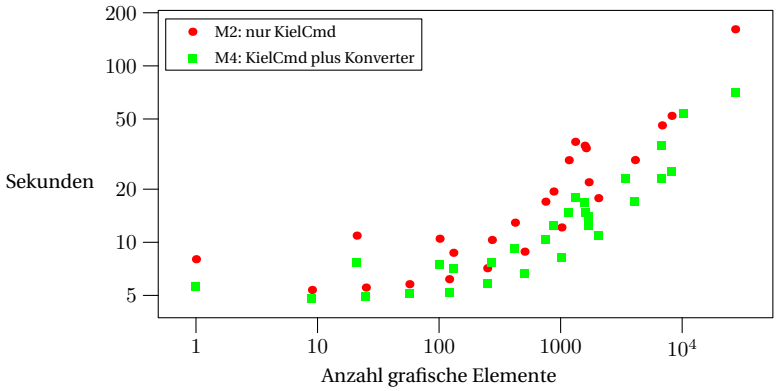



**Abbildung 5.1:** Übersicht über die Ergebnisse aller Laufzeitmessungen, mit und ohne Einbeziehung der Laufzeiten der nachträglichen Konvertierungen

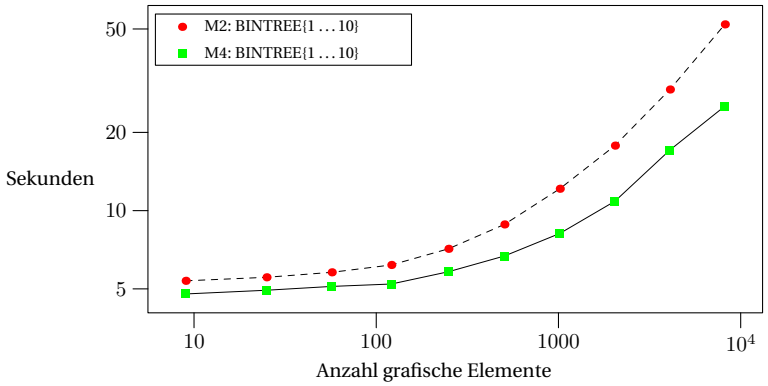



**Abbildung 5.2:** Laufzeit der `bintree`-Reihe

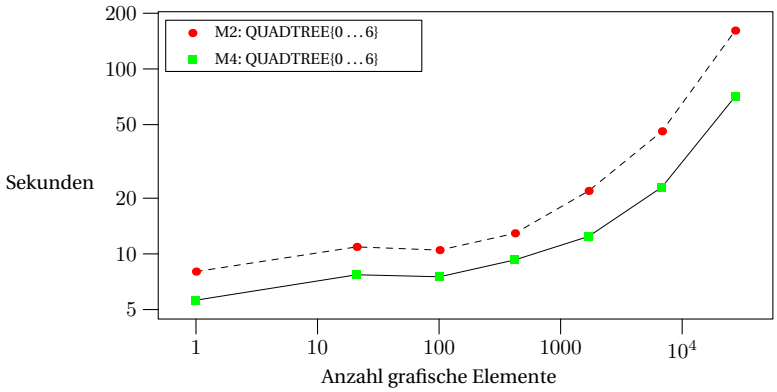



**Abbildung 5.3:** Laufzeit der `quadtree`-Reihe

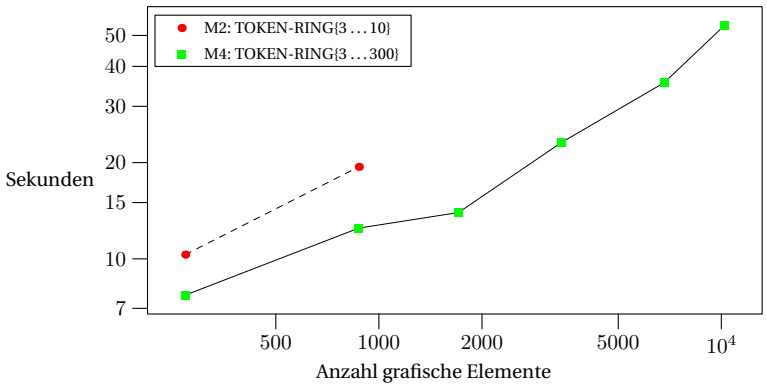



**Abbildung 5.4:** Laufzeit der `Token-Ring`-Reihe

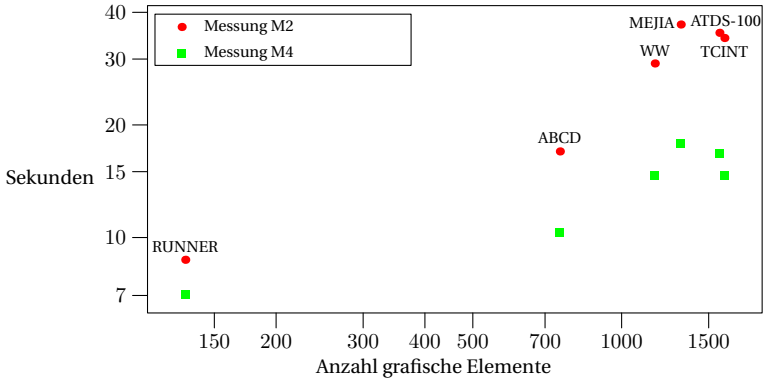



**Abbildung 5.5:** Laufzeit der `EstBench`-Statecharts

Mit zunehmender Komplexität der Statecharts steigt die Laufzeit stark an. Für das komplexeste Statechart der Messung, `quadtree-6`, wurde eine mittlere Dauer von 71 s für den Aufruf von KielCmd und 161 s für die vollständige Bearbeitung durch KielTEX gemessen. Die hohe Differenz zwischen KielCmd und dem endgültigen Ergebnis ergibt sich durch die nachträgliche Konvertierung der Abbildung ins EPS- und PDF-Format. Da beide Stufen der Nachbearbeitung die produzierte Vektorgrafik auswerten müssen, steigt der Zeitaufwand mit der Komplexität der Grafiken. Eine nachträglich durchgeführte Stichprobenmessung deutet darauf hin, dass bei komplexen Statecharts vor allem die Bestimmung der genauen Ausmaße der Abbildung (der *Bounding Box*) einen Großteil der gemessenen Zeit in Anspruch nimmt.

Wie in Abschnitt 4.2.5 erwähnt wurde das Caching bereits erzeugter Abbildungen erst nach Durchführung der Laufzeitmessungen implementiert. Eine nachträglich durchgeführte Stichprobenmessung ergab bei bereits im Cache befindlichen Statecharts

eine Dauer von ca. 3 s für die Laufzeit von KielCmd, unabhängig von der Größe des Statecharts.

Zusammenfassend lässt sich feststellen, dass die Geschwindigkeit von KielTEX für den Arbeitsalltag ausreichend ist, solange nicht sehr viele oder sehr große Statecharts verwendet werden.

## 5.2 Demonstration von KielTEX

Zur Demonstration des KielTEX-Pakets sind auf den folgenden Seiten eine Reihe von KielTEX-Befehlen zusammen mit den erzeugten Ausgaben abgebildet. Die Beispiele veranschaulichen verschiedene Aspekte von KielTEX:

- Steuerung der Grafikanzeige (Abb. 5.6),
- Ansteuerung verschiedener Layouter (Abb. 5.7),
- Kontrolle der Layoutparameter (Abb. 5.8),
- Anzeige aktiver Zustände (Abb. 5.9),
- Statecharts verschiedener Komplexität (Abb. 5.10),
- Konvertierung von Esterel-Programmen in Statecharts (Abb. 5.11),
- Eingebettete Statecharts mit verschiedenen Anzeigemodi (Abb. 5.12 und Abb. 5.13),

**KielTEX-Kommandos:**

```
(1)\includeKIELstatechart[chartwidth=0.1\linewidth]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(2)\includeKIELstatechart[chartwidth=0.2\linewidth]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(3)\includeKIELstatechart[chartwidth=0.4\linewidth]%
   {../../../../examples/kit/spr/traffic-light-short.kit}
```

**Resultierende Ausgabe:**

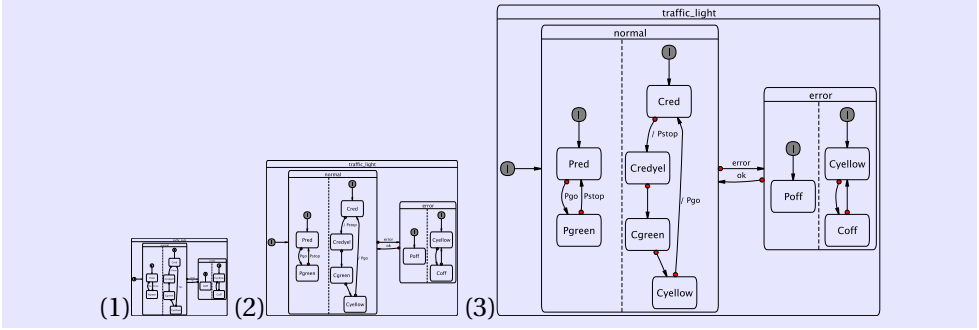



**Abbildung 5.6:** KielTEX-Demonstration: Steuerung der Darstellungsgröße von Statechart-Abbildungen im Text

**KielTEX-Kommandos:**

```
(1)\includeKIELstatechart[chartwidth=5cm]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(2)\includeKIELstatechart[chartwidth=5cm,layouter=HVLayouter]%
  {../../../../examples/kit/spr/traffic-light-short.kit}

(3)\includeKIELstatechart[chartwidth=10cm,layouter=CircoLayouter]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
```

**Resultierende Ausgabe:**

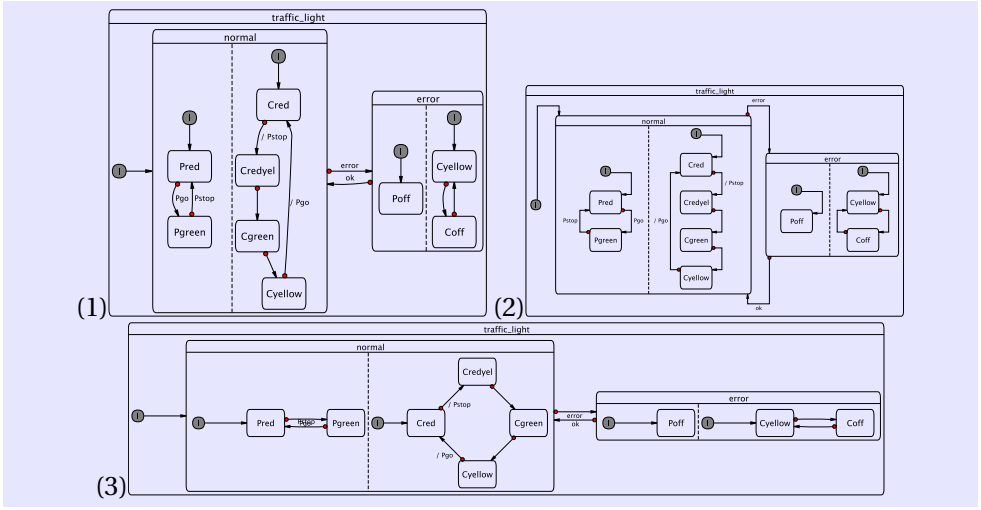



**Abbildung 5.7:** KielTEX-Demonstration: Demonstration des DotLayouters (1), HVLayouters (2) und CircoLayouters (3)

**KielTEX-Kommandos:**

```
(1)\includeKIELstatechart[chartwidth=5cm]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(2)\includeKIELstatechart[chartwidth=5cm,kielopts={layouter.graphviz.rankdir.
    horizontal=RL}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}

(3)\includeKIELstatechart[chartwidth=5cm,kielopts={layouter.graphviz.rankdir.
    vertical=BT}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(4)\includeKIELstatechart[chartwidth=5cm,kielopts={layouter.graphviz.rankdir.
    horizontal=RL,
     layouter:graphviz.rankdir.vertical=BT}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
```

**Resultierende Ausgabe:**

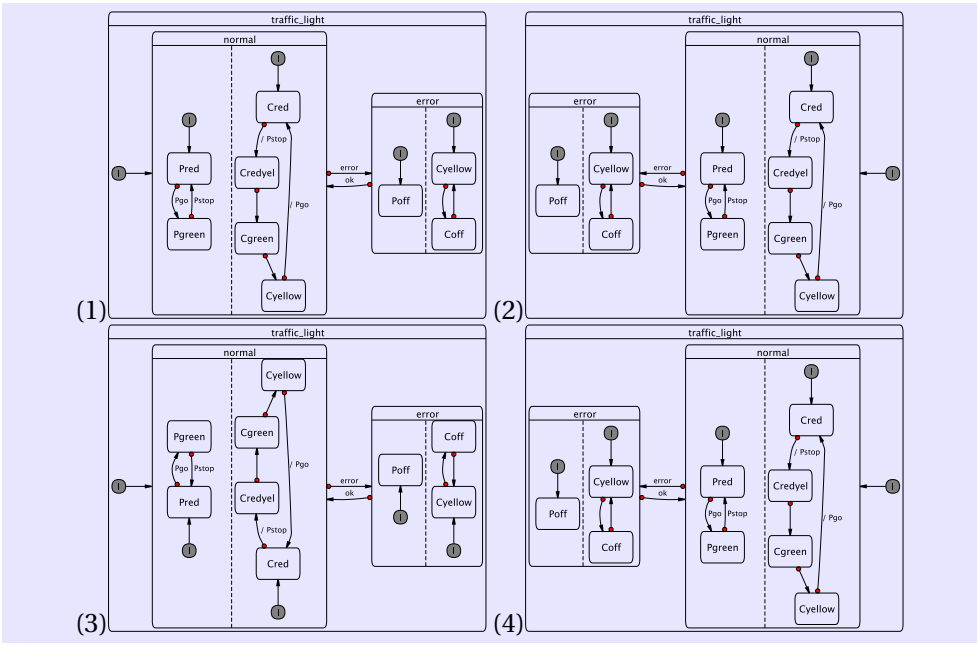



**Abbildung 5.8:** KielTEX-Demonstration: Modifikation der Layout-Parameter des Dot-Layouters (Horizontale und vertikale Orientierung)

**KielTEX-Kommandos:**

```
(1)\includeKIELstatechart[chartwidth=5cm,config={Pred Cgreen}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(2)\includeKIELstatechart[chartwidth=5cm,config={Poff Coff}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}

(3)\includeKIELstatechart[chartwidth=5cm,config={Pred Cgreen},kielopts={layouter.
    semanticZoom=false}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
%
(4)\includeKIELstatechart[chartwidth=5cm,config={Poff Coff},kielopts={layouter.
    semanticZoom=false}]%
  {../../../../examples/kit/spr/traffic-light-short.kit}
```

**Resultierende Ausgabe:**

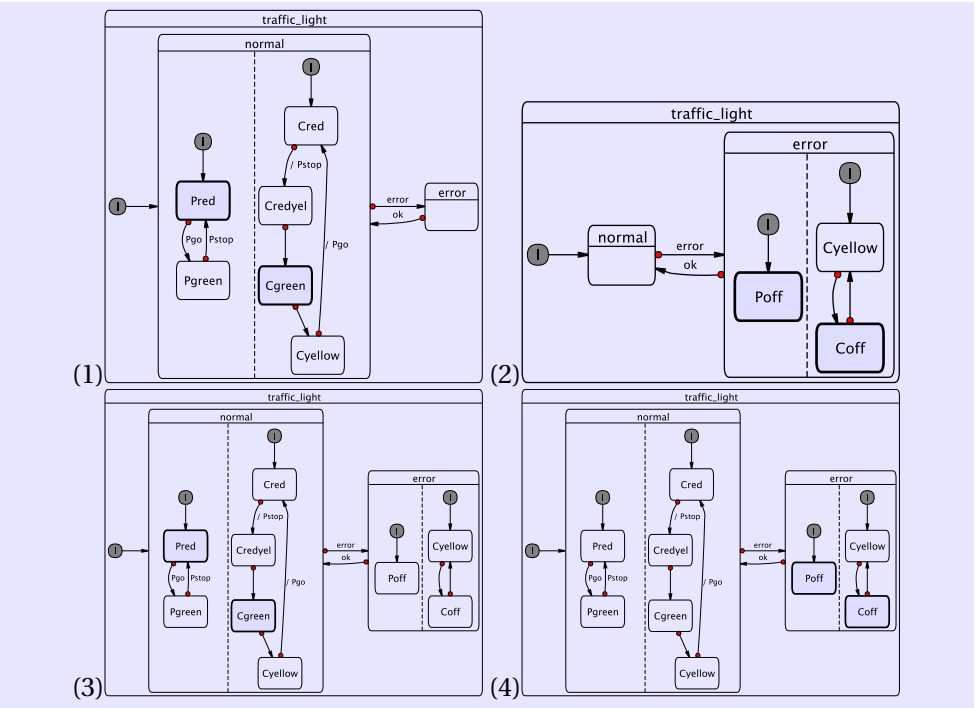



**Abbildung 5.9:** KielTEX-Demonstration: Auswahl von Konfigurationen über Angabe aktiver Zustände, mit und ohne dynamischer Ausblendung nichtaktiver hierarchischer Zustände

**KielTEX-Kommandos:**

```
(1)\includeKIELstatechart[chartwidth=0.2\linewidth]%
{../../../../examples/esterelstudio/spr/date06/bintree-1.scg}
(2)\includeKIELstatechart[chartwidth=0.65\linewidth]%
{../../../../examples/esterelstudio/spr/date06/bintree-10.scg}
```

**Resultierende Ausgabe:**


**Abbildung 5.10:** KielTEX-Demonstration: Einfaches (`bintree-1`) und komplexes (`bintree-10`) Statechart

**KielTEX-Kommandos:**

```
\KIELsetup{chartwidth=0.4\linewidth,sourcewidth=0.4\linewidth}

(1)\includeKIELstatechart[show=split]%
  {../../../../examples/esterel/spr/await1.strl}

(2)\includeKIELstatechart[show=split,optimize=true]%
  {../../../../examples/esterel/spr/await1.strl}
```

**Resultierende Ausgabe:**

(1) **paper-fig18.pdf not found!**

```
module module_await:
output A,B;

[
  emit A;
  pause;
  pause; emit A
||
  await A;
  emit B
]

end module
```

(2) **paper-fig19.pdf not found!**

```
module module_await:
output A,B;

[
  emit A;
  pause;
  pause; emit A
||
  await A;
  emit B
]

end module
```

**Abbildung 5.11:** KielTEX-Demonstration: Aus Esterel generiertes Statechart ohne und mit Optimierung

**KielTEX-Kommandos:**

```
\begin{KIELstatechart}[language=Esterel,optimize=true,chartwidth=0.6\linewidth]
module ABRO:

input A, B, R;

output O;

loop
  [ await A || await B];
  emit O;
each R

end module
\end{KIELstatechart}
```

**Resultierende Ausgabe:**

**paper-fig20.pdf not found!**

**Abbildung 5.12:** KielTEX-Demonstration: Einbettung von Statechart-Quellcode (ABRO) im LATEX-Dokument

**KielTEX-Kommandos:**

```
\KIELsetup{gfxopts={width=0.5\linewidth}}

\begin{KIELstatechart}[language=Esterel,show=source]
module ABRO:

input A, B, R;

output O;

loop
  [ await A || await B];
  emit O;
each R

end module
\end{KIELstatechart}
```

**Resultierende Ausgabe:**

```
module ABRO:

input A, B, R;

output O;

loop
  [ await A || await B];
  emit O;
each R

end module
```

**Abbildung 5.13:** KielTEX-Demonstration: Eingebettetes ABRO-Esterel-Statechart, dargestellt als Quelltext

# 6 Zusammenfassung und Ausblick

Statecharts sind ein etabliertes Werkzeug zur grafischen Modellierung reaktiver Systeme, vor allem eingebetteter Systeme. Das KIEL-Framework bildet den Rahmen für Techniken zur Verbesserung der Modellierung von Statecharts. Viele der Komponenten von KIEL wurden mit dem Fokus auf die interaktive Benutzeroberfläche entwickelt.

In dieser Arbeit wurde untersucht, wie sich die vorhandene Funktionalität von KIEL besser in das Schreiben von Texten mit LaTeX integrieren lässt. Mit dem LaTeX-Paket KielTeX wurde dazu eine Möglichkeit geschaffen, ohne einen Wechsel der Arbeitsumgebung direkt im LaTeX-Dokument an Statecharts zu arbeiten und die Parameter jedes einzelnen Statecharts individuell anzupassen. KielTeX benutzt zur Verarbeitung von Statecharts die KIEL-Kommandozeilenschnittstelle KielCmd, die dafür im Rahmen dieser Arbeit um weitere Funktionen ergänzt wurde. Beispielsweise ist es nun möglich, Programmeinstellungen von KIEL nur für die Erzeugung eines Statecharts zu verändern, ohne die Konfigurationsdateien von KIEL zu editieren.

Die dafür durchgeführte Erweiterung von KielCmd bringt nicht nur im LaTeX-Betrieb viele Vorteile. Die Einführung von *Jobfiles* zur Speicherung des Programmeinstellungen von KIEL für ein bestimmtes Statechart und die Möglichkeit, Einstellungen nichtpersistent an der Kommandozeile zu setzen, erleichtert es dem Anwender, Resultate und Ausgaben von KIEL abzuspeichern und einfach zu reproduzieren.

Um die Änderung von Programmeinstellungen über die Kommandozeile zu ermöglichen, wurde das Konfigurationssystem von KIEL neu implementiert. Im Zuge dessen wurde das Speicherformat der Einstellungen in ein XML-Format überführt. Die Neuimplementierung des Konfigurationssystems bietet darüber hinaus dem Benutzer der grafischen Oberfläche von KIEL einen interaktiven Konfigurationsdialog, welcher fehlerhafte Einstellungen schon während des Editierens erkennt und markiert. Der Konfigurationsdialog greift dafür auf XML-Schema-Dokumente zurück, welche den korrekten Aufbau und Inhalt der Einstellungen festlegen. Die XML-Schemata werden auch dafür verwendet, beim Starten von KIEL alle Konfigurationsdateien auf Korrektheit zu überprüfen. Darüber hinaus werden aus den Schemata mit dem Verfahren des *XML Data Binding* Zugriffsklassen generiert, die dem Entwickler eine unkomplizierte Verwendung der XML-Programmeinstellungen ermöglichen.

Die im Anschluss an die Implementierung vorgenommenen Laufzeitanalysen zeigen, dass die Geschwindigkeit von KielTEX für Dokumente mit kleinen oder wenigen Statecharts schon mehr als ausreichend ist. Für die Zukunft wäre dennoch eine Optimierung der Ausführungszeit wünschenswert, insbesondere bei komplexen Statecharts mit mehreren tausend Elementen. Besonders die nachträgliche Konvertierung der exportierten Statechart-Abbildungen ins EPS- und PDF-Format mit Werkzeugen wie `ps2eps` und `epstopdf` ist bei diesen Statecharts zeitaufwendig. Ein direkt in KIEL integrierter PDF-Export könnte hier eine Erhöhung der Verarbeitungsgeschwindigkeit bewirken.

Wie in Abschnitt 5.2 gezeigt wird, vereinfacht KielTEX das Arbeiten mit Statecharts in LATEX-Texten erheblich. Es ist nun schnell und unkompliziert möglich, beispielsweise eine Reihe von Abbildungen zu erzeugen, bei denen das gleiche Statechart jeweils mit unterschiedlichen aktiven Zuständen dargestellt wird.

KielTEX wurde in LATEX entwickelt. Trotz der unverändert großen Verbreitung von LATEX als Textsatzprogramm im akademischen Bereich findet eine Beschäftigung mit TEX als Programmiersprache an den Universitäten nur selten statt. Eine Ursache ist sicher das Fehlen von klassischen Programmierwerkzeugen wie z. B. Debuggern. So ist das Verfassen selbst einfacher Programme vergleichsweise fehleranfällig. Die aktuelle Entwicklung, den TEX-Compiler um die Skriptsprache `Lua` [**?**, **?**] zu erweitern, ist sicher ein Schritt in die richtige Richtung.

# 7 Literaturverzeichnis


[1] *JAXB Reference Implementation Project*, 2006. https://jaxb.dev.java.net/.

[2] *The pstricks package*, 2007. http://tug.org/PSTricks/.

[3] ADOBE SYSTEMS INCORPORATED: *Encapsulated PostScript (EPS) File Format Specification Version 3.0 #5002*, Mai 1992. http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf.

[4] ADRIAENS, HENDRI: *The xkeyval Package*. ftp://tug.ctan.org/pub/tex-archive/macros/latex/contrib/xkeyval/doc/xkeyval.pdf.

[5] BEECK, MICHAEL VON DER: *A Comparison of Statecharts Variants*. In: LANGMAACK, H., W. P. DE ROEVER und J. VYTOPIL (Herausgeber): *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Band 863 der Reihe *Lecture Notes in Computer Science*, Seiten 128–148. Springer-Verlag, 1994.

[6] BERRY, GÉRARD: *The Esterel v5 Language Primer, Version v5_91*. Centre de Mathématiques Appliquées Ecole des Mines and INRIA, 06565 Sophia-Antipolis, 2000. ftp://ftp-sop.inria.fr/esterel/pub/papers/primer.pdf.

[7] CARLISLE, DAVID und S. P. Q. RAHTZ: *The graphicx package*, 1999. ftp://ctan.org/tex-archive/macros/latex/required/graphics.

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

[9] EDWARDS, STEPHEN A.: *CEC: The Columbia Esterel Compiler*. http://www1.cs.columbia.edu/~sedwards/cec/.

[10] ESTEREL TECHNOLOGIES: *Free Sofware Esterel Compiler*. http://www.esterel-technologies.com/technology/free-software/esterel-compiler.html.

[11] ESTEREL TECHNOLOGIES: *Esterel Studio*. http://www.esterel-technologies.com/products/esterel-studio/overview.html.

[12] *e-TEX repository on CTAN*, 2004. http://ctan.org/tex-archive/systems/e-tex/.

[13] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns*. Addison-Wesley, 1995.

[14] GERSTBACH, PETER: *XML Data Binding*. Bakkalaureatsarbeit, TU Wien, Wien, November 2004. http://www.gerstbach.at/2004/XMLDataBinding/.

[15] GOLUBOV, FELIX: *XAmple XML Editor*, 2007. http://www.felixgolubov.com/XMLEditor/.

[16] GRAPHVIZ: *GraphViz—Graph Drawing Tools*. http://graphviz.org/.

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

[18] HAREL, DAVID, HAGI LACHOVER, AMNON NAAMAD, AMIR PNUELI, MICHAL POLITI, RIVI SHERMAN, AHARON SHTULL-TRAURING und MARK TRAKHTENBROT: *STATEMATE: A Working Environment for the Development of Complex Reactive Systems*. IEEE Transactions on Software Engineering, 16(4):403–414, April 1990.

[19] HAREL, DAVID und AMNON NAAMAD: *The STATEMATE Semantics of Statecharts*. ACM Transactions on Software Engineering and Methodology, 5(4):293–333, Oktober 1996.

[20] HEINZ, CARSTEN: *The Listings Package*, 2004. ftp://tug.ctan.org/pub/tex-archive/macros/latex/contrib/listings/listings.pdf.

[21] KLOSS, TOBIAS: *Automatisches Layout von Statecharts unter Verwendung von GraphViz*. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Mai 2005.

[22] KNUTH, DONALD E.: *A torture test for TEX, Version 3*. Stanford University, Januar 1990. ftp://tug.ctan.org/pub/tex-archive/systems/knuth/tex/tripman.tex.

[23] LAMPORT, LESLIE: *LaTEX A Document Preparation System*. Adddison-Wesley, 1994.

[24] *LaTeX – A document preparation system*, visited 04/2007. http://www.latex-project.org/.

[25] MANGNER, TORSTEN: *Verarbeitung, Austausch und Speicherung von Dokumenten mit Hilfe von XML Data Binding*. Studienjahresarbeit, Technische Universität Ilmenau, 2002. http://www.imt.tu-ilmenau.de/~schoen/prakinf-studentenarbeiten/sa-2002-mangner.pdf.

[26] MEGGINSON, DAVID: *SAX 2.0: The Simple API for XML*, 2000. http://www.megginson.com/SAX/index.html.

[27] OBERDIEK, HEIKO: *The kvsetkeys package*, 2006. http://ctan.org/tex-archive/macros/latex/contrib/oberdiek.

[28] OBJECT MANAGEMENT GROUP: *Unified Modeling Language: Superstructure, Version 2.0*, Aug 2005. http://www.omg.org/docs/formal/05-07-04.pdf.

[29] *pdfTEX homepage*, visited 04/2007. http://www.tug.org/applications/pdftex/.

[30] PROCHNOW, STEFFEN und REINHARD VON HANXLEDEN: *Statechart Development Beyond WYSIWYG*. In preparation.

[31] PROCHNOW, STEFFEN und REINHARD VON HANXLEDEN: *Visualisierung komplexer reaktiver Systeme – Annotierte Bibliographie*. Technical Report 0406, Christian-Albrechts-Universität Kiel, Department of Computer Science, Juni 2004. http://www.informatik.uni-kiel.de/uploads/tx_publication/2004_tr06.pdf.

[32] PROCHNOW, STEFFEN, CLAUS TRAULSEN und REINHARD VON HANXLEDEN: *Synthesizing Safe State Machines from Esterel*. In: *Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, Ottawa, Canada, Juni 2006.

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

[34] SCHÖPF, RAINER, BERND RAICHLE und CHRIS ROWLEY: *A New Implementation of LaTeX's verbatim and verbatim* Environments*, 2001. ftp://tug.ctan.org/pub/tex-archive/macros/latex/required/tools/verbatim.pdf.

[35] SESHIA, SANJIT A., R. K. SHYAMASUNDAR, A. K. BHATTACHARJEE und S. D. DHODAPKAR: *A Translation of Statecharts to Esterel*. In: WING, J., J. WOODCOCK und J. DAVIES (Herausgeber): *FM'99 volume 2— World Congres on Formal Methods*, Band 1709 der Reihe *LNCS*, Seiten 983–1007. Springer-Verlag, 99.

[36] SUN DEVELOPER NETWORK: *Java Homepage*. http://java.sun.com/.

[37] SUN MICROSYSTEMS, INC.: *The Java Web Services Tutorial 1.6*, 2005. http://java.sun.com/webservices/docs/1.6/tutorial/doc/.

[38] TANTAU, TILL: *PGF and TikZ – Graphic systems for TeX*, 2007. http://sourceforge.net/projects/pgf/.

[39] THE KIEL PROJECT (KIEL INTEGRATED ENVIRONMENT FOR LAYOUT): *Project Homepage*, 2006. http://www.informatik.uni-kiel.de/rtsys/kiel/.

[40] THE MATHWORKS: *Stateflow—Design and simulate event-driven systems*. http://www.mathworks.com/products/stateflow/.

[41] THOMPSON, HENRY S., DAVID BEECH, MURRAY MALONEY und NOAH MENDELSOHN: *XML Schema Part 1: Structures Second Edition*. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, Oktober 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028.

[42] W3C DOM IG: *W3C Document Object Model*, 2007. http://www.w3.org/DOM/.

[43] WORLD WIDE WEB CONSORTIUM (W3C) : *XPath Homepage*. http://www.w3.org/TR/xpath.

# A Tabeller aller Properties von KIEL

Die im Folgenden abgedruckte Tabelle listet alle Properties von KIEL zur Zit der Entstehung dieser Arbeit auf. Die einzelnen Tabellenspalten haben folgenden Inhalt:

**Modul** Das KIEL-Modul der Properties. Jedes Modul speichert seine Einstellungen in einer eigenen Datei.

**Property** Der Name der Property.

**Beschreibung** Eine kurze Beschreibung der Property, die auch im Konfigurationsdialog angezeit wird.

**Typ** Der Datentyp der Property. Er wird verwendet, um die Editorkomponente auszuwählen, die für die zugehörige Property im Konfigurationsdialog angezeigt wird. Gleichzeitig wird der Datentyp bei der generierung der Zugriffsfunktionen für das Language Binding verwendet. Nicht zuletzt dient der Datentyp dazu, die Konfigurationsdateien beim Starten von KIEL auf korrekte Einträge zu überprüfen.

**Alternativen** Für einige Optionen soll einen Menge von gültigen Werten in einer Auswahlbox angeboten werden. Die Wahlmöglichkeiten sind hier aufgelists.

**Neust.** Gibt an, ob ein Neustart des KIEL-GUI notwendig ist, um die Änderung der Einstellung wirksam werden zu lassen. Es gibt drei mögliche Werte:

- — Ob ein Neustart notwendig ist, ist nicht bekannt
- **Ja** Es ist ein Neustart des KIEL-GUI notwendig
- **Nein** Es ist kein Neustart des KIEL-GUI notwendig, die Änderungen werden sofort oder beim Laden des nächsten Statecharts wirksam.

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| browser | Animation.Duration | Animation Timings in milliseconds | int | | — | 750 |
| browser | Browser.AlwaysShowPrios | Always show Priorities | Boolean | | Nein | false |
| browser | Browser.AlwaysStepThrough | Always step through | Boolean | | Nein | true |
| browser | Browser.AnimateConfigInMicroSteps | Animate config in ministeps | Boolean | | Nein | false |
| browser | Browser.Animation | Animate | Boolean | | Nein | true |
| browser | Browser.ColorizeNewConfigurations | Colorize newly set Configurations | Boolean | | Nein | false |
| browser | Browser.doDrawingBench | Benchmark Drawing | Boolean | | — | false |
| browser | Browser.doLayouterBench | Benchmark Layouter | Boolean | | — | false |
| browser | Browser.doSimulatorBench | Benchmark Simulator | Boolean | | — | false |
| browser | Browser.doSimulatorOutput | Do simulator output | Boolean | | — | true |
| browser | Browser.DrawingBenchFile | Drawing-Benchmark Filename Prefix | String | | — | DrawingBench_ |
| browser | Browser.HighlightActiveStates | Colorize active states | Boolean | | Nein | true |
| browser | Browser.KeyStroke.AddChoice | Key: Add choice | String | | — | ctrl +K |
| browser | Browser.KeyStroke.AddDynamicChoice | Key: Add dynamic choice | String | | — | ctrl +L |
| browser | Browser.KeyStroke.AddHistory | Key: Add history | String | | — | ctrl +M |
| browser | Browser.KeyStroke.AddORState | Key: Add OR state | String | | — | ctrl +J |
| browser | Browser.KeyStroke.AddState | Key: Add state | String | | — | ctrl +I |
| browser | Browser.KeyStroke.AddSuspend | Key: Add Suspend | String | | — | ctrl +N |
| browser | Browser.KeyStroke.AlwaysFitScreen | Key: Always fit screen | String | | — | ctrl +F |
| browser | Browser.KeyStroke.ChangeEdgeLabel | Key: Change edge label | String | | — | ctrl +V |
| browser | Browser.KeyStroke.ChangeNodeLabel | Key: change node label | String | | — | ctrl +U |
| browser | Browser.KeyStroke.CountElements | Key: Count elements | String | | — | ctrl +B |
| browser | Browser.KeyStroke.FitScreen | Key: Fit screen | String | | — | ctrl +G |
| browser | Browser.KeyStroke.ForceUpdate | Key: Force update | String | | — | ctrl +W |
| browser | Browser.KeyStroke.LoadTrace | Key: Load trace | String | | — | ctrl +E |
| browser | Browser.KeyStroke.NewStatechart | Key: New statechart | String | | — | ctrl +C |
| browser | Browser.KeyStroke.NewTransition | Key: New transistion | String | | — | ctrl +R |
| browser | Browser.KeyStroke.RemoveObjects | Key: Remove objects | String | | — | ctrl +T |
| browser | Browser.KeyStroke.ResetZoom | Key: Reset zoom | String | | — | ctrl +H |
| browser | Browser.KeyStroke.SaveKit | Key: Save to Kit file | String | | — | ctrl +D |
| browser | Browser.KeyStroke.SwapTransition | Key: Swap transition | String | | — | ctrl +S |
| browser | Browser.KeyStroke.UpgradeState | Key: Upgrade state | String | | — | ctrl +P |
| browser | Browser.LayouterBenchFile | Layouter-Benchmark Filename Prefix | String | | — | LayouterBench_ |
| browser | Browser.LogCompare | Logger Compare Operator | String | | — | >= |
| browser | Browser.LogLevel | Loglevel | Loglevel | | — | 10 |
| browser | Browser.Mnemonic.EditMode | EditMode Mnemonic | String | | — | D |
| browser | Browser.Mnemonic.MacroStep | MacroStep Mnemonic | String | | — | A |
| browser | Browser.Mnemonic.MicroStep | MicroStep Mnemonic | String | | — | I |
| browser | Browser.Mnemonic.Reset | Reset Mnemonic | String | | — | X |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| browser | Browser.Mnemonic.State | State Mnemonic | String | | — | S |
| browser | Browser.Mnemonic.Statechart | Statechart Mnemonic | String | | — | C |
| browser | Browser.Mnemonic.TraceFastForward | TraceFastForward Mnemonic | String | | — | Q |
| browser | Browser.Mnemonic.TracePlay | TracePlay Mnemonic | String | | — | P |
| browser | Browser.Mnemonic.TraceRewind | TraceRewind Mnemonic | String | | — | W |
| browser | Browser.Mnemonic.TraceStepBack | TraceStepBack Mnemonic | String | | — | B |
| browser | Browser.Mnemonic.TraceStepForward | TraceStepForward Mnemonic | String | | — | R |
| browser | Browser.Mnemonic.TraceStop | TraceStop Mnemonic | String | | — | U |
| browser | Browser.Mnemonic.Transition | Transition Mnemonic | String | | — | T |
| browser | Browser.Mnemonic.View | View Mnemonic | String | | — | V |
| browser | Browser.ShowInterface | Show the Interface | Boolean | | — | false |
| browser | Browser.ShowTooltip | Show Tooltips | Boolean | | — | true |
| browser | Browser.SimulatorBenchFile | Simulator-Benchmark Filename Prefix | String | | — | SimBench.txt |
| browser | Browser.Title | Browser Title | String | | — | Kiel Statechart Browser 2.0 |
| browser | BrowserCanvas.StringLabelColor | Color of labels | Color | | Nein | #ff0000 |
| browser | BrowserTables.InputEventColor | Color of input events | Color | | Nein | #0000ff |
| browser | BrowserTables.LocalEventColor | Color of local events | Color | | Nein | #808080 |
| browser | BrowserTables.OutputEventColor | Color of output events | Color | | Nein | #ff0000 |
| browser | BrowserTables.VariablesColor | Color of variables | Color | | Nein | #00ff00 |
| browser | BrowserTree.MarkEdgeColor | Color of marked edge | Color | | Nein | #a8a000 |
| browser | BrowserTree.MarkNodeBrightColor | Color of marked node (bright) | Color | | Nein | #ffffbb |
| browser | BrowserTree.MarkNodeColor | Color of marked node | Color | | Nein | #ffff00 |
| browser | KitEditor.FontSize | Fontsize in embedded kit Editor | int | | Nein | 10 |
| browser | MicroStep.ActualConfigState | CSS layout of actual config state | String | | — | fill :#ddddff;stroke : black ; stroke −width:2.0 px; |
| browser | MicroStep.ExecuteTransition | CSS layout of executed transition | String | | — | fill : none; stroke : green ; stroke −width:2.0px; |
| browser | MicroStep.OnEntry | MicroStep.OnEntry formatting | String | | — | fill : blue ; stroke : black ; stroke −width:2.0px; |
| browser | MicroStep.OnExit | Microstep.OnExit formatting | String | | — | fill : green; stroke : black ; stroke −width:2.0px ; |
| browser | MicroStep.OnInside | Microstep.OnInside formatting | String | | — | fill : gray ; stroke : black ; stroke −width:2.0px; |
| browser | MicroStep.SleepTime | Time between two microsteps in continue mode (in millis) | int | | — | 100 |
| browser | MicroStep.StateActivated | MicroStep.StateActivated formatting | String | | — | fill : none; stroke : black ; stroke −width:2.0px; |
| browser | MicroStep.StateDeactivated | MicroStep.StateDeactivated formatting | String | | — | fill : white ; stroke : black ; stroke −width:2.0px ; |
| browser | MicroStep.StateSuspended | MicroStep.StateSuspended formatting | String | | — | fill : yellow ; stroke : black ; stroke −width:2.0 px; |
| browser | MicroStep.TestTransition | MicroStep.TestTransition formatting | String | | — | fill : none; stroke : blue ; stroke −width:1.0px; |
| browser | ParserThread.Threshold | Update Thread waiting interval in ms | int | | — | 1000 |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| checking | CheckingLogLevel | Checking Loglevel | Loglevel | | — | 10 |
| checking | CorrLogLevel | Correction Log Level | Loglevel | | — | 10 |
| checking | cvclMode | Use JCVCL Library or CVCL Binary | String | bin, lib | — | bin |
| checking | errorColor | Error Color | Color | | Nein | 255,0,0 |
| checking | SemLogLevel | Semantic-Checker Log Level | Loglevel | | — | 10 |
| checking | showClassification | Show Classification Messages | Boolean | | — | true |
| checking | showCounters | Show Counter Messages | Boolean | | — | true |
| checking | showNodeID | Show NodeID Messages | Boolean | | — | true |
| checking | showNodeName | Show NodeName Messages | Boolean | | — | true |
| checking | showPriority | Show Priority Messages | Boolean | | — | true |
| checking | showRuleName | Show RuleName Messages | Boolean | | — | true |
| checking | showTransitions | Show Transistion Messages | Boolean | | — | false |
| checking | SynLogLevel | Syntax-Checker Log Level | Loglevel | | — | 10 |
| checking | timing | Perform time measurement | Boolean | | — | false |
| checking | timingPoststring | Timing Filename Postfix | String | | — | .bench |
| checking | timingPrestring | Timing Filename Prefix | String | | — | SemBench– |
| checking | transitionOverlapHierarchy | Robustness Rule: transistionOverla-pHierarchy | Boolean | | — | true |
| checking | warningColor | Warning color | Color | | Nein | 178,140,0 |
| editor | arcWidthHeight | Height/width of arc | int | | — | 20 |
| editor | cellSelectionTolerance | Cell selection tolerance | int | | — | 10 |
| editor | compositeStateImageOffset | Offset of composite state image | int | | — | 6 |
| editor | defaultNode | Default node | Pair | | — | 20 20 |
| editor | defaultRootState | Default root state | Pair | | — | 800 600 |
| editor | defaultState | Default state | Pair | | — | 55 35 |
| editor | finalStateBorder | Final state border | int | | — | 4 |
| editor | flipArea | Flip area | Quadru-pel | | — | 6 6 10 10 |
| editor | gridColor | Grid color | Color | | — | 170,170,170 |
| editor | gridEnabled | Enable grid | Boolean | | — | true |
| editor | gridSize | Grid size | int | | — | 5 |
| editor | gridVisible | Show grid | Boolean | | — | false |
| editor | guiButtonColorSelectedDark | Color of selected button (dark shade) | Color | | Nein | 255,144,0 |
| editor | guiButtonColorSelectedLight | Color of selected button (light shade) | Color | | Nein | 255,238,217 |
| editor | guiColorDark | Overall GUI color (dark shade) | Color | | Nein | 141,172,238 |
| editor | guiColorLight | Overall GUI color (light shade) | Color | | Nein | 228,247,255 |
| editor | initialGraphOffset | Initial graph offset | int | | — | 5 |
| editor | intersectionSensitivity | Intersection sensivity | int | | — | 2 |
| editor | labelOffset | Label offset | Pair | | — | 10 −5 |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| editor | language | Interface language | String | en | — | en |
| editor | lastFilesCount | Number of last files stored | int | | — | 9 |
| editor | layoutOnInsert | Perform layout on insert | Boolean | | — | true |
| editor | lineEndDiameter | Durchmesser von Linienenden | int | | — | 6 |
| editor | morphingLayouterBorderDistance | Morphing Layouter: Border distance | int | | — | 10 |
| editor | morphingLayouterCheckInterval | Morphing Layouter: Check interval | int | | — | 10 |
| editor | morphingLayouterSpeed | Morphing Layouter: Speed | int | | — | 200 |
| editor | morphingLayouterStepInterval | Morphing Layouter: Step interval | int | | — | 0 |
| editor | overviewColor | Overview color | Color | | — | 210,210,210 |
| editor | placeOfDeepHistory | Place of deep history | Pair | | — | 0 0 |
| editor | portColor | Port color | Color | | — | 141,172,238 |
| editor | pseudoStateNameOffset | Pseudo state name offset | Pair | | — | 7 14 |
| editor | replayStepTime | Replay step-time | int | | — | 600 |
| editor | replayStepTimeFast | Fast replay step-time | int | | — | 200 |
| editor | showAlignmentToolBarOnLoad | Show alignment toolbar on load | Boolean | | — | true |
| editor | showCollapseButtons | Show collapse buttons | Boolean | | — | true |
| editor | showCompositeStateEvents | Show composite state events | Boolean | | — | true |
| editor | showCompositeStateVariables | Show composite state variables | Boolean | | — | true |
| editor | showOverviewLayer | Show overview layer | Boolean | | — | false |
| editor | showReplayToolBarOnLoad | Show replay toolbar on load | Boolean | | — | true |
| editor | showSplashScreen | Show splash screen | Boolean | | — | true |
| editor | showStateActions | Show state actions | Boolean | | — | true |
| editor | sizeHandleColor | Color of size handle | Color | | — | 255,255,255 |
| editor | sizeHandleSensitivity | Sensitivity of size handle | int | | — | 4 |
| editor | sizeHandleSize | Size of size handle | int | | — | 4 |
| editor | sizeHandleVisible | Show size handle | Boolean | | — | true |
| editor | traceLogSeparator | Trace log separator | String | | — | ; |
| editor | validAreaBorder | Valid Area Border | int | | — | 2 |
| editor | zoomInFactor | Zoom in factor | Float | | — | 1.1 |
| editor | zoomOutFactor | Zoom out factor | Float | | — | 0.9 |
| esterel | AutomaticOptimizationOfEsterel-StateCharts | Automatically optimize esterel state charts | Boolean | | Nein | false |
| esterel | cecLinuxPath | Path to cec compiler (linux) | String | | Nein | ../ tools/cec-0.3/linux/bin/ |
| esterel | cecLogfileName | Cec logfile name | String | | Nein | cec.log |
| esterel | CecMacOSPath | Path to cec compiler (MacOS) | String | | Nein | ../ tools/cec-0.3/mac/bin/ |
| esterel | cecWinPath | Path to cec compiler (windows) | String | | Nein | ../ tools/cec-0.3/windows/bin/ |
| esterel | cygWinBinPath | Cygwin bin path | String | | Nein | /usr/bin |
| esterel | cygWinDir | Cygwin win dir | String | | Nein | C:\\\hlinecygwin\\\hlinebin |
| esterel | DebugMode | Run in debug mode | Boolean | | — | false |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| esterel | NumberOfStatesToStartWithOptimization | Number of states to start with optimization | int | | — | 1000 |
| esterel2estudio | ArgumentSeperator | Argument separator | String | | — | , |
| esterel2estudio | Functions | Get Functions (unsused) | Boolean | | — | false |
| esterel2estudio | PostString | New Signal Poststring | String | | — | _ |
| esterel2estudio | PreSting | New Signal Prestring | String | | — | |
| esterel2estudio | Procedure | Get Procedures (unused) | Boolean | | — | false |
| esterel2estudio | Tasks | Get Tasks (unused) | Boolean | | — | false |
| fileInterface | OpenDir | Open File Dialog directory | String | | Nein | ../examples |
| fileInterface | PluginDir | Plugin Search Path | String | | Nein | fileInterface /build/ |
| fileInterface | SaveDir | Save File Dialog Directory | String | | Nein | . |
| fileInterface | ShowLog | Show the Log | Boolean | | Nein | true |
| graphicalInformations | ACTIVITY.FONTFAMILY | The font family for on Entry/on Exit activity | String | | — | SansSerif |
| graphicalInformations | ACTIVITY.FONTSIZE | The font size in pt for on Entry/on Exit activity | int | | — | 8 |
| graphicalInformations | COMPOSITESTATE.DEFAULTSIZE | The default size for composite states | Pair | | — | 80 80 |
| graphicalInformations | DESCRIPTION.FONTFAMILY | The font family for descriptions like state names or the identifier in pseudostates | String | | — | SansSerif |
| graphicalInformations | DESCRIPTION.FONTSIZE | The font size in pt for descriptions like state names or the identifier in pseudostates | int | | — | 14 |
| graphicalInformations | FINALCOMPOSITESTATE.OFFSET | Offset between the two shapes from in final(...)state | int | | — | 5 |
| graphicalInformations | FINALSTATE.OFFSET | Offset between the two shapes from in final(...)state | int | | — | 4 |
| graphicalInformations | HEADER.HEIGHT | The height from the name separator to the state border | int | | — | 20 |
| graphicalInformations | LABEL.FONTFAMILY | The font family for labels | String | | — | SansSerif |
| graphicalInformations | LABEL.FONTSIZE | The font size in pt for labels | int | | — | 11 |
| graphicalInformations | LEFT.OFFSET | The offset from left border of the state to the left start of content area | int | | — | 5 |
| graphicalInformations | LINE.WIDTH | The line with for every drawn element | int | | — | 1 |
| graphicalInformations | LOWER.OFFSET | The offset from end of contentArea to the border of the state | int | | — | 5 |
| graphicalInformations | PSEUDOSTATE.DEFAULT.HEIGHT | The default height for Pseudostates | int | | — | 20 |
| graphicalInformations | PSEUDOSTATE.DEFAULT.WIDTH | The default width for Pseudostates | int | | — | 20 |
| graphicalInformations | RIGHT.OFFSET | The offset from right end of content area to the right border of state | int | | — | 5 |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| graphicalInformations | STATE.DEFAULT.HEIGHT | The default height for states | int | | — | 36 |
| graphicalInformations | STATE.DEFAULT.WIDTH | The default width for states | int | | — | 54 |
| graphicalInformations | UPPER.OFFSET | The offset from separator to start of contentArea | int | | — | 5 |
| kitFileInterface | kriteLayoutInf | | Boolean | | — | false |
| layouter | defaultLayouter | class name of layouter | String | OriginalLayout, DotBinLayouter, CircoBinLayouter, NeatoBinLayouter, TwopiBinLayouter, DotLibLayouter, CircoLibLayouter, NeatoLibLayouter, TwopiLibLayouter | Ja | DotBinLayouter |
| layouter | graphviz.dpi | Resolution | int | | Nein | 72 |
| layouter | graphviz.emptyLabelReplacement.horizontal | Replace empty horizontal label | String | | Nein | |
| layouter | graphviz.emptyLabelReplacement.vertical | Replace empty vertical label | String | | Nein | M |
| layouter | graphviz.finalStateShape | State shape (final state) | String | | Nein | ellipse |
| layouter | graphviz.fontsizeScale | scaling for fontsize used by dot | Float | | Nein | 1.1 |
| layouter | graphviz.initialTransitionWeightGain | Weight gain | int | | Nein | 1 |
| layouter | graphviz.labeledTransitionWeightLoss | Weight loss | int | | Nein | 0 |
| layouter | graphviz.linuxCommand.circo | Start circo command (Linux) | String | | Nein | startCirco.sh |
| layouter | graphviz.linuxCommand.dot | Start dot command (Linux) | String | | Nein | startDot.sh |
| layouter | graphviz.linuxCommand.neato | Start Neato command (Linux) | String | | Nein | startNeato.sh |
| layouter | graphviz.linuxCommand.twopi | Start twopi command (Linux) | String | | Nein | startTwopi.sh |
| layouter | graphviz.mode | Graphviz mode | String | bin, lib, both | — | bin |
| layouter | graphviz.priorityAngle.bt | angle from source port (bt) | int | | Nein | −30 |
| layouter | graphviz.priorityAngle.lr | angle from source port (lr) | int | | Nein | −20 |
| layouter | graphviz.priorityAngle.rl | angle from source port (rl) | int | | Nein | −20 |
| layouter | graphviz.priorityAngle.tb | angle from source port (tb) | int | | Nein | −30 |
| layouter | graphviz.priorityDistance | distance from source port | Float | | Nein | 1.5 |
| layouter | graphviz.pseudoStateShape | State shape (pseudo shape) | String | | Nein | ellipse |
| layouter | graphviz.rankdir.horizontal | special horizontal rankdirection | String | LR, RL | Nein | LR |
| layouter | graphviz.rankdir.vertical | special vertical rankdirection | String | TB, BT | Nein | TB |
| layouter | graphviz.stateShape | State shape (default) | String | | — | box |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| layouter | graphviz.winCommand.circo | Start circo command (Windows) | String | | Nein | circo .exe −Tdot |
| layouter | graphviz.winCommand.dot | Start dot command (Windows) | String | | Nein | dot.exe −Tdot |
| layouter | graphviz.winCommand.neato | Start Neato command (Windows) | String | | Nein | neato.exe −Tdot |
| layouter | graphviz.winCommand.twopi | Start twopi command (Windows) | String | | Nein | twopi.exe −Tdot |
| layouter | hv.horizontalStateStepWidth | Horizontal state step width | int | | Nein | 30 |
| layouter | hv.horizontalTransitionStepWidth | Horizontal transition step width | int | | Nein | 15 |
| layouter | hv.labelBorder | Border width (label) | int | | Nein | 5 |
| layouter | hv.stateAlignment | Alignment | String | top/left, center, bottom/right | Nein | center |
| layouter | hv.stateBorder | Border width (state) | int | | Nein | 15 |
| layouter | hv.verticalStateStepWidth | Vertical state step width | int | | Nein | 30 |
| layouter | hv.verticalTransitionStepWidth | Vertical transition step width | int | | Nein | 15 |
| layouter | layerbased.alternateDirection | Switch layout direction for each layer | Boolean | | Nein | true |
| layouter | layerbased.initialDirection | Initial direction | String | v, h, auto | Nein | h |
| layouter | layerbased.layerAlignment.horizontal | Horizontal Layer alignment: [l]eft [c]enter [r]ight | String | l, c, r | Nein | c |
| layouter | layerbased.layerAlignment.vertical | Vertical layer alignment: [t]op [c]enter [b]ottom | String | t, c, b | Nein | c |
| layouter | logLevel | the log level specified by kiel.util. LogFile | Loglevel | | Nein | 10 |
| layouter | semanticZoom | semantic zoom | Boolean | | Nein | true |
| matlabStateflow | doBenchmark | Perform benchmark | Boolean | | — | false |
| matlabStateflow | matlabErrorIndicator | Matlab error indicator | String | | — | ??? |
| matlabStateflow | MatlabProcessInfo.matlabExecutableFileName | Matlab program file name | String | | — | matlab−class |
| matlabStateflow | MatlabProcessInfo.nodisplay | Don't display matlab processinfo | Boolean | | — | false |
| matlabStateflow | matlabPrompt | Matlab prompt | String | | — | ›› |
| matlabStateflow | MatlabStateflowGrabber.chartScale | Chart scale of matlab stateflow grabber | Float | | — | 1.0 |
| matlabStateflow | StateflowSimulator.inputVariableNamePrefix | Prefix of input variables from stateflow simulator | String | | — | VarMat |
| optimizer | DebugMode | Run Optimizer in Debug mode | Boolean | | — | false |
| optimizer | OptimizeESTEliminateNeedlessConditional | Activate 'EST Eliminate Needless Conditional' optimization | Boolean | | — | true |
| optimizer | OptimizeESTEliminateNeedlessNormalTransitions | Activate 'EST Eliminate Needless Normal Transitions' optimization | Boolean | | — | true |
| optimizer | OptimizeESTEliminateNeedlessSimpleStates | Activate 'EST Eliminate Needless Simple States' optimization | Boolean | | — | true |

| Modul | Property | Beschreibung | Typ | Alternativen | Neust. | Vorgabewert |
|---|---|---|---|---|---|---|
| optimizer | OptimizeESTEliminateNotAbortednWithoutLocalsOrStates | Activate 'EST Eliminate Not Abortedn Without Locals Or States' optimization | Boolean | | — | true |
| optimizer | OptimizeESTJoinFinalStates | Activate 'EST Join Final States' optimization | Boolean | | — | true |
| optimizer | OptimizeESTOrStatesWithTwoSubstates | Activate 'EST Or States With Two Substates' optimization | Boolean | | — | true |
| optimizer | OptimizeESTUpdateFinalStates | Activate 'EST Update Final States' optimization | Boolean | | — | true |
| optimizer | OptimizeImmediate | Optimize immediately | Boolean | | — | false |
| preferences | loglevel | Loglevel | Loglevel | | Nein | 10 |
| robustness | robustness.cvclMode | cvcl mode | String | | Nein | lib |
| robustness | robustness.errorColor | Error Color | Color | | Nein | 255,0,0 |
| robustness | robustness.RobustnessLogLevel | Robustness Loglevel | Loglevel | | Nein | 10 |
| robustness | robustness.SemLogLevel | Semantic-Checker Log Level | Loglevel | | Nein | 10 |
| robustness | robustness.SynLogLevel | Syntax-Checker Log Level | Loglevel | | Nein | 10 |
| robustness | robustness.warningColor | Warning color | Color | | Nein | 178,140,0 |

# B Implementierungshinweise für Autoren neuer KIEL-Module

Damit ein neues KIEL-Modul mit der XMLPreferences-Klasse zusammenarbeiten kann, sind folgende Punkte zu beachten:

1. Es muss für die Einstellungen des Moduls ein XML-Schema erstellt werden, welches den gültigen Aufbau der Modul-Konfigurationsdateien beschreibt. Das Schema muss darüber hinaus die nötigen Metainformationen für das JAXB-Binding und den Konfigurationsdialog enthalten.
2. Es muss eine XML-Konfigurationsdatei nach dem Schema aus Punkt 1 erstellt werden, welche die Vorgabewerte der Einstellungen enthält.
3. Das Build-Skript des Moduls muss Eintragungen zur Erstellung des JAXB-Bindings enthalten.
4. Die Property-Klasse des Moduls muss, wenn KIEL startet, ein XMLPreferences-Objekt und das Data Binding instantiieren.

## B.1 Erstellung der XML-Dateien

Die Konfigurationsdateien (Schema und Vorgabewerte) können mit dem Skript `kiel/documents/papers/khe/propertylist/generate_properties.sh` automatisch erstellt werden. Das Skript verwendet die beigefügte *elementree*-Bibliothek zum verarbeiten von XML-Daten in Python, des Weiteren wird das Programm `xmlstarlet`[1] benötigt. Es führt nacheinander folgende Aktionen aus:

1. Erzeugen der XML-Dateien aus der CSV-Datei mit Python/Elementree
2. Reformatieren der XML-Dateien mit xmlstarlet
3. Valieren der XML-Default-Dateien gegen die XML-Schemata mit `xmlstarlet`

Das Skript nimmt als Parameter den Namen einer *Comma Separated Value* (CSV)-Datei, welche alle die Einstellungsoptionen des Moduls betreffen Eintragungen enthält. Das Format der Datei besteht aus fünf mit Tabulator getrennten Spalten mit folgender Bedeutung:

[1] Debian-Paket: `xmlstarlet`

1. Name des Moduls mit klein geschriebenem Anfangsbuchstaben (z. B. `browser`)
2. Name der Einstellung (z. B. `timingPoststring`)
3. Vorgabewert (z. B. `.bench`)
4. Datentyp (`string`, `int`, `float`, `boolean`, `color`, `loglevel`, `pair` oder `quadrupel`)
5. Beschreibung der Einstellung für den Konfigurationsdialog (in Englisch)

Beispiel:

```
"Module"→"Property Id"↪"Default value"↪"Property Type"↪"Description"
"checking"→"timingPoststring"→".bench"→"String"→"Timing Filename Postfix"
```

**Hinweis:** Die erste Zeile der CSV-Datei ist für die Spaltenüberschrift gedacht und wird von dem Skript ignoriert

Das Skript erzeugt dann für jedes Modul die XML-Dateien `<modulname>-schema.xsd` und `<modulname>-defaults.xml`. Diese müssen in das Verzeichnis kopiert werden, in dem auch die Basisklasse des Moduls liegt, beispielsweise in `kiel/kiel/browser/src/kiel/browser`.

Verwendet das Modul neue Datentypen, so müssen diese in das Python-Skript `csv2xml.py` eingetragen werden, dabei helfen die im Skript enthaltenen Kommentare.

## B.2 Anpassung des Build-Skriptes

Jedes KIEL-Modul verfügt über ein build-Skript `build.xml`. Dieses muss die in Listing B.1 aufgeführten Eintragungen enthalten. Eine Vorlage für eine solche Datei findet sich unter `kiel/documents/papers/khe/moduletemplate/build.xml.tmpl`. Aus dieser kann mit

```
$ sed -e 's:@@MODULE@@:ModuleName:g' -e 's:@@module@@:moduleName:g' < build.↩
    xml.tmpl > build.xml
```

in die tatsächliche `build.xml`-Datei konvertiert werden.

**Listing B.1:** Notwendige Eintragungen in der `build.xml`-Datei eines KIEL-Moduls

```
<project ...>
  ...
  <property name="jaxb.dir" value="../../tools/jaxb-1.0.5" />
  <taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
    <classpath>
      <fileset dir="${jaxb.dir}" includes="*.jar" />
    </classpath>
```

```
⟨/taskdef⟩
...
⟨target name="init"⟩
  ...
  ⟨path id="module.class.path"⟩
    ...
    ⟨fileset dir="${jaxb.dir}"⟩
      ⟨include name="*.jar" /⟩
    ⟨/fileset⟩
  ⟨/path⟩
  ⟨property name="xml.schema" value="src/kiel/⟨Modul⟩/⟨Modul⟩-schema.xsd" /⟩
  ⟨property name="xml.gensrcdir" value="src/kiel/⟨Modul⟩/xmlconf" /⟩
⟨/target⟩
...
⟨!--  generate xml bindings --⟩
⟨target name="xmlbindings" depends="init"⟩
  ⟨echo message="Generating␣XML␣bindings" /⟩
  ⟨xjc schema="${xml.schema}" package="kiel.⟨Modul⟩.xmlconf" target="${
      module.src.dir}"⟩
    ⟨produces dir="${xml.gensrcdir}" includes="**/*.java" /⟩
  ⟨/xjc⟩
⟨/target⟩
...
⟨!-- compile module --⟩
⟨target name="compile" depends="prepare,␣actiontest,␣actiongenerate,␣
    xmlbindings"⟩
  ...
  ⟨copy todir="${module.build.dir}"⟩
    ⟨fileset dir="${module.src.dir}"⟩
       ⟨include name="**/*.properties" /⟩
       ⟨include name="**/*.xsd" /⟩
       ⟨include name="**/*.xml" /⟩
       ⟨include name="**/*.ser" /⟩
     ⟨/fileset⟩
   ⟨/copy⟩
   ...
⟨/target⟩
...
⟨!-- clean build directory --⟩
⟨target name="clean" depends="init,␣sableremove"⟩
  ⟨delete includeEmptyDirs="true" failonerror="false"⟩
    ...
    ⟨fileset dir="${xml.gensrcdir}"/⟩
  ⟨/delete⟩
⟨/target⟩
...
```

```
⟨/project⟩
```

## B.3 Die Property-Klasse des Moduls

KIEL-Module enthalten üblicherweise eine Klasse `<ModulName>Properties.java`, welche des Zugriff auf die Einstellungen mit `get-` und `set`-Methoden kapselt. Eine Vorlage für eine solche Datei findet sich unter `kiel/documents/papers/khe/moduletemplate/ModulenameProperties.java.tmpl`. Aus dieser kann mit

```
$ sed -e 's:@@MODULE@@:ModuleName:g' -e 's:@@module@@:moduleName:g' ⟨ ↩
    ModulenameProperties.java.tmpl ⟩ ModulenameProperties.java
```

in die tatsächliche Java-Datei konvertiert werden. Um auf die XML-Einstellungen zugreifen zu können, muss diese Klassen folgenden Code enthalten:

**Listing B.2:** Gerüst einer Property-Klasse

```
package kiel.⟨ModulName⟩;

...

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import org.w3c.dom.Document;

import kiel.preferences.XMLPreferences;
import kiel.⟨ModulName⟩.xmlconf.Configuration;

...

public final class ⟨ModulName⟩Properties {

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

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

    /**
     * The Configuration object, unmarshalled from XML.
```

```
     */
    private static Configuration conf;

    static {
        xmlprefs = XMLPreferences.createInstance("⟨modulName⟩",
                ⟨ModulName⟩Properties.class);

        logger = xmlprefs.getLogger();

        try {
            jc = JAXBContext.newInstance("kiel.⟨modulName⟩.xmlconf",
                                         ⟨ModulName⟩Properties.class.↩
                                             getClassLoader());
        } catch (JAXBException je) {
            je.printStackTrace();
        }

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

        Observer ⟨ModulName⟩Observer = new Observer() {
            public void update(Observable o, Object arg) {
                if (arg.equals(XMLPreferences.PREFERENCES_REMARSHAL)) {
                    XMLPreferences.log(LogFile.DEBUG,
                            "Reloading␣⟨ModulName⟩␣Properties");
                    ⟨ModulName⟩Properties.reload();
                }
            }
        };
        xmlprefs.addObserver(⟨ModulName⟩Observer);
    }

    /**
     * reload preferences.
     * @return success
     */
    public static boolean reload() {
        try {
            XMLPreferences.log(LogFile.INFO, "Reloading␣⟨ModulName⟩");
            XMLPreferences.log(LogFile.INFO, "Schema:␣" + xmlprefs.↩
                getXMLSchemaURL());
            XMLPreferences.log(LogFile.INFO, "Schema:␣" + xmlprefs.↩
                getXMLPreferencesURL());
            Unmarshaller u = jc.createUnmarshaller();
```

```
            Document doc = xmlprefs.getPreferences();
            //XMLPreferences.log(LogFile.INFO, "XML Doc: " + doc);
            conf = (Configuration) u.unmarshal(doc);

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

    /**
     * Not for use.
     */
    private ⟨ModulName⟩Properties() {
    }

    ...

}
```

# C Quellcode

## C.1 Handbuch und Quellcode von KielTEX

Die folgenden Seiten enthalten das Handbuch und den (kommentierten) Quellcode von KielTEX in dem bei der Entwicklung von LATEX-Paketen üblichen `dtx`-Format.

# The KielTEX package

Karsten Heymann

v0.4 (2007/05/14)

## 1 Introduction

This document describes the KielTEX package. The KielTEX package can be used to include Statecharts that are generated with the Kiel Integrated Environment for Layout (KIEL)[1].

## 2 Installation

### 2.1 Dependencies

KielTeX has some requirements which have to be met:

- KielTEX depends on pdfTEX 1.30 or newer, which is standard on all current TEX distributions. Both `dvi`- and `pdf`-output are supported.
- The following LaTeX packages are required (⋆: not included in TEXlive 2007):
  - graphicx
  - moreverb
  - xkeyval
  - kvsetkeys (⋆)
  - ifpdf
  - ifdraft
- The following external programs are called if shell escape is enabled:
  - `kielcmd`, the command line interface to the Kiel Integrated Environment for Layout. For instructions how to install KIEL, refer to the KIEL Project Manual (`http://rtsys.informatik.uni-kiel.de/~rt-kiel/kiel/documents/project-management/`
  - `ps2eps`[2]
  - `epstopdf`, part of TeXlive

[1] `http://www.informatik.uni-kiel.de/~rt-kiel/`
[2] `http://www.tm.uka.de/~bless/ps2eps`

### 2.2 Installing KielTEX

Put `kieltex.sty`, `kieltex-kit.sty` and `kieltex-esterel.sty` in a directory where LaTeX can find it, usually something like `texmf/tex/latex/kieltex`. Don't forget to refresh the TeX filename database afterwards (on Unix/Linux, run `texhash`)!

### 2.3 Compiling this manual

In case you want to regenerate this manual from the KielTeX source (`kieltex.dtx`), run:

- `pdflatex kieltex.dtx`
- `makeindex -s gglo.ist -o kieltex.gls kieltex.glo`
- `makeindex -s gind.ist -o kieltex.ind kieltex.idx`
- `pdflatex kieltex.dtx`
- `pdflatex kieltex.dtx`

## 3 Usage of KielTeX

### 3.1 Loading the package

KielTeX is loaded with

```
\usepackage[\meta{Options}]{kieltex}
```

Possible Options are

**shell** This option is set by default. If set, KielTeX will try to launch all external programs needed to create a statechart image from a statechart file. Depending on your setup, this may require you to reconfigure your TeX system (not recommended) or to pass a command line switch (usually `-shell-escape`) to your LaTeX interpreter.

**noshell** Deactivates the behaviour of the `shell` option, even if calling of external applications is allowed.

**optimizeeps** This option is set by default. If set, generated images are postprocessed by the `ps2eps` program, mainly to create a correct bounding box. Be aware that this takes additional time, especially when processing very large statecharts.

**nooptimizeeps** Deactivates the behaviour of the `optimizeeps` option.

**generatepdf** This option is set by default when running `pdflatex` in pdf mode. If set, the `eps` images created by `kielcmd` are automatically convertet do `pdf` files with the `epstopdf` program.

**nogeneratepdf** As before, deactivates the behaviour of the `generatepdf` option.

**cache** This option is set by default. If set, the caching feature of `kielcmd` is activated. This setting is highly recommended, as regenerating each and every statechart image on every TeX run can take up an insane amount of time!

**nocache** Deactivates caching. Each Statechart image is regenerated every time (as long as calling external programs is allowed and not disabled by the `noshell` option). The cache will not be looked at at all.

**recache** Sometimes wrong or defect image may happen to enter the cache. The `recache` option forces a regeneration of all statechart images in your document, whichâĂŤin contrast to the nocache optionâĂŤwill end up in the cache.

# 4 Using statecharts

KielTeX provides two ways to include statecharts in your document: You can either include the statechart source in the document, or you can reference external statechart files, as you would do with i.e. pictures. Of course both ways can be mixed in one and the same document!

To include statechart code directly in your document, you can use the `KIELstatechart` environment.

```
\begin{KIELstatechart}[\meta{Options}]
<Statechart-Code>
\end{KIELstatechart}
```

To reference an external statechart file, use

```
\includeKIELstatechart[\meta{Options}]{\meta{Statechart-File}}
```

Both commands understand the same set of options, which can also be preset with

```
\KIELsetup{\meta{Options}}
```

A note on how embedded statecharts work: The content of the `KIELstatechart` environment is written to a temporary file at runtime (similiar to the `filecontents`-Environment) and externally processed by commands called from LaTeX. In the end, this results in an image file, which then is included into the document. Statechart referenced via `\includeKIELstatechart` are processed identically.

The possible options are subject of the next section.

# 5 Setting options

The statecharts generated by KielTeX can be customized via options. Options that are set by `\KIELsetup` are set for the rest of the document (unless changed again later), whereas options set by `\includeKIELstatechart` or at the `KIELstatechart` environment are set only for the respective statechart.

Options understood by KielTeX are:

**language**=⟨***dialect***⟩ The format of the (textual representation of) the statechart. This option is used for two purposes: For embedded statecharts, it is used to set the filename extension of the gererated temporary statechart file. This is important as KielCmd recognizes statechart dialects by their extensions. Aditionally, if kieltex is configured to produce a listing of the statechart code instead of or additionally to the statecharts image (see the `shwo` option below), the `language` option selects the right syntax highlighting of the file.

At the time of writing, KIEL (and thus KielCmd) is able to process the following statechart dialects:

- Esterel
- EsterelStudio
- KIT
- KIEL

**filetype**=⟨***extension***⟩ Allows to manually specify the extension of the temporary statechart file generated from the content of a `KIELstatechart` environment. Usually the extension is set by the `language` option, but it is restricted to statechart dialects known to KielTeX. The `filetype` option exists to add additional formats without the need to change the KielTeX package (although loosing the ability to select a matching source highlighting for these statecharts automatically (they can be selected nevertheless by the `\lstset` command).

**layouter**=⟨***Layouter***⟩ Controls, which of KIELs autolayouters is used to format the statechart. A list of available layouters is produced by typing

```
kielcmd -list
```

on the command line. At the time of writing the following layouters are available:

- CircoLayouter
- DotLayouter
- HVLayouter
- NeatoLayouter
- OriginalLayout
- TwopiLayouter

**config**=**{StateA StateB ...}** Allows to specify a configuration by its active states. States are referenced by their name, which must be unique in the statechart. They are separated by a blank. Active states are colored by default (cf. `highlight` option).

**highlight**=⟨***true/false***⟩ Enables or disables the coloring of the active states in a statechart configuration.

**focus={[fromNode]:[toNode]:[auto/⟨*priority*⟩]}** With the `focus` option, it is possible to mimick the state selection mechanism used in KIELs browser module for keyboard based editing of statechart. It is possible to select up to two states (by their names, which have to be unique), and optionally a transition between the two states if one exists. In case there are more two or more transitions connecting the two selected states, and they differ in their priorities, a specific transistion can be selected by specifiing its priority. Otherwise, if the keyword `auto` is given instead of a priority, the transition with the highest priority is selected.

**optimize**=⟨***false/true***⟩ Enables or disables the automatic optimization of the statechart.

**show**=⟨***chart/source/split/none***⟩ This option controls the visual layout of the statechart done by KielTeX. Its main purpose is to allow the display of the statechart image and the corresponding statechart source side by side. The default value `chart` causes only the statechart image to be displayed, whereas `source` only displays the source code of the statechart. The `split` option causes the statechart image and the statechart code to be displayed side by side. The splitting can be further customized by the following options:

**chartwidth**=⟨***length***⟩

**sourcewidth**=⟨***length***⟩

**totalwidth**=⟨***length***⟩ These options control the layout of statecharts displayed with the `show=split` option.

**gfxopts={graphicx-options}** This allows to pass options from the `graphicx` package to the `includegraphics` command used by KielTeX to display the statechart images. Possible graphicx options are for example: `height`, `width`, `scale`, `keepaspectratio`, or `angle`.

**lstopts={listings-options}** This options allows it to pass settigns to the lstlistings-commands used to typeset statechart listings.

**kielopts={kiel options}** With this options, it is possible to set KIEL configuration properties, as with the `-option` parameter of KielCmd. Example:

```
\KIELsetup{
    kielopts={layouter.graphviz.dpi=100,
        layouter.graphviz.rankdir.horizontal=LR,
   }
}
```

Be aware that there are two limitations for allowed option values at the moment:

- Options containing a `=` or `,` sign must be enclosed in curly braces: `kielopts={option1={a,b,c}}`.
- The `#` sign is not allowed in KIEL options at all! This affects especially the notation of color values using the `#rrggbb` syntax. To change color settings, you have to use the alternative `{rrr,ggg,bbb}` nomenclature instead:
  `kielopts={some.color.property={100,200,75}}`

# 6 The code

## 6.1 Initialization

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}[1999/12/01]
3 \ProvidesPackage{kieltex}
4     [2007/05/14 v0.4 KIEL statecharts in LaTeX]
```

Load required packages.

```
5 \RequirePackage{graphicx,moreverb,xkeyval,listings,ifpdf}
```

The `kvsetkeys` provides a `\kvsetkeys` macro which behaves like `\setkeys` from the (x)keyval package but adds additional hooks to i. e. react on undefined keys.

```
6 \RequirePackage{kvsetkeys}[2006/10/19]
```

KielTEX ships with its own syntax definitions for the `esterel` and `kit` languages to allow syntax highlighted source listings for those statechart dialects. As the syntax definitions included in KielTEX are quite young and might be completed locally, it is first checked if `esterel.sty` and/or `kit.sty` are available, which are supposed to contain extended local definitions of those languages. Languages with no local definitions use the supplied `kieltex-esterel.sty` and/or `kieltex-kit.sty` files.

```
7 \IfFileExists{esterel.sty}{%
8   \RequirePackage{esterel}%
9 }{%
10   \RequirePackage{kieltex-esterel}}%
11 \IfFileExists{kit.sty}{%
12   \RequirePackage{kit}%
13 }{%
14   \RequirePackage{kieltex-kit}}%
```

## 6.2 Global variables

Names of external applications. These programs must reside in PATH.

```
15 \def\KIEL@app@kielcmd@cache{kielcmd -c}%
16 \def\KIEL@app@kielcmd@nocache{kielcmd}%
17 \def\KIEL@app@kielcmd@recache{kielcmd -cr}%
18 \def\KIEL@app@epstopdf{epstopdf}%
19 \def\KIEL@app@pstoeps{ps2eps -f}%
```

Allocate a global `\write` to use for all filesystem access.

```
20 \newwrite\KIEL@write
```

## 6.3 Package Options

`shell` `shell`: try to launch converters via `\write18`.

```
21 \newif\ifKIEL@ShellEscape
22 \DeclareOption{shell}{\KIEL@ShellEscapetrue}
23 \DeclareOption{noshell}{\KIEL@ShellEscapefalse}
```

`optimizeeps` `optimizeeps`: if set, optimize eps by running through `ps2eps`. Enables shell escape.

```
24 \newif\ifKIEL@OptimizeEPS
25 \DeclareOption{optimizeeps}{\KIEL@OptimizeEPStrue\KIEL@ShellEscapetrue}
26 \DeclareOption{nooptimizeeps}{\KIEL@OptimizeEPSfalse}
```

generatepdf

generatepdf: if set, optimize eps by running through epstopdf. Enables shell escape.

```
\newif\ifKIEL@Generatepdf
\DeclareOption{generatepdf}{\KIEL@Generatepdftrue\KIEL@ShellEscapetrue}
\DeclareOption{nogeneratepdf}{\KIEL@Generatepdffalse}
```

cache
nocache
recache

cache: enable cachting (default). nocache: ignore kiel image cache completely. recache: force regeneration of images and put them into the cache.

```
\DeclareOption{cache}{\let\KIEL@app@kielcmd=\KIEL@app@kielcmd@cache}
\DeclareOption{nocache}{\let\KIEL@app@kielcmd=\KIEL@app@kielcmd@nocache}
\DeclareOption{recache}{\let\KIEL@app@kielcmd=\KIEL@app@kielcmd@recache}
```

Set default options and execute option parser.

```
\DeclareOption{draft}{\PassOptionsToPackage{\CurrentOption}{ifdraft}}
\DeclareOption{final}{\PassOptionsToPackage{\CurrentOption}{ifdraft}}
```

Set default options and execute option parser.

```
\ExecuteOptions{shell,optimizeeps,cache}
\ifpdf\ExecuteOptions{generatepdf}\fi
\DeclareOption*{\PackageWarning{kieltex}{`\CurrentOption' ignored}}
\ProcessOptions\relax
```

The package ifdraft must be loaded after option processing due to \PassOptionToPackage.

```
\RequirePackage{ifdraft}
```

Warn if generatepdf-Option is set but we don't run pdftex in pdf mode.

```
\ifKIEL@Generatepdf
  \ifpdf % then nothing
  \else
    \KIEL@Generatepdffalse
    \PackageWarningNoline{kieltex}%
   {%
     Option `generatepdf' was given but pdftex is not in pdf mode.\MessageBreak
     Option was disabled.
   }%
  \fi
\fi
```

\KIELsetup

Options for KielTeX can be set via \KIELsetup:

```
\newcommand{\KIELsetup}{%
  \kvsetkeys{KIEL}%
}
```

## 6.4 Test whether \write18 is enabled

Test whether shell escape is enabled. Depends on running pdftex.

```
\begingroup
\@ifundefined{pdfshellescape}
{%
  \@latex@warning@no@line
  {%
    Status of \string\write18-feature cannot be detected
    automatically!\MessageBreak
    pdfTeX 1.30 or higher is needed for the check via%
    \MessageBreak\string\pdfshellescape.\MessageBreak
    It is assumed that \string\write18\space is disabled%
```

```
  }%
  \KIEL@ShellEscapefalse
}%
%
\ifKIEL@ShellEscape
{%
  \ifnum\pdfshellescape>0\else
    \@latex@warning@no@line
    {Option ‘shell’ given but \string\write18-feature is not enabled.\MessageBreak
      Try command line option -shell-escape (teTeX/texlive)
      or \MessageBreak -enable-write18 (mikTeX).\MessageBreak%
      External programs will not be run
    }%
    \KIEL@ShellEscapefalse
  \fi
}
\fi
\endgroup
%   \end{macrocode}
% If shell escaping is disabled notify the user that he will have to convert
% extracted code snippets himself.
%    \begin{macrocode}
\ifKIEL@ShellEscape
    \PackageInfo{kieltex}%
    {Automatically converting via kielcmd}%
\else
    \PackageWarningNoLine{kieltex}%
        {Shell escape not enabled.\MessageBreak
        You’ll need to run kielcmd manually}%
\fi
```

## 6.5 Statechart numbering

KIEL@figcount

Every statechart in a document has an internal number. The number is used to construct the names of temporary statechart and job files.

```
\newcounter{KIEL@figcount}
\def\KIEL@figname{\jobname-fig\theKIEL@figcount}
```

## 6.6 String comparison

LaTeX does not offer a simple way to compare two strings by default. The following code by David Kastrup (msgid:85ekbc34jj.fsf@lola.goethe.zz in Usenet) provides a string comparison method.

```
\def\KIEL@strequal#1{\number\KIEL@strequalstart{}{}#1\relax}
\def\KIEL@strequalstart#1#2#3{\if#3\relax\KIEL@strequalstop\fi
  \KIEL@strequalstart{\if#3#1}{#2\fi}}
\def\KIEL@strequalstop\fi\KIEL@strequalstart#1#2#3{\fi#1#3\relax’#213 }
```

## 6.7 Write verbatim environments to a file

The following code is modeled after the examples in moververb.sty.

```
\def\KIEL@verbatimwrite#1{%
```

```
    \@bsphack
    \immediate\openout \verbatim@out #1%
    \KIEL@BeforeStream%
    \let\do\@makeother\dospecials
    \catcode`\^^M\active
    \def\verbatim@processline{%
        \immediate\write\verbatim@out
        {\the\verbatim@line}}%
    \verbatim@start}
\def\endkielverbatimwrite{%
    \immediate\closeout\verbatim@out
    \@esphack}
```

Maybe we want to add some text before the extracted code in the temporary file. (At the moment we don't).

```
\def\KIEL@BeforeStream
{\message{Opening Kiel stream=\KIEL@figname.\KIEL@filetype}%
    %\immediate\write\verbatim@out{module \KIEL@figname:}
}
```

## 6.8 KielTEX command parameters

The property handling is done by `kvsetkeys`. Kiel settings are stored in two keyval families, KIEL and KIELPROPS. The KIEL family is what the user interfaces with, and the Unknown keys are handled as KIEL properties and written to the job file. The following code was inspired by Heiko Oberdiek (`Message-ID: <eshl6p`$fps$`1@news.BelWue.DE>`).

```
\kv@set@family@handler{KIELPROPS}{%
  \ifx\kv@value\relax
    \@latex@warning{Key `#1' does not have a value}%
  \else
    \@ifundefined{KIEL@#1}{%
      \edef\KIEL@opts{%
        \unexpanded\expandafter{\KIEL@opts}%
        \ifx\KIEL@opts\@empty
        \else
          ^^J%
        \fi
        -option=#1=\expandafter\noexpand\csname KIEL@#1\endcsname
      }%
    }{}%
    \expandafter\def\csname KIEL@#1\endcsname{#2}%
  \fi
}
```

\KIELreset `\KIELreset` can be used to clear all set KIEL options.

```
\newcommand*{\KIELreset}{%
  \let\KIEL@opts\@empty
}
```

Whether to show the source and/or chart. Default to chart.

```
\newif\ifKIEL@showsource
\newif\ifKIEL@showchart
\newif\ifKIEL@splitview
```

```
\g@addto@macro\KIELreset{%
  \KIEL@showsourcefalse
  \KIEL@showcharttrue
  \KIEL@splitviewfalse
}
```

`kielopts` The option `kielopts` is used to set KIEL properties. Parameters are forwarded to the `KIELPROPS` setting family.

```
\define@key{KIEL}{kielopts}{\kvsetkeys{KIELPROPS}{#1}}%
```

`gfxopts` The option `gfxopts` is used to set KIEL properties. Parameters are forwarded to the `Gin` setting family.

```
\define@key{KIEL}{gfxopts}{\setkeys{Gin}{#1}}%
```

`srcopts` The option `srcopts` is used to set KIEL properties. Parameters are forwarded to the `listings` package.

```
\define@key{KIEL}{lstopts}{\lstset{#1}}%
```

`language` Other options are for KielTeX itself. The `language` option instructs KielTEX about the dialect used to describe the statechart. This determines the file extension of the temporary statechart file (by which `kielcmd` detects the statechart type, and the syntax definition used to highlight keywords in statechart source listings.

```
\define@choicekey{KIEL}{language}[\val\nr]{Esterel,kit,EsterelStudio,Kiel}{%
  \lstset{language={}}%
  \ifcase\nr\relax
    \KIELsetup{filetype=strl}%
    \lstset{language={#1}}%
  \or
    \KIELsetup{filetype=kit}%
    \lstset{language={#1}}%
  \or
    \KIELsetup{filetype=scg}%
  \or
    \KIELsetup{filetype=kiel}%
  \fi
}%
\g@addto@macro\KIELreset{%
  \let\KIEL@language\@empty
}
```

`filetype` The `filetype` option should be seldomly be used directly. It controls the file ending of the temporary statechart file. One possible use is when a new statechart dialect is added to KIEL, for which no `language` option exists. Then the `filetype` option could be used to explicitely specify the extension for statecharts in this dialect.

```
\define@cmdkeys{KIEL}[KIEL@]{filetype}[\@empty]%
\g@addto@macro\KIELreset{%
  \let\KIEL@filetype\@empty
}
```

`layouter` `config` `focus` The `layouter`-, `config`- and `focus`-options add `kielcmd` command line arguments.

```
\define@cmdkeys{KIEL}[KIEL@]{layouter,config,focus}[\@empty]%
\g@addto@macro\KIELreset{%
  \let\KIEL@layouter\@empty
```

```
  \let\KIEL@config\@empty
  \let\KIEL@focus\@empty
}
```

highlight
optimize

The options `highlight` and `optimize` are boolean keys. They are not interpreted by KielTEX but cause specific command line options for `kielcmd` to be set.

```
\define@boolkeys{KIEL}{highlight,optimize}[true]%
\g@addto@macro\KIELreset{%
  \KV@KIEL@highlighttrue
  \KV@KIEL@optimizefalse
}
```

show

The `show` option controls the way KielTEX actually displays the statechart.

```
\define@choicekey{KIEL}{show}[\val\nr]%
    {chart,source,split}{
  \xdef\KIEL@layoutmode{\nr}%
}%
\g@addto@macro\KIELreset{%
  \KIELsetup{show=chart}%
}
```

layoutcmd

The `layoutcmd` option lets the user provide a custom method to align and arrange statechart picture and statechart sourcecode. The option takes a function that may not take arguments. The function can call the methods `\KIEL@displaychart` and/or `\KIEL@displaysource` to place the actual graphic/source. Keep in mind that, if a `layoutcmd` is set, the `layout`-option is ignored.

Example:

```
\makeatletter
\newcommand\mylayout{%
   \begin{minipage}{3cm}[t]
     \textbf{Statechart:}\par
     \vfill
     \KIEL@displaychart
   \end{minipage}
   \begin{minipage}{10cm}[t]%
     \textbf{Statechart source:}\par
     \KIEL@displaysource
   \end{minipage}%
 }
\makeatother
\KIELsetup{layoutcmd=\mylayout}
```

```
\define@cmdkey{KIEL}[KIEL@]{layoutcmd}[\@empty]{}
\g@addto@macro\KIELreset{%
  \let\KIEL@layoutcmd\@empty
}
```

label

The `label` option allows reusing a already generated image. To allow this, if a statecharts has a label, the file names of the corresponding source and graphic files are stored in macros named `\KIEL@{img,src}@\meta{label}`.

```
\define@key{KIEL}{label}{
  \expandafter\xdef\csname KIEL@img@#1\endcsname{\KIEL@figname}%
  \expandafter\xdef\csname KIEL@src@#1\endcsname{\KIEL@CutFile}}
```

sourcewidth
chartwidth
totalwidth

```
\newlength{\KIEL@chartwidth}%
\newlength{\KIEL@sourcewidth}%
\newlength{\KIEL@totalwidth}%
\define@key{KIEL}{chartwidth}{\setlength{\KIEL@chartwidth}{#1}}
\define@key{KIEL}{sourcewidth}{\setlength{\KIEL@sourcewidth}{#1}}
\define@key{KIEL}{totalwidth}{\setlength{\KIEL@totalwidth}{#1}}
```

`\KIELreset` Initialize all options to their defaults by calling `\KIELreset`.

```
\KIELreset
```

## 6.9 Layouting the statechart

`\KIEL@calculatewidths` Depending on which of `totalwidth`, `chartwidth` and `sourcewidth` are set, the missing ones are computed.

```
\newcommand{\KIEL@calculatewidths}{%
```

Step 1: If the total width is unset, calculate or set it.

```
  \ifdim\KIEL@totalwidth=\z@
    \ifdim\KIEL@sourcewidth=\z@
      \setlength{\KIEL@totalwidth}{\linewidth}%
    \else
      \ifdim\KIEL@chartwidth=\z@
        \setlength{\KIEL@totalwidth}{\linewidth}%
      \else
        \setlength{\KIEL@totalwidth}{\KIEL@chartwidth}%
        \addtolength{\KIEL@totalwidth}{\KIEL@sourcewidth}%
      \fi
    \fi
  \fi
```

Step 2: If chart width or source with are unknown, calculate them.

```
  \ifdim\KIEL@sourcewidth=\z@
    \ifdim\KIEL@chartwidth=\z@
      \setlength{\KIEL@chartwidth}{0.5\KIEL@totalwidth}%
      \setlength{\KIEL@sourcewidth}{0.5\KIEL@totalwidth}%
    \else %chart set, source unset
      \setlength{\KIEL@sourcewidth}{\KIEL@totalwidth}%
      \addtolength{\KIEL@sourcewidth}{-\KIEL@chartwidth}%
    \fi
  \else % source set
    \ifdim\KIEL@chartwidth=\z@
      \setlength{\KIEL@chartwidth}{\KIEL@totalwidth}%
      \addtolength{\KIEL@chartwidth}{-\KIEL@sourcewidth}%
    \fi
  \fi
  \KIELsetup{gfxopts={width=\KIEL@chartwidth}}%
  %\typeout{** Total width:  \the\KIEL@totalwidth}%
  %\typeout{** Chart width:  \the\KIEL@chartwidth}%
  %\typeout{** Source width: \the\KIEL@sourcewidth}%
}
```

## 6.10 Jobfile creation

`\KIEL@writejobfile` The `\KIEL@writejobfile` writes a jobfile containing the current options.

```
\def\KIEL@writejobfile{%
    \immediate\openout\KIEL@write\KIEL@JobFile
    \ifx\KIEL@opts\@empty % then nothing
    \else
      \immediate\write\KIEL@write{\KIEL@opts}%
    \fi
    \ifx\KIEL@config\@empty % then nothing
    \else
      \immediate\write\KIEL@write{-config=\KIEL@config}%
    \fi
    \ifx\KIEL@focus\@empty % then nothing
    \else
      \immediate\write\KIEL@write{-focus=\KIEL@focus}%
    \fi
    \ifx\KIEL@layouter\@empty % then nothing
    \else
      \immediate\write\KIEL@write{-L=\KIEL@layouter}%
    \fi
    \ifKV@KIEL@highlight
      \else
      \immediate\write\KIEL@write{-nohighlight}%
    \fi
    \ifKV@KIEL@optimize
      \immediate\write\KIEL@write{-Opt}%
    \fi
```

If EPS optimization mode is activated, `kielcmd` is instructed to export a postscript file which then is converted to eps by the `ps2eps` program.

```
    \ifKIEL@OptimizeEPS
      \immediate\write\KIEL@write{-o=\KIEL@figname.ps}%
    \else
      \immediate\write\KIEL@write{-o=\KIEL@figname.eps}%
    \fi
    \immediate\write\KIEL@write{\KIEL@CutFile}%
    \immediate\closeout\KIEL@write
}
```

## 6.11 Embedded statecharts

`KIELstatechart` The KIELstatechart environment. Usage:
`\begin{KIELstatechart}[`⟨*Options*⟩`]`
*embedded statechart code*
`\end{KIELstatechart}`

```
\newenvironment{KIELstatechart}[1][]{%
    \stepcounter{KIEL@figcount}%
    \kvsetkeys{KIEL}{#1}%
    \ifx\KIEL@filetype\@empty
      \PackageError{kieltex}{option 'filetype' is not set!}{%
        The language option must be set for KIELstatechart \MessageBreak
        environments! Set |[language=Esterel/Kit/EsterelStudio]|}%
    \fi
    \xdef\KIEL@CutFile{\KIEL@figname.\KIEL@filetype}%
    \KIEL@verbatimwrite{\KIEL@CutFile}}%
    {\endkielverbatimwrite%
```

```
277       \KIEL@graphicsinclude%
278     }
```

`\KIEL@graphicsprocess` The `\KIEL@graphicsprocess` calls the external programs to process the statechart into an graphic file.

```
279 \long\gdef\KIEL@graphicsprocess{%
280     \xdef\KIEL@JobFile{\KIEL@figname.job}%
281     \KIEL@writejobfile
282     \ifKIEL@ShellEscape
283     \IfFileExists{\KIEL@CutFile}{%
284       \typeout{ *** Running \KIEL@app@kielcmd\space -j=\KIEL@JobFile\space *** }%
285       \immediate\write18{\KIEL@app@kielcmd\space -j=\KIEL@JobFile}%
286       \ifKIEL@OptimizeEPS
287         \typeout{ *** Running \KIEL@app@pstoeps\space \KIEL@figname.ps *** }%
288         \immediate\write18{\KIEL@app@pstoeps\space \KIEL@figname.ps}%
289       \fi
```

Test if external calls were successful.

```
290       \ifpdf
291         \ifKIEL@Generatepdf
292           \typeout{ *** Running \KIEL@app@epstopdf\space \KIEL@figname.eps ***}%
293           \immediate\write18{\KIEL@app@epstopdf\space \KIEL@figname.eps}%
294         \fi
295         \def\KIEL@graphfile{\KIEL@figname.pdf}%
296       \else
297         \def\KIEL@graphfile{\KIEL@figname.eps}%
298       \fi
299       \IfFileExists{\KIEL@graphfile}{%
300         \PackageInfo{kieltex}%
301           {\KIEL@CutFile\space converted}}%
302         {\PackageWarningNoLine{kieltex}%
303           {Creation of \KIEL@CutFile\space failed.}}%
304       }{%
305         \PackageWarningNoLine{kieltex}%
306           {kielcmd source file \KIEL@CutFile\space not found.}%
307       }%
308   \fi
309 }
```

`\KIEL@graphicsinclude` The `\KIEL@graphicsinclude` macro includes a statechart graphic. It implements the different layouts which statecharts can be displayed with.

```
310 \long\gdef\KIEL@graphicsinclude{%
311   \KIEL@calculatewidths
312   \ifx\KIEL@layoutcmd\@empty
313     \ifcase\KIEL@layoutmode\relax
314       % values are: chart,source,split
315       \KIEL@displaychart
316     \or % source
317       \KIEL@displaysource
318     \or % both (same as split1): chart | source
319       \begin{minipage}{\KIEL@chartwidth}%
320         \KIEL@displaychart
321       \end{minipage}
322       \begin{minipage}{\KIEL@sourcewidth}%
323         \KIEL@displaysource
```

```
      \end{minipage}
    \or % split1: chart | source
      \begin{minipage}{\KIEL@chartwidth}%
        \KIEL@displaychart
      \end{minipage}
      \begin{minipage}{\KIEL@sourcewidth}%
        \KIEL@displaysource
      \end{minipage}
    \or % split2: source | chart
      \begin{minipage}{\KIEL@sourcewidth}%
        \KIEL@displaysource
      \end{minipage}
      \begin{minipage}{\KIEL@chartwidth}%
        \KIEL@displaychart
      \end{minipage}
    \or % stack1: chart \\ source
      \KIEL@displaychart\par
      \KIEL@displaysource
    \or % stack2: source \\ chart
      \KIEL@displaysource\par
      \KIEL@displaychart
    \or % none
    \fi
  \else
    \KIEL@layoutcmd
  \fi
}
\newcommand{\KIEL@displaysource}{%
  \lstinputlisting{\KIEL@CutFile}%
  }
\newcommand{\KIEL@displaychart}{%
  \KIEL@graphicsprocess%
  \ifpdf
    \def\KIEL@graphfile{\KIEL@figname.pdf}%
  \else
    \def\KIEL@graphfile{\KIEL@figname.eps}%
  \fi
  \IfFileExists{\KIEL@graphfile}{%
    \includegraphics{\KIEL@graphfile}%
  } {\centerline{\textbf{\KIEL@graphfile\ not found!}}%
      \PackageWarningNoLine{kieltex}%
        {\KIEL@graphfile\ not found! \MessageBreak
          Please convert \KIEL@CutFile\space manually}}%
}
```

`\includeKIELstatechart` Statecharts can be read from external files. With `\includeKIELstatechart[`⟨*KIEL options*⟩`]{`⟨*Statechart filename*⟩`}` an external statechart file in one of the dialects KIEL understands can be referenced. The options are the same as for the `KIELstatechart` environment.

```
\newcommand{\includeKIELstatechart}[2][]{%
  \begingroup
    \stepcounter{KIEL@figcount}%
    \xdef\KIEL@CutFile{#2}%
    \kvsetkeys{KIEL}{#1}%
```

```
    \KIEL@graphicsprocess
    \KIEL@graphicsinclude
  \endgroup
}
```

`\useKIELpicture` Reuse an already generated KIEL image which had a `label=` option set by referencing the label. As the image is already generated, only `graphicx`-options can be set.

```
\newcommand{\useKIELpicture}[2][]{
  \ifcsname KIEL@src@#2\endcsname
    \includegraphics[#1]{\csname KIEL@img@#2\endcsname}%
  \else
    \PackageWarningNoLine{kieltex}%
        {No statechart defined with label `#2Âť!}%
      \fbox{\textbf{Statechart labelled with #2 could not be found!}}%
  \fi
}
```

`\useKIELsource` Same as `\useKIELpicture`, but show the source of the statechart. Only options for the `listings` package can be set.

```
\newcommand{\useKIELsource}[2][]{
  \ifcsname KIEL@src@#2\endcsname
    \lstinputlisting[#1]{\csname KIEL@src@#2\endcsname}%
  \else
    \PackageWarningNoLine{kieltex}%
        {No statechart defined with label `#2Âť!}%
      \fbox{\textbf{Statechart labelled with #2 could not be found!}}%
  \fi
}
\endinput
```

## C.2 Das Modul kiel.preferences

Im Rahmen der Überarbeitung der Property-Verwaltung von KIEL entstand das Modul `kiel.preferences`, dessen Quellcode im Folgenden abgedruckt ist. Es enthält folgende Klassen:

```
public class FixedComboBox extends com.fg.ftreenodes.FComboBox;
public class XColorChooser extends javax.swing.JButton implements
  com.fg.ftreenodes.ICellControl, java.awt.event.ActionListener;
 public final class XMLConfigDialog extends javax.swing.JDialog
  implements java.awt.event.ActionListener,
  java.awt.event.ItemListener, com.fg.xmleditor.FXModelStatusListener;
 public class XMLEditPane \\implements java.util.Observer;
 public final class JAXPValidator public final class LogFile;
 public class PreferencesMessage;
 public final class TypeConverters;
 public final class XMLPreferences extends java.util.Observable;
```

Die Klassen des Moduls lassen sich drei Gruppen zuordnen:

**Konfigurationsverwaltung** Die Klasse `XMLPreferences` enthält die neue Konfigurationsverwaltung von KIEL. Jedes KIEL-Modul erstellt eine `XMLPreferences`-Instanz zur Verwaltung seiner Einstellungen. Dass für jedes Modul nur eine Instanz erstellt wird, wird über die statischen Methoden der Klasse sichergestellt, die die Instanzen nach dem Singleton-Schema [13] verwalten.

**Einstellungsdialog** Der grafische Konfigurationsdialog der KIEL-GUI ist in der Klasse `XMLConfigDialog` implementiert. Er besteht aus einer Reihe von Karteireitern, die jeweils eine Instanz der `XMLEditPane`-Klasse enthalten, die jeweils einem KIEL-Modul zugeordnet ist und über das XAmple-Framework [15] die Einstellungen des zugehörigen Moduls bearbeitbar macht. Das XAmple-Framework erlaubt es, für Felder eigene Editierkomponenten anzugenen, eine solche Komponente ist in der `XColorChooser`-Klasse für Farbeintragungen umgesetzt. Die Klasse `FixedComboBox` stellt eine fehlerbereinigte Version der XAmple-Kombobox zur Verfügung.

**Helferklassen** Die Klasse `JAXPValidator` stellt einen statischen XML-Validator bereit, welcher Instanzen von XML-Dokumenten gegen XML-Schemata validieren kann. In der Klasse `TypeConverters` sind Parsing-Funktionen für KIEL-spezifische Propertytypen definiert. Die Klasse `PreferencesMessage` stellt Nachrichtenobjekte bereit, die bei der Benachrichtigung über geänderte Einstellungen verwendet werden.

## C.2.1 XMLPreferences.java

```
package kiel.preferences;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
import java.util.Observable;
import java.util.Observer;
import java.util.Set;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;

import java.net.MalformedURLException;
import java.net.URL;

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import kiel.preferences.xmlconf.Configuration;

import org.apache.commons.jxpath.JXPathContext;
import org.apache.commons.jxpath.JXPathException;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.Text;
import org.xml.sax.SAXException;


/**
 * The XML-based Preferences class encapulates preferences handling.<br/>
 *
 * @author khe
 */
public final class XMLPreferences extends Observable {
    // ////////////////////////////////////////////////////////////////////////
    // Static part implementing a modified singleton pattern:
    // ////////////////////////////////////////////////////////////////////////

    /**
     *
     */
    private static HashMap hashMap = new HashMap();

    /**
     * Stores options for modules not yet registered.
     */
    private static HashMap cachedOptions = new HashMap();

    /**
     *
     */
    private static LogFile sl = new LogFile("XMLPreferencesRegistry",
            LogFile.ERROR);

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

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

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


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

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

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

        sl.setLogLevel(getLoglevel());

        Observer preferencesObserver = new Observer() {
            public void update(final Observable o, final Object arg) {
```

```
            if (arg.equals(XMLPreferences.PREFERENCES_REMARSHAL)) {
                XMLPreferences.log(LogFile.DEBUG,
                        "Reloading Browser Properties");
                XMLPreferences.reload();
            }
            }
        };
        xmlprefs.addObserver(preferencesObserver);
    }

    /**
     *
     * @return Loglevel for the Preferences Module.
     */
    public static int getLoglevel() {
        return conf.getLoglevel();
    }

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

            return true;
        } catch (JAXBException je) {
            sl.log(LogFile.ERROR, "Error_parsing "
                    + xmlprefs.getXMLPreferencesURL() + ": "
                    + je.getLinkedException().getMessage());
            return false;
        }
    }

    /**
     * Returns XMLPreferences instance for moduleName or null if the modules
     * XML files could not be read.
     *
     * @param moduleName
     *            The (unique) name for which the Preferences object should be
     *            created.
     * @param schemaURL
     *            URL to XML Schema
     * @param defaultURL
     *            java.net.URL object pointing to the default properties
     *            resource
     * @param logger
     *            reference to LogFile object
     * @return Preferences object for moduleName.
     */
    public static synchronized XMLPreferences createInstance(
            final String moduleName, final URL schemaURL, final URL defaultURL,
            final LogFile logger) {
        sl.log(LogFile.DEBUG, "XMLPreferences instance for " + moduleName
                + " requested.");
        if (defaultURL == null) {
            sl.log(LogFile.ERROR,
                    "FATAL: Invalid reference to default preferences!");
            return null;
        }
        if (schemaURL == null) {
            sl.log(LogFile.ERROR, "FATAL: Invalid reference to schema!");
            return null;
        }
        sl.log(LogFile.DEBUG, "with default values from "
                + defaultURL.toString());
        if (!hashMap.containsKey(moduleName)) {
            sl.log(LogFile.DEBUG, "No existing instance for " + moduleName
                    + ", creating new one.");
            XMLPreferences newprefs = new XMLPreferences(moduleName, schemaURL,
                    defaultURL, logger);
            if (newprefs != null) {
                hashMap.put(moduleName, newprefs);
            } else {
                sl.log(LogFile.ERROR,
                        "Could not create XMLPreferences instance for "
                        + moduleName);
                return null;
            }
        } else {
            sl.log(LogFile.DEBUG, "Reusing existing instance for" + moduleName);
        }
        return ((XMLPreferences) hashMap.get(moduleName));
    }

    /**
     *
     * @param moduleName
     *            Name of the module
     * @param schemaURL
     *            URL of the XML schema
     * @param defaultURL
     *            URL of the default XML config file
     * @return XMLPreferences object
     */
    public static XMLPreferences createInstance(final String moduleName,
            final URL schemaURL, final URL defaultURL) {
        // TODO khe: set Loglevel from properties?
        LogFile l = new LogFile(moduleName + "prefs", LogFile.ERROR);
        return createInstance(moduleName, schemaURL, defaultURL, l);
    }

    /**
     * Create a XMLPreferences object for a class/module. Tries to determine
     * the location of the XML-Schema file and XML-Default-Values file from
     * a reference to the main class of the module.
     *
     * @param moduleName Name of the module.
     * @param mClass Reference to base class of the module
     * @return Reference to Preferences object or null if module xml files could
     *  not be found.
     */
```

```
    public static XMLPreferences createInstance(final String moduleName,
            final Class mClass) {

        sl.log(LogFile.INFO, "Initializing preferences for module: "
                + moduleName);

        URL xsdURL = mClass.getResource(moduleName + "-schema.xsd");
        URL xmlURL = mClass.getResource(moduleName + "-defaults.xml");

        sl.log(LogFile.ALL, "XML Schema: " + xsdURL.toString());
        sl.log(LogFile.ALL, "XML default preferences: " + xmlURL.toString());

        return createInstance(moduleName, xsdURL, xmlURL);
    }


    /**
     *
     * @param moduleName .
     * @return .
     */
    public static XMLPreferences getInstance(final String moduleName) {
        return ((XMLPreferences) hashMap.get(moduleName));
    }

    /**
     * Returns the names of registered modules.
     *
     * @return Set of Strings
     */
    public static Set getModuleNamesSet() {
        return hashMap.keySet();
    }

    /**
     * Set a property of a module registered to the Preferences manager.
     * If module is not registered (yet), the options are stored and set as soon
     * as the module is registered.
     * @param module Name of the module
     * @param property Name of the property
     * @param value value to be set
     * @param validate revalidate DOM tree after setting
     * @param reload notify Observers to refresh their properties
     */
    public static void setProperty(final String module, final String property,
            final String value, final boolean validate, final boolean reload) {
        if (hashMap.containsKey(module)) {
            ((XMLPreferences) hashMap.get(module)).setModuleProperty(property,
                    value, validate, reload);
        } else {
            if (!cachedOptions.containsKey(module)) {
                cachedOptions.put(module, new HashMap());
            }
            ((HashMap) cachedOptions.get(module)).put(property, value);
        }
    }

    /**
     * Cause all modules to send an update notification to their observers.
     */
    public static void updateAll() {
        for (Iterator it = hashMap.values().iterator(); it.hasNext();) {
            XMLPreferences modprefs = (XMLPreferences) it.next();
            modprefs.update();
        }
    }

    /**
     * Root of all XML Namespaces defined for KIEL properties.
     */
    static final  String NAMESPACEBASE =
        "http://rtsys.informatik.uni-kiel.de/%7Ert-kiel/kiel/kiel/preferences/";

    /**
     * Generates the complete XML Namespace for a module.
     * @param module Name of the module
     * @return Namespace of the module's XML Config.
     */
    public static String getModuleNamespace(final String module) {
        //return NAMESPACEBASE + module.toLowerCase();
        return NAMESPACEBASE + module;
    }

    /**
     * Normalize path by prepending the value of the kiel.basedir system
     * property to the path if (1) kiel.basedir is set and (2) the path
     * doesn't already start with an /.
     * @param relativePath possibly relative path
     * @return absoulte path
     */
    public static String normalizePath(final String relativePath) {
        String path = relativePath;
        if (System.getProperty("kiel.basedir") != null) {
            if (!path.startsWith("/")) {
                try {
                    path = new File(System.getProperty("kiel.basedir")
                            + File.separator + path).getCanonicalPath();
                } catch (IOException e) {
                    log(LogFile.ERROR, "Failed to normalize path "
                            + path + ": " + e.getMessage());
                }
            }
        }
        log(LogFile.ALL, relativePath + " normalized to " + path);
        return path;
    }

    /**
     * components can register to be notified whenever the preferences have
     * changed, that is, when the user edited the preferences table in the
     * preferences dialog.
     *
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
     */
    public interface Listener {

        /**
         * Is called whenever the preferences values have changed.
         */
        void preferencesChanged();
    }
```

```
//    /**
//     * The listeners to be notified whenever the preferences changed.
//     */
//    private static ArrayList listeners = new ArrayList();

    /**
     * Send this to observers if preferences have changed.
     */
    public static final PreferencesMessage PREFERENCES_UPDATED =
        new PreferencesMessage();


    /**
     * Send this to observers to issue remarshalling of DOM tree.
     */
    public static final PreferencesMessage PREFERENCES_REMARSHAL =
        new PreferencesMessage();

    /**
     * List of the listeners using the legacy Listerers interface. New Classes
     * should use the Observer/Observable Interface.
     * @deprecated
     */
    private static ArrayList listeners = new ArrayList();


    /**
     * Add a listener to the listeners' list.
     *
     * @param listener .
     * @deprecated
     */
    public static void addListener(final Listener listener) {
        //XMLPreferences.log(LogFile.DEBUG, "Depreceated addListener called.");
        listeners.add(listener);
    }

    /**
     * notify all listeners.
     * @deprecated
     */
    public static void notifyListeners() {
        if (listeners != null) {
          for (int i = 0; i < listeners.size(); i++) {
              ((Listener) listeners.get(i)).preferencesChanged();
          }
        }
    }

    /**
     * Return the LogFile object of the property manager. Use this logger
     * for all Preferences-Related messages.
     * @return <code>kiel.util.LogFile</code> object
     */
    public static LogFile getStaticLogger() {
        return sl;
    }

    /**
     * @param level Loglevel
     * @param message Mesage being logged
     */
    public static void log(final int level, final String message) {
        sl.log(level, message);
    }

    /**
     * Cache for loaded images.
     */
    private static Hashtable images = new Hashtable();

    /** Generic icon loader, caches loaded icons.
     *
     * @param filename Name of the icon file.
     * @return icon or null if not found.
     */
    public static Icon getIcon(final String filename) {
        if (filename == null) {
            return null;
        }
        Object result = images.get(filename);
        if (result == null) {
            result = new ImageIcon(XMLPreferences.class.getResource(filename));
            images.put(filename, result);
        }
        return (ImageIcon) result;
    }

    /**
     * Get a icon for the preferences.
     * @return icon or null if not found.
     */
    public static Icon getPrefsIcon() {
        return getIcon("Properties16.gif");
    }


    // ////////////////////////////////////////////////////////////////////////
    // The instance part:
    // ////////////////////////////////////////////////////////////////////////
    /**
     * The name of the module this Object is associated to. Determines the name
     * of the config files too.
     */
    private String moduleName;

    /**
     * URL pointing to the default resources file. Needed for reloading. Usually
     * points to a resource inside a jar file.
     */
    private URL defaultPropertiesURL;

    /**
     * URL pointing to the XML Schema file of XMLified modules.
     */
    private URL schemaURL;

    /**
     * The user settings, stored in ~/.kiel/moduleName.properties.
     */
    private URL userPropertiesURL;
```

```
    /**
     * Path to user settings folder.
     */
    private final String userPathPrefix = System.getProperty("user.home")
    + File.separator + ".kiel" + File.separator;

    /**
     * The xml configuration read into memory.
     */
    private Document xmldata;

    /**
     * The xml schema read into memory.
     */
    private Document xsddata;

    /**
     *
     */
    private LogFile l;

    /**
     *
     */
    private String userPath;

    /**
     * Access the XML configuration.
     */

    /**
     * Internal constructor, run createInstance to get an instance.
     *
     * @param pmoduleName
     *            Name of the Module for which the prefs are handled.
     * @param pschemaURL
     *            URL of the XML schema file
     * @param pdefaultPropertiesURL
     *            URL pointing to default properties file.
     * @param log
     *            Reference to existing kiel.util.LogFile instance.
     */
    private XMLPreferences(final String pmoduleName, final URL pschemaURL,
            final URL pdefaultPropertiesURL, final LogFile log) {
        moduleName = pmoduleName;
        defaultPropertiesURL = pdefaultPropertiesURL;
        schemaURL = pschemaURL;
        userPath = userPathPrefix + moduleName + ".xml";
        this.l = log;
        try {
            // check whether user preferences exist, else copy defaults
            if (!(new File(userPath).exists())) {
                copyDefaultsToUserPrefs();
            }
            userPropertiesURL = new URL("file:" + userPath);

            // check whether user preferences are valid, else copy defaults
            if (!JAXPValidator.validateXmlBySchema(
                    schemaURL, userPropertiesURL)) {
                sl.log(LogFile.ERROR, "User preferences in "
                        + userPropertiesURL.getPath() + " are invalid.");
                copyDefaultsToUserPrefs();
            }
        } catch (MalformedURLException e) {
            sl.log(LogFile.ERROR, "Invalid user preferences file"
                    + e.getMessage());
            copyDefaultsToUserPrefs();
        }
        loadSchema();
        loadUserXML();

        if (cachedOptions.containsKey(moduleName)) {
            Set options = ((HashMap) cachedOptions.get(moduleName)).entrySet();
            for (Iterator iter = options.iterator(); iter.hasNext();) {
                Map.Entry entry = (Map.Entry) iter.next();
                setModuleProperty((String) entry.getKey(),
                        (String) entry.getValue(), false, false);
            }
            cachedOptions.remove(moduleName);
            update();
        }
    }

    /**
     *  This method writes a DOM document to a file.
     *  @param doc DOM tree object.
     *  @param filename Filename to write DOM tree to.
     */
    public static void writeXmlFile(final Document doc, final String filename) {
        try {
            // Prepare the DOM document for writing
            Source source = new DOMSource(doc);

            // Prepare the output file
            File file = new File(filename);
            Result result = new StreamResult(file);

            // Write the DOM document to the file
            Transformer xformer = TransformerFactory.newInstance()
                .newTransformer();
            xformer.transform(source, result);
        } catch (TransformerConfigurationException e) {
            e.printStackTrace();
        } catch (TransformerException e) {
            e.printStackTrace();
        }
    }

    /**
     * Read a XML file into a Document instance.
     * @param path Location of the file to be read.
     * @return Document containing the parsed file.
     */
    private static Document readXML(final String path) {
        Document d;
        try {
            d = readXML(new FileInputStream(path));
        } catch (FileNotFoundException e) {
            XMLPreferences.log(LogFile.ERROR, "Could not open " + path
                    + " for reading!");
            d = null;
        }
```

```
        return d;
    }

    /**
     * Parse a XML input Stream into a Document instance.
     * @param is Input Stream.
     * @return Document instance or null if an error occured.
     */
    private static Document readXML(final InputStream is) {
        // TODO khe: better error handling
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document dom = null;
        try {
            DocumentBuilder db = dbf.newDocumentBuilder();
            dom = db.parse(is);
            XMLPreferences.log(LogFile.DEBUG, "root_of_dom_tree:␣"
                    + dom.getDocumentElement().getNodeName());
            return dom;
        } catch (ParserConfigurationException pce) {
            //pce.printStackTrace();
            return null;
        } catch (SAXException se) {
            //se.printStackTrace();
            return null;
        } catch (IOException ioe) {
            //ioe.printStackTrace();
            return null;
        }
    }

    /**
     * (Re-)Read XML Schema into xsddata field.
     * @return success.
     */
    private boolean loadSchema() {
        Document dom;
        try {
            dom = readXML(schemaURL.openStream());

        } catch (IOException e) {
            l.log(LogFile.ERROR, "Error␣parsing␣schema:␣" + e.getMessage());
            return false;
        }
        xsddata = dom;
        return true;
    }

    /**
     * (Re-)Read user preferences. The parsed preferences are stored in the
     *  xmldate field.
     * @return success.
     */
    private boolean loadUserXML() {
        Document dom;
        try {
            dom = readXML(userPropertiesURL.openStream());
        } catch (IOException e) {
            l.log(LogFile.ERROR, "Error␣parsing␣user␣Preferences:␣"
                    + e.getMessage());
            return false;
        }
        xmldata = dom;
        return true;
    }

    /**
     * Replace the user Preferences with the default preferences. Calls update()
     * after copying.
     */
    public void loadDefaults() {
        copyDefaultsToUserPrefs();
        loadUserXML();
        update();
    }

    /**
     * Set the preferences DOM object. Calls update() after setting.
     * @param dom new Preferences.
     */
    public void updatePreferences(final Document dom) {
        xmldata = dom;
        update();
    }

    /**
     * Set the preferences from a DOMsource object.
     * @param source DOMsource instance.
     */
    public void updatePreferences(final DOMSource source) {
        updatePreferences((Document) source.getNode());
    }

    /**
     * @return xml schema DOM tree.
     */
    public Document getSchema() {
        return xsddata;
    }

    /**
     * @return xml preferences DOM tree.
     */
    public Document getPreferences() {
        return xmldata;
    }

    /**
     *
     * @return xml schema url.
     */
    public URL getXMLSchemaURL() {
        return schemaURL;
    }

    /**
     *
     * @return xml default properties url
     */
    public URL getXMLDefaultsURL() {
        return defaultPropertiesURL;
    }
```

```
    /**
     *
     * @return URL to xml preferences file
     */
    public URL getXMLPreferencesURL() {
        return userPropertiesURL;
    }

    /**
     * Set a property by directly manipulating the DOM tree. Use only if
     * the name of the property is not known at compilation time, otherwise
     * use the setter methods of the relevant xxProperties class.
     * @param property Name of property
     * @param value Value to set
     * @param validate validate DOM tree after setting
     * @param reload notify Observers after setting
     */
    private void setModuleProperty(final String property, final String value,
            final boolean validate, final boolean reload) {
        try {
            log(LogFile.DETAIL, "Setting " + property + "_to_" + value);
            JXPathContext context = JXPathContext.newContext(xmldata);
            context.registerNamespace("ns0", getModuleNamespace(moduleName));
            Node node = (Node) context.selectSingleNode("ns0:configuration/ns0:"
                    + property);
            if (node.getFirstChild() == null) {
                node.appendChild(xmldata.createTextNode(value));
            } else {
                ((Text) node.getFirstChild()).setData(value);
            }
            //if (validate) {
            //    // TODO khe: implement?
            //}
            if (reload) {
                update();
            }
        } catch (JXPathException e) {
            XMLPreferences.log(LogFile.ERROR, "Error:_Could_not_set_property_'"
                    + property + "'_in_module_" + this.moduleName);
        }
    }

    /**
     * Save configuration of the corresponding module to the user preferences
     * file.
     * @return sucess
     */
    public boolean save() {
        l.log(LogFile.DEBUG,
                "Attempting_to_save_XML_config_for_" + this.moduleName);
        String fileName = userPath;
        XMLPreferences.log(LogFile.INFO, "Saving_to_" + fileName);
        File file = new File(fileName);
        FileOutputStream out = null;
        Writer writer = null;
        try {
            file.createNewFile();
            out = new FileOutputStream(file);
            OutputFormat format = new OutputFormat(xmldata, "UTF-8", true);
            writer = new OutputStreamWriter(out, "UTF-8");
            XMLSerializer serial = new XMLSerializer(writer, format);
            serial.asDOMSerializer();
            serial.serialize(xmldata);
            update();
//            notifyListeners();
            return true;
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            try { out.close(); } catch (Exception ignore) { return false; }
            try { writer.close(); } catch (Exception ignore) { return false; }
        }
        return false;
    }

    /**
     * Notifies the Observers about changed preferences. Also notifies
     * Listeners using the legacy listener interface.
     * @return success.
     */
    public boolean update() {
        l.log(LogFile.DEBUG,
                "Attempting_to_(re)load_XML_config_for_" + this.moduleName);
        // first update round: let the marshallers pick up the changes
        XMLPreferences.log(LogFile.DEBUG, "Updating_marshallers");
        setChanged();
        notifyObservers(PREFERENCES_REMARSHAL);
        // second update round: read the changed values
        XMLPreferences.log(LogFile.DEBUG, "Updating_properties_via_Observer");
        setChanged();
        notifyObservers(PREFERENCES_UPDATED);
        // update components that use the editor's Listener interface
        XMLPreferences.log(LogFile.DEBUG, "Updating_properties_via_Listener");
        XMLPreferences.notifyListeners();
        return true;
    }

    /**
     * Check if userspecific config directory exists and try to create it if it
     * does not.
     *
     * @return success.
     *
     */
    private boolean checkHomeConfigDir() {
        File f = new File(System.getProperty("user.home")
        + java.io.File.separator + ".kiel");
        if (!f.isDirectory()) {
            if (!f.mkdir()) {
                l.log(LogFile.ERROR, "Couldn't_create_Settings_Dir:_"
                        + f.getName());
                return false;
            }
        }
        return true;
    }

    /**
     * copy the defaults resource over the user prefs file. This is done either
     * when the user file does not exist or when it is outdated (getProperty
     * sets userFileOutdated if a requested value is) not found in userprefs but
```

```
     * in default prefs.
     */
    private void copyDefaultsToUserPrefs() {
        l.log(LogFile.INFO, "Copying_default_values_to_user_file.");
        checkHomeConfigDir();
        InputStream is;
        FileWriter fw;

        try {
            is = defaultPropertiesURL.openStream();
            fw = new FileWriter(userPath);

            int c = is.read();
            while (c >= 0) {
                fw.write(c);
                c = is.read();
            }
            fw.flush();

            is.close();
            fw.close();
        } catch (IOException storeException) {
            l.log(LogFile.ERROR, storeException.toString());
        }
    }

    /**
     * returns whether values have changed. Not implemented yet.
     *
     * @return .
     */
    public boolean changed() {
        //TODO khe: implement
        return false;
    }

    /**
     *
     * @param pl
     *             kiel.util.LogFile
     */
    public void setLogger(final LogFile pl) {
        this.l = pl;
    }

    /**
     *
     * @return LogFile
     */
    public LogFile getLogger() {
        return this.l;
    }

}
```

## C.2.2 XMLConfigDialog.java

```
package kiel.preferences;

import java.awt.Dimension;
import java.awt.Toolkit;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;

import com.fg.xmleditor.FXModel;
import com.fg.xmleditor.FXModelStatusListener;
import com.fg.xmleditor.FXStatusEvent;
import com.fg.xmleditor.FXView;


/**
 * XML-based property configuration dialog.
 * @author Karsten Heymann (khe@informatik.uni-kiel.de)
 */
public final class XMLConfigDialog extends JDialog implements ActionListener,
        ItemListener, FXModelStatusListener {

    /**
     * The dialog instance.
     */
    private static XMLConfigDialog instance = null;

    /**
     * Create a XMLConfigDialog instance if none exists yet and return the
     * existing or created instance.
     * @param parent JFrame
     * @return XMLConfigDialog
     */
    public static XMLConfigDialog getInstance(final JFrame parent) {
        if (instance == null) {
            instance = new XMLConfigDialog(parent);
        }
        return instance;
    }

    /**
     * Button panel.
     */
    private JPanel buttonPanel;

    /**
     * Cancel button.
     */
    private JButton cancelButton;

    /**
     * Apply button.
     */
    private JButton applyButton;

    /**
     * Load defaults button.
     */
    private JButton loadDefaultsButton;

    /**
     * Save button.
     */
    private JButton saveButton;

    /**
     * Name of default tab.
     * TODO make this a property?
     */
    private static final String DEFAULTTAB = "browser";

    /**
     * Serial version uid.
     */
    private static final long serialVersionUID = -58192155248387278 80L;

    /**
     * HashMap containing the edit panes.
     */
    private HashMap panes;

    /**
     * Dialog title.
     */
    private static final String TITLE = "KIEL_Preferences";

    /**
     * Dialog border size.
     */
    private static final int BORDERSIZE = 10;

    /**
     * Insert a centered label into the dialog.
     * @param text Text of the label.
     */
    private void addInfoLine(final String text) {
        JLabel l = new JLabel(text);
        l.setAlignmentX(JLabel.CENTER_ALIGNMENT);
        getContentPane().add(l);
    }
```

```
    /**
     * @param s String.
     * @return String with first letter capitalized.
     */
    private String capitalize(final String s) {
        return s.substring(0, 1).toUpperCase() + s.substring(1);
    }

    /**
     *
     * @param parent Parent window
     */
    private XMLConfigDialog(final JFrame parent) {
        super(parent, "", true);

        Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        final int dialogSize = 545;
        //final int headerHight = 32;
        setBounds(
            (d.width - dialogSize) / 2,
            (d.height - dialogSize) / 2,
            dialogSize,
            dialogSize);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        getContentPane().setLayout(new BoxLayout(getContentPane(),
                BoxLayout.Y_AXIS));

        addInfoLine(XMLConfigDialog.TITLE);

        Set moduleNames = XMLPreferences.getModuleNamesSet();
        panes = new HashMap();
        int defaultTabIndex = -1;
        addInfoLine("Number of registered modules: " + moduleNames.size());
        if (moduleNames.isEmpty()) {
            getContentPane().add(Box.createVerticalGlue());
            addInfoLine("No module registered to the XML property manager");
            getContentPane().add(Box.createVerticalGlue());
        } else {
            JTabbedPane tabbedPane = new JTabbedPane();
            for (Iterator it = moduleNames.iterator(); it.hasNext();) {
                String name = it.next().toString();
                XMLEditPane pane = new XMLEditPane(name);
                pane.getModel().addModelStatusListener(this);
                panes.put(name, pane);
                String title = capitalize(name);
                tabbedPane.addTab(title, null, pane.getView(),
                        title + " Prefs");
                if (name == DEFAULTTAB) {
                    defaultTabIndex = tabbedPane.indexOfTab(title);
                }
            }
            if (defaultTabIndex != -1) {
                tabbedPane.setSelectedIndex(defaultTabIndex);
            }
            getContentPane().add(tabbedPane);
        }

        buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
        buttonPanel.setBorder(BorderFactory.createEmptyBorder(
                BORDERSIZE,    //top
                BORDERSIZE,    //left
                BORDERSIZE,    //bottom
                BORDERSIZE)); //right

        cancelButton = new JButton("Cancel");
        ActionListener cancelButtonListener = new ActionListener() {
            public void actionPerformed(final ActionEvent actionEvent) {
                setVisible(false);
            }
        };
        cancelButton.addActionListener(cancelButtonListener);

        loadDefaultsButton = new JButton("Load_default_values");
        ActionListener loadDefaultsButtonListener = new ActionListener() {
            public void actionPerformed(final ActionEvent actionEvent) {
                loadDefaults();
            }
        };
        loadDefaultsButton.addActionListener(loadDefaultsButtonListener);

        applyButton = new JButton("Apply");
        ActionListener applyButtonListener = new ActionListener() {
            public void actionPerformed(final ActionEvent actionEvent) {
                applyPanes();
                setVisible(false);
            }
        };
        applyButton.addActionListener(applyButtonListener);

        saveButton = new JButton("Save");
        ActionListener saveButtonListener = new ActionListener() {
            public void actionPerformed(final ActionEvent actionEvent) {
                savePanes();
                setVisible(false);
            }
        };
        saveButton.addActionListener(saveButtonListener);

        buttonPanel.add(cancelButton);
        buttonPanel.add(loadDefaultsButton);
        buttonPanel.add(Box.createHorizontalGlue());
        //buttonPanel.add(applyButton);
        //buttonPanel.add(Box.createHorizontalStrut(30));
        buttonPanel.add(saveButton);

        updateSaveButton();
        getContentPane().add(buttonPanel);
    }

    /**
     * Updated KIELs preferences store for all changed edit panes.
     *
     */
    private void applyPanes() {
        for (Iterator it = panes.keySet().iterator(); it.hasNext();) {
            String name = it.next().toString();

            XMLEditPane pane = ((XMLEditPane) panes.get(name));
            FXView view = pane.getView();
            if (view.isDocChanged()) {
```

```
                XMLPreferences.log(LogFile.INFO, "== applying changes "
                        + "for module " + name + " ==");
                pane.applyChanges();
            } else {
                XMLPreferences.log(LogFile.INFO, "Module " + name
                        + " unchanged");
            }
        }
    }

    /**
     * Save all changed panes.
     */
    private void savePanes() {
        for (Iterator it = panes.keySet().iterator(); it.hasNext();) {
            String name = it.next().toString();

            XMLEditPane pane = ((XMLEditPane) panes.get(name));
            FXView view = pane.getView();
            if (view.isDocChanged()) {
                XMLPreferences.log(LogFile.INFO, "== Saving module "
                        + name + " ==");
                pane.saveChanges();
            } else {
                XMLPreferences.log(LogFile.INFO, "Module " + name
                        + " unchanged");
            }
        }
    }

    /**
     * Load default values for all modules.
     */
    private void loadDefaults() {
        for (Iterator it = panes.keySet().iterator(); it.hasNext();) {
            String name = it.next().toString();

            XMLEditPane pane = ((XMLEditPane) panes.get(name));
            pane.loadDefaults();
        }
    }


    /**
     * @see java.awt.event.ActionListener#actionPerformed(ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) { }

    /**
     * @see java.awt.event.ItemListener#itemStateChanged(ItemEvent)
     */
    public void itemStateChanged(final ItemEvent e) { }

    /**
     * Check all panes for validity.
     * @return true if all panes contain valid editors, else false.
     */
    public boolean isValid() {
        boolean valid = true;
        for (Iterator it = panes.keySet().iterator(); it.hasNext();) {
            String name = it.next().toString();
            FXModel model = ((XMLEditPane) panes.get(name)).getModel();
            valid = valid && model.isDocumentValid();
        }
        return valid;
    }

    /**
     * Disables the Save button if at least one editpane contains invalid data,
     * else enables it.
     *
     */
    public void updateSaveButton() {
        saveButton.setEnabled(isValid());
    }

    /**
     * @see com.fg.xmleditor.FXModelStatusListener
     *      #docValidityStatusChanged(FXStatusEvent)
     */
    public void docValidityStatusChanged(final FXStatusEvent arg0) {
        updateSaveButton();
    }

    /**
     * @see com.fg.xmleditor.FXModelStatusListener
     *      #newDocumentLoaded(FXStatusEvent)
     */
    public void newDocumentLoaded(final FXStatusEvent arg0) { }

}
```

## C.2.3 XMLEditPane.java

```
package kiel.preferences;

import java.net.URL;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JOptionPane;

import org.w3c.dom.Document;
import org.w3c.dom.Node;

import com.fg.xmleditor.FXDocumentModel;
import com.fg.xmleditor.FXDocumentModelImpl;
import com.fg.xmleditor.FXView;

/**
 * An edit pane containing a XAmple editing component for the preferences of
 * one KIEL module.
 * @author khe
 *
 */
public class XMLEditPane implements Observer {

    /**
     * FXView.
     */
    private FXView xmlView;

    /**
     * FXDocumentModel.
     */
    private FXDocumentModel model;

    /**
     * Reference to xml preferences.
     */
    private XMLPreferences xmlprefs;

    /**
     * Create a edit pane for module moduleName. Reads xml schema and current
     * xml preferences for moduleName and generates a editing component from
     * that.
     * @param moduleName Name of the module.
     */
    public XMLEditPane(final String moduleName) {
        XMLPreferences.log(LogFile.INFO, "Creating␣XMLEditPane␣for␣Module␣"
                + moduleName);
        xmlprefs = XMLPreferences.getInstance(moduleName);

        if (xmlprefs != null) {
            model = new FXDocumentModelImpl();
            xmlView = new FXView(model, FXView.NORTH);

            loadXMLSchema();
            loadXMLDocument();

            xmlprefs.addObserver(this);
        }
    }

    /**
     * Fetch the (possibly changed) DOM Document from the XAmple editor
     * component and submit it to the XMLPreferences instance of the
     * corresponding module.
     */
    public final void applyChanges() {
        Document dom = model.getDocument();
        xmlprefs.updatePreferences(dom);
    }

    /**
     * As applyChanges(), but also save the changes to disk.
     *
     */
    public final void saveChanges() {
        applyChanges();
        xmlprefs.save();
    }

    /**
     * Returns the view.
     * @return FXView.
     */
    public final FXView getView() {
        return xmlView;
    }

    /**
     * Returns the Document Model.
     * @return Document Model.
     */
    public final FXDocumentModel getModel() {
        return model;
    }

    /**
     * (re)-loads XML schema.
     * @return success.
     */
    private boolean loadXMLSchema() {
        try {
            URL url = xmlprefs.getXMLSchemaURL();
            model.newDocument(url);
            xmlView.showInfoMessage("XML␣Schema:␣" + url.getPath());
            XMLPreferences.log(LogFile.DEBUG, "XML␣Schema:␣" + url.toString());
            return true;
        } catch (Exception ex) {
            xmlView.showErrorMessage("Error:␣" + ex.toString());
            JOptionPane.showMessageDialog(xmlView, "Can't␣load␣the␣XML␣Schema",
                    "Error", JOptionPane.ERROR_MESSAGE);
            return false;
        }
    }

    /**
     * Loads the default values for this panes XMLPreferences object.
```

```
     *
     */
    public final void loadDefaults() {
        xmlprefs.loadDefaults();
    }

    /**
     * (Re-)Loads the XML preferences.
     * @return success
     */
    private boolean loadXMLDocument() {
        try {
            Document doc = xmlprefs.getPreferences();
            java.util.List lostElements = model.openDocument(model
                    .getSchemaURL(), doc);
            if (lostElements != null) {
                StringBuffer sb = new StringBuffer(
                        "Error:_The_source_XML_document_is_invalid.\n"
                                + "The_following_elements_have_not_been_loaded:");
                for (int i = 0; i < lostElements.size(); i++) {
                    sb.append("\n");
                    int k = sb.length();
                    Node element = (Node) lostElements.get(i);
                    sb.append(element.getNodeName());
                    Node node = element.getParentNode();
                    while (node != null && !(node instanceof Document)) {
                        sb.insert(k, node.getNodeName() + "/");
                        node = node.getParentNode();
                    }
                }
                xmlView.showErrorMessage(sb.toString());
            }
            return true;
        } catch (Exception ex) {
            // xmlView.clear();
            xmlView.showErrorMessage("Error:_" + ex.getMessage());
            JOptionPane.showMessageDialog(xmlView, "Can't_load_XML_Document",
                    "Error", JOptionPane.ERROR_MESSAGE);
            return false;
        }
    }

    /**
     * Reload preferences from DOM object. Called from Observable.
     * @param o Observable
     * @param arg argument
     */
    public final void update(final Observable o, final Object arg) {
        if (arg.equals(XMLPreferences.PREFERENCES_UPDATED)) {
            XMLPreferences.log(LogFile.ALL, "Updating_xml_editpane");
            loadXMLDocument();
        }
    }
}
```

## C.2.4 XColorChooser.java

```
package kiel.preferences;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JColorChooser;

import com.fg.ftreenodes.ICellControl;
import com.fg.ftreenodes.Params;
import com.fg.xmleditor.FXBasicView;

/**
 * The XColorChooser class implements a color chooser component for the XAmple
 * XML editor.
 *
 * @author khe@informatik.uni-kiel.de
 */
public class XColorChooser extends JButton
    implements ICellControl, ActionListener {

    /**
     * Serial version uid.
     */
    private static final long serialVersionUID = 6900320494273562304L;

    /**
     * Current chosen color.
     */
    private Color col;

    /**
     * Default x size of color chooser button.
     */
    private static final int DEFAULT_XSIZE = 100;

    /**
     * Default y size of color chooser button.
     */
    private static final int DEFAULT_YSIZE = 20;

    /**
     * Constructor.
     */
    public XColorChooser() {
        super();
        setPreferredSize(new Dimension(DEFAULT_XSIZE, DEFAULT_YSIZE));
        setFont(getFont().deriveFont(Font.PLAIN));
        col = new Color(0, 0, 0);
        updateColor();
    }

    /**
     * Horizontal size of editor field.
     */
     private static final int XSIZE = 300;

    /**
     * Vertical size of editor field.
     */
    private static final int YSIZE = 20;

    /**
     * Update the color field. Also update the text field with a textual
     * representation of the selected color.
     */
    private void updateColor() {
        setBackground(col);
        setText(TypeConverters.printColorToString(col));
    }

    /**
     * @see com.fg.ftreenodes.ICellControl#getData()
     */
    public final Object getData() {
        return TypeConverters.printColorToString(col);
    }

    /**
     * @see com.fg.ftreenodes.ICellControl#initCellControl(boolean)
     */
    public final void initCellControl(final boolean isEditor) {
        if (isEditor) { // Field editor
            addActionListener(this);
        } // else: Field renderer
    }

    /**
     * @see com.fg.ftreenodes.ICellControl#updateCellControl(boolean, boolean,
     *      boolean, java.lang.Object, com.fg.ftreenodes.Params)
     */
    public final void updateCellControl(final boolean isEditor,
            final boolean enabled, final boolean editable,
            final Object data, final Params params) {
        setEnabled(enabled && editable);
        try {
            col = TypeConverters.parseStringToColor(data.toString());
            updateColor();
        } catch (Exception ex) {
            setPreferredSize(new Dimension(XSIZE, YSIZE));
            setText("Error: " + ex.getMessage());
        }
    }

    /**
     * @param e Event.
     */
    public final void actionPerformed(final ActionEvent e) {
        Color newcol = JColorChooser.showDialog(XColorChooser.this,
                "Choose a color", col);
        if (newcol != null) {
            col = newcol;
            updateColor();
        }
        FXBasicView.stopCellEditing(XColorChooser.this);
    }
}
```

## C.2.5 FixedComboBox.java

```
/**
 *
 */
package kiel.preferences;

import java.util.Iterator;
import java.util.List;

import com.fg.ftreenodes.FComboBox;
import com.fg.ftreenodes.ListParams;
import com.fg.ftreenodes.Params;

/**
 * <p>A bugfix version of the com.fg.ftreenodes.FComboBox class. On certain XML
 * documents, FComboBox inserts a superflous <code>null\n</code> into the edit
 * field. This class removes this string, and behaves otherwise exactly like
 * FComboBox.
 * </p>
 *  @author <a href="mailto:khe@informatik.uni-kiel.de">Karsten Heymann</a>
 */
public class FixedComboBox extends FComboBox {

    /**
     * <p>Patch FComboBox.updateCellControl to remove superflous
     * <code>null\n</code> string.
     * </p>
     * @see com.fg.ftreenodes.FComboBox.updateCellControl
     */
    public final void updateCellControl(
            final boolean isEditor,
            final boolean enabled,
            final boolean editable,
            final Object data,
            final Params params) {
        List paramList = params.getList();
        Iterator itr = paramList.iterator();
        ListParams fixedParams = new ListParams(0);
        for (int i = 0, n = paramList.size(); i < n; i++) {
          Object o = itr.next();
          String s = o.toString();
          if (s.startsWith("null\n")) {
              s = s.substring("null\n".length());
          }
          fixedParams.add(s);
        }
        super.updateCellControl(isEditor, enabled, editable, data, fixedParams);
    }
}
```

## C.2.6 PreferencesMessage.java

```
package kiel.preferences;

/**
 * <p>Description: A simple message class that can contains a string parameter
 * and an unique id for fast comparing.</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: Uni Kiel</p>
 *
 * @author <a href="mailto:miwi@informatik.uni-kiel.de">Mirko Wischer</a>
 * @version $Revision: 1.1 $ last modified $Date: 2007/02/23 01:51:15 $
 */
public class PreferencesMessage {
  /**
   * max type in all messages.
   */
  private static int max = 0;
  /**
   * parameter for this message.
   */
  private Object parameter;
  /**
   * type of message.
   */
  private int type;

  /**
   * ModelMessage constructor sets unique id for this message.
   */
  public PreferencesMessage() {
    setType(max);
    max++;
  }


  /**
   *
   * @param t int set message type to this
   */
  private void setType(final int t) {
    this.type = t;
  }

  /**
   * @param para sets string parameter
   */
  public final void setParameter(final Object para) {
    this.parameter = para;
  }

  /**
   *
   * @return type of message
   */
  public final int getType() {
    return type;
  }

  /**
   * @return Object Parameter
   */
  public final Object getParameter() {
    return this.parameter;
  }

  /**
   * Test if to messages are the same (according type).
   *
   * @param m ModelMessage
   * @return boolean
   */
  public final boolean equals(final PreferencesMessage m) {
    if (m.getType() == this.type) {
      return true;
    }
    return false;
  }

  /**
   * Returns the hashcode of the Message.
   * @return Returns the hashcode of the Message
   */
 public final int hashCode() {
     return this.type;
 }

 /**
  *
  * @return String of this class
  */
 public final String toString() {
   return "" + type;
 }

}
```

## C.2.7 JAXPValidator.java

```
package kiel.preferences;

import java.io.IOException;
import java.net.URL;

import javax.xml.XMLConstants;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

import org.xml.sax.SAXException;

/**
 * Simple JAXP based XML Schema Validator.
 * @author khe
 */
public final class JAXPValidator {

    /**
     * @param xsd XML Schema
     * @param xml XML File to be verified against schema
     * @return true if schema verifies XML, else false.
     */
    public static boolean validateXmlBySchema(final URL xsd, final URL xml) {

        LogFile l = XMLPreferences.getStaticLogger();

        if ((xsd == null) || (xml == null)) {
            l.log(LogFile.DEBUG, "JAXPValidator:_Schema_or_XML_file_URL_were_"
                    + "null!");
            return false;
        }
        SchemaFactory schemaFactory = SchemaFactory
                .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
        Source schemaSource;
        Schema schema;
        Validator validator;
        try {
            schemaSource = new StreamSource(xsd.openStream());
            schema = schemaFactory.newSchema(schemaSource);
            validator = schema.newValidator();
            validator.validate(new StreamSource(xml.openStream()));
        } catch (IOException e) {
            l.log(LogFile.DETAIL, "JAXPValidator:_IOException_"
                    + e.getMessage());
            return false;
        } catch (SAXException e) {
            l.log(LogFile.DETAIL, "JAXPValidator:_SAXException_"
                    + e.getMessage());
            return false;
        }
        return true;
    }

    /**
     * Don't use.
     */
    private JAXPValidator() {
        System.err.println("Don't_call_me!");
    }

}
```

## C.2.8 TypeConverters.java

```
package kiel.preferences;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Rectangle;
import java.util.StringTokenizer;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.swing.KeyStroke;

/**
 * Some static type conversion functions.
 * @author khe
 *
 */
public final class TypeConverters {

    /**
     * Pretty-Prints a color as a "red,green,blue"-String.
     * @param c Color to be shown.
     * @return String representing the color.
     */
    public static String printColorToString(final Color c) {
        return c.getRed() + "," + c.getGreen() + "," + c.getBlue();
    }

    /**
     * Numeric base for decimal numbers.
     */
    private static final int DECIMALBASE = 10;
    /**
     * Numeric base for hexadecimal numbers.
     */
    private static final int HEXBASE = 16;

    /**
     * Index of red color component.
     */
    private static final int REDINDEX = 1;
    /**
     * Index of green color component.
     */
    private static final int GREENINDEX = 2;
    /**
     * Index of blue color component.
     */
    private static final int BLUEINDEX = 3;

    /**
     * Parse a color string into a color. Allowed formats are decimal ("r,g,b")
     * and hexadecimal ("0x77AAEE").
     * @param s String representing a color.
     * @return A java.awt.Color instance or none if the color could not be
     *   parsed.
     */
    public static Color parseStringToColor(final String s) {
        if (s == null) {
            return null;
        }
        Color col = new Color(0, 0, 0);
        int base = 0;
        // Color format 1: abc,def,xyz
        Matcher m1 = Pattern.compile("(\\d+),(\\d+),(\\d+)").matcher(s);
        // Color format 2: #aabbcc
        String hex = "(\\p{XDigit}\\p{XDigit})";
        Matcher m2 = Pattern.compile("#" + hex + hex + hex).matcher(s);
        // make checkstyle happy:
        Matcher m = Pattern.compile("DUMMYPATTERN").matcher("IMPOSSIBLE");
        if (m1.matches()) {
            base = DECIMALBASE;
            m = m1;
        } else if (m2.matches()) {
            base = HEXBASE;
            m = m2;
        }
        if (base > 0) {
            int red = Integer.parseInt(m.group(REDINDEX), base);
            int green = Integer.parseInt(m.group(GREENINDEX), base);
            int blue = Integer.parseInt(m.group(BLUEINDEX), base);
            col = new Color(red, green, blue);
        }
        return col;
    }

    /**
     * Returns the preference value of the given key as dimension.
     * If that fails (because of an invalid value in the preferences file), the
     * appropriate value of the default preferences is returned.
     * @param d .
     * @return .
     */
    public static Dimension parseDimensionFromString(final String d) {
        Dimension result;
        try {
            StringTokenizer st = new StringTokenizer(d, "␣");
            result = new Dimension(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
        } catch (NumberFormatException e) {
                result = null;
        }
        return result;
    }

    /**
     * @param d Dimension
     * @return String representation of dimension
     */
    public static String printDimensionToString(final Dimension d) {
        return d.height + "␣" + d.width;
    }

    /**
     * @param p String representing a point
```

```
     * @return Point
     */
    public static Point parsePointFromString(final String p) {
        Point result;
        try {
            StringTokenizer st = new StringTokenizer(p, "␣");
            result = new Point(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
        } catch (NumberFormatException e) {
            result = null;
        }
        return result;
    }

    /**
     * @param p Point
     * @return String representing the Point
     */
    public static String printPointToString(final Point p) {
        return p.x + "␣" + p.y;
    }

    /**
     * @param r Rectangle
     * @return String representing rectangle
     */
    public static Rectangle parseRectangleFromString(final String r) {
        Rectangle result;
        try {
            StringTokenizer st = new StringTokenizer(r, "␣");
            result = new Rectangle(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
        } catch (NumberFormatException e) {
            result = null;
        }
        return result;
    }

    /**
     * @param r Rectangle
     * @return String representing the rectangle
     */
    public static String printRectAngleToString(final Rectangle r) {
        return r.x + "␣" + r.y + "␣" + r.width + "␣" + r.height;
    }

    /**
     * Convert String to KeyStroke. Format must be ⟨code⟩"KEY"⟨/code⟩ or
     * ⟨code⟩"MODIFIER Key"⟨/code⟩ or ⟨code⟩"MODIFIER+Key"⟨/code⟩.
     * TODO khe: rename to parseXXXFromYYY/printXXXToYYY
     *
     * @param s String with KeyStroke.
     * @return  KeyStroke object.
     */
    public static KeyStroke getKeyStrokeFromString(final String s) {
        return KeyStroke.getKeyStroke(s.replace('+', '␣'));
    }

    /**
     * Not ment for instantiation.
     */
    private TypeConverters() { }

}
```

## C.3 XML-Converter-Skripte

Die XML-Converter-Skripte können verwendet werden, um aus einer Property-Liste im CSV-Format XML-Schemata und XML-Konfigurationsdateien mit den Vorgabewerten für ein oder mehrere Module zu generieren. Auch wenn die Skripte primär für die Entwicklung von KielTEX verwendet wurden, können sie dennoch auch für Autoren zukünftiger KIEL-Module von Interesse sein (s. Abschnitt B) und sind daher an dieser Stelle mit abgedruckt.

**csv2xml.py** generiert aus einer CSV-Datei XML-Schemata und XML-Konfigurationsdateien. Es ist in Python geschrieben und verwendet zur XML-Verarbeitung die *ElementTree*-Bibliothek[1].

**generate-properties.sh** ist ein bash-Skript, welches `cvs2xml.py` aufruft und die erzeuten XML-Dateien anschließend mit dem xmlstarlet-Werkzeug[2] formatiert und validiert.

[1] `http://effbot.org/zone/element-index.htm`
[2] `http://xmlstar.sourceforge.net/`

## C.3.1 generate-properties.sh

```
#!/bin/bash
set -e
CSV=${1:-KielPropertyList.csv}
SXC=KielPropertyList.sxc
if $(which xmlstarlet > /dev/null)
then
  XMLBIN=$(which xmlstarlet)
else
  echo "*** ERROR: The program 'xml' could not be found in path! ***"
  echo "*** On Debian, please install the package xmlstarlet     ***"
  echo "*** (see http://xmlstar.sourceforge.net/)                ***"
  exit 1
fi
[ -x $XMLBIN ] || {
  echo "ERROR: could not execute xmlstarlet ($XMLBIN)"
  echo "Please provide it (http://xmlstar.sourceforge.net/)"
  exit 1
}
[ -r csv2xml.py ] || {
  echo "ERROR: csv2xml.py not found in current directory."
  echo "Please go to kiel/documents/papers/kieltex/scripts and try again."
  echo "aborting."
  exit 1
}
[ -r $CSV ] || {
  echo "ERROR: $CSV not found."
  echo "Please regenerate it from $SXC (csv, delimiter: tab)"
  echo "aborting"
  exit 1
}
[ $CSV -nt $SXC ] || {
  echo "WARNING: $CSV is older than $SXC"
  echo "maybe you should re-export it (csv format, delimiter: tab)"
  echo "continuing..."
}
echo
echo "====================="
echo " generating XML files"
echo "====================="
python csv2xml.py $CSV

echo
echo "============================"
echo " formatting generated files"
echo "============================"
for FILE in *.xml *.xsd
do
  echo -n .
  $XMLBIN ed $FILE > $FILE.new
  mv $FILE.new $FILE
done
echo "done"

echo
echo "==================================="
echo " adding RuleSets to checking files  "
echo "==================================="
patch < checking-defaults.xml.diff
patch < checking-schema.xsd.diff


echo
echo "==================================="
echo " checking validity of generated files"
echo "==================================="

for XSD in *.xsd
do
  BASE=$(basename $XSD -schema.xsd)
  $XMLBIN val -e -s $XSD $BASE-defaults.xml
done
```

## C.3.2 csv2xml.py

```
#!/usr/bin/env python
# csv2xml.py (c) 2007, Karsten Heymann
# part of the kiel.preferences module.

import sys
import csv

from elementtree import ElementTree as ET
from ElementBuilder import Element, Namespace, QName

#loglevels = "ALL DEBUG DETAIL ERROR INFO STRUCTURE".split()
loglevels = "0 1 2 3 4 5 10".split()

# mapping of datatypes. meaning of the entries:
# xml-schema-datatype, custom editing class, selector values, default value, regular expression
xsd_type_map = {
    'string':    ('string',  None,                                 []               , None),
    'int':       ('int',     None,                                 []               , None),
    'float':     ('float',   None,                                 []               , None),
    'boolean':   ('boolean', 'kiel.preferences.FixedComboBox',     ['true', 'false'], None),
    'color':     ('string',  'kiel.preferences.XColorChooser',     []               , None),
    'loglevel':  ('int',     'kiel.preferences.FixedComboBox',     loglevels        , None),
    'pair':      ('string',  None,                                 None,             '-?\\d+ -?\\d+'),
    'quadrupel': ('string',  None,                                 None,             '-?\\d+ -?\\d+ -?\\d+ -?\\d+'),
    }

# Namespace prefixes
xs=Namespace("http://www.w3.org/2001/XMLSchema",'xs')
xa=Namespace("http://www.FGolubov.com/XMLEditor",'xa')
tns_base="http://rtsys.informatik.uni-kiel.de/%7Ert-kiel/kiel/kiel/preferences/"

def generate_xsd_schema(module, data):
    tns=Namespace(tns_base+module,"tns")
    root = xs.schema("\n",
                     elementFormDefault="qualified", targetNamespace=tns.uri)
    root.attrib["xmlns:xa"]= xa.uri
    root.attrib["elementFormDefault"] = "qualified"
#    root.attrib["attributeFormDefault"] = "qualified"
    confbase = xs.element("\n",name="configuration")
    root.append(confbase)
    conftype = xs.complexType("\n")
    confbase.append(conftype)
    conflist = xs.sequence("\n")
    conftype.append(conflist)
    conftype.append(xs.attribute("\n",name="module", type="xs:string",
                                 fixed=module, use="required"))
    for property in data:
        name = property[0]
        default = property[1]
        type = property[2]
        desc = property[6]
        nodeInfo = Element('xa:node-info', Element("xa:message","-----"))
        nodeInfo.attrib["message"] = desc

        if ":" in type:
            # complex type with list of alternatives
            (type, alternatives) = type.split(":",1)
            alternatives = alternatives.split("|")
            alternatives = [ x.strip() for x in alternatives ]
        else:
            alternatives=[]
        type=type.lower()

        entry = xs.element("\n",name=name)
        conflist.append(entry)

        if type in xsd_type_map.keys():
            (xtype, xeditor, xparams, xpattern) = xsd_type_map[type]

            if xpattern:
                entry.append(
                    xs.simpleType(
                        xs.restriction({'base': 'xs:'+xtype},
                            xs.pattern(value=xpattern))))
            else:
                entry.attrib["type"] = 'xs:'+xtype

            if alternatives:
                xeditor = 'kiel.preferences.FixedComboBox'
                xparams = alternatives

            if xeditor:
                nodeInfo.attrib['editor-class'] = xeditor
                for param in xparams:
                    nodeInfo.append(Element('xa:param', param))

        else:
            print 'WARNING: %s (%s) has unhandled type )%s(' % (name, default, type)
            type = 'xs:string'

        entry.insert(0, xs.annotation("\n", xs.appinfo(nodeInfo, "\n")))

    tree = ET.ElementTree(root)
    tree.write("%s-schema.xsd" % module)

def generate_default_config(module, data):
    root=ET.Element("configuration",module=module,xmlns=tns_base+module)
    root.attrib["xmlns:xsi"]="http://www.w3.org/2001/XMLSchema-instance"
    root.attrib["xsi:schemaLocation"]=tns_base + module + " " + module+"-schema.xsd"

    for property in data:
        name = property[0]
        default = property[1]
        root.append(Element(name,default))

    tree = ET.ElementTree(root)
    tree.write("%s-defaults.xml" % module)



def main():
    data = {}
    reader = csv.reader(open(sys.argv[1], "rb"), delimiter='\t')
    # throw away heading:
    reader.next()
```

```
    for row in reader:
        (module, settings) = (row[0], row[1:])
        if not module in data.keys():
            data[module]=[settings]
        else:
            data[module].append(settings)

    for module in data.keys():
        print "Generating %s (%d properties)" % (module, len(data[module]))
        generate_xsd_schema(module,data[module])
        generate_default_config(module,data[module])

main()
```