

Comfortable Modeling of Complex Reactive Systems

Steffen Prochnow and Reinhard von Hanxleden

Department of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
University of Kiel

DATE'06, March 2006

Contents

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

Summary and Conclusions

Introduction

Motivation:

- Statecharts possess high complexity (combinations of components, dependencies, system dynamics, concurrency)
- tools for modeling Statecharts provide restricted facilities to enter and understand complex system behavior

Introduction

Motivation:

- Statecharts possess high complexity (combinations of components, dependencies, system dynamics, concurrency)
- tools for modeling Statecharts provide restricted facilities to enter and understand complex system behavior

Purpose:

- formulation of improvements for easy modeling, analyzing and understanding complex Statecharts
- establishment of these improvements in a highly configurable tool for modeling and simulation
- validation of operativeness of the tool

Introduction

Three Observations and Proposals:

- ① graphical models nice to browse, but hard to write
⇒ let the computer help more!
- ② graphical languages appealing, but not effective enough
⇒ should develop and consciously use secondary notations!
- ③ graphical languages good for understanding structures, but bad for analyzing dynamics
⇒ use dynamic charts!

Outline

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

Summary and Conclusions

The KIEL Statechart Layouter

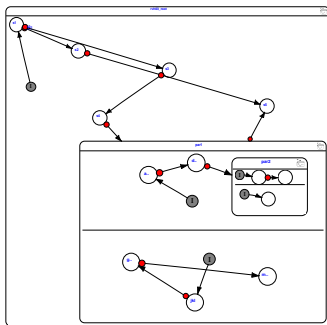
Kiel Integrated Environment for Layout

- uses several layout heuristics to choose from
 - a simple horizontal/vertical layout scheme
 - more advanced schemes, provided by GraphViz
- provides generic wrapper to create hierarchical layout from flat layout schemes
- implemented in Java
- highly configurable



GraphViz.
Graphviz—graph drawing tools.
<http://graphviz.org/>.

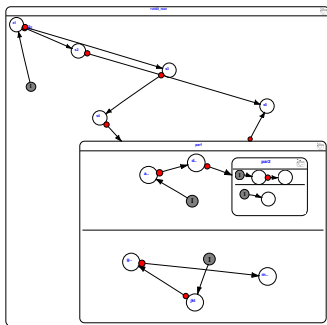
Example from KIEL



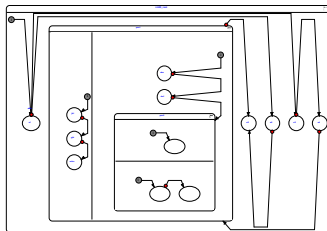
(a) Original Layout

Figure: Auto-layout from KIEL, 2003

Example from KIEL



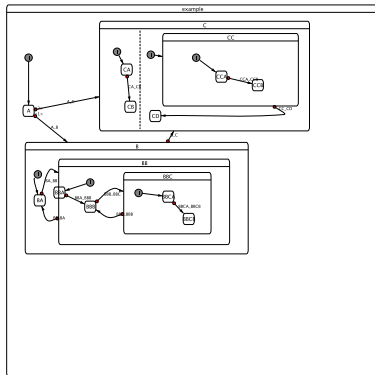
(a) Original Layout



(b) After Auto-layout

Figure: Auto-layout from KIEL, 2003

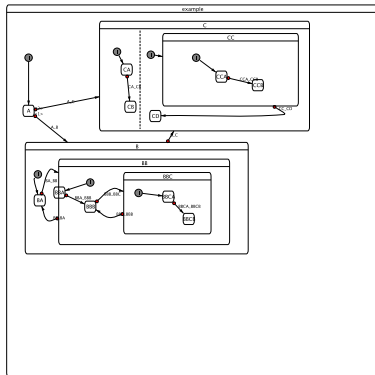
Example from KIEL



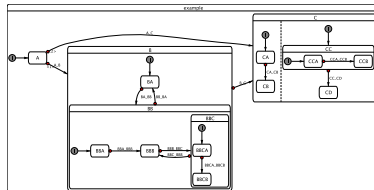
(a) Original Layout

Figure: Auto-layout from KIEL, 2006

Example from KIEL



(a) Original Layout



(b) After auto-layout

Figure: Auto-layout from KIEL, 2006

Outline

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

Summary and Conclusions

The Simulation in KIEL

- simulation step triggers the potential view change
- steps with different granularity:
 - Micro step:** stops after each elementary step computation, highlighting of associated statechart component
 - Macro Step:** accumulates micro steps of same instances
- trace playback, forward/backward simulation
- simulate Statecharts according to different semantics:
 - SSM:** internal simulator (according by André)
 - Stateflow:** using the Stateflow API



Charles André.

Semantics of S.S.M (Safe State Machine).

Technical report, I3S, Sophia-Antipolis, France, 2003.



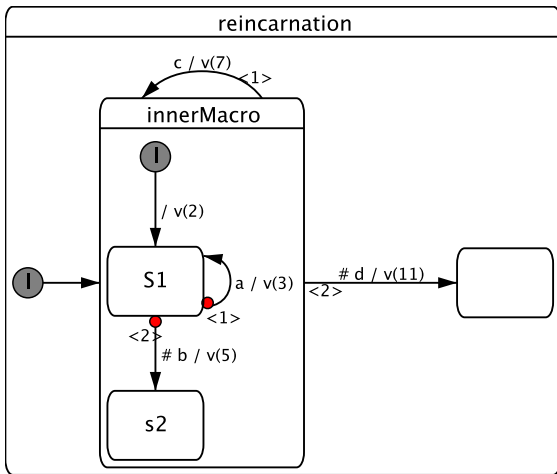
Mathworks Inc.

Stateflow Application Programming Interface, 2005.

<http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/>.

Demo: Simulation

v:integer combined with *



Visualizing Complex Behaviors

Approach:

- ① provide overview of whole system in single picture
(Deep Layout)
- ② allow level of detail to vary
- ③ Dynamic Statecharts

Dynamic Statecharts

Idea: Views should hide in-active sub-states

- present dynamically changing views dependent on
 - ① simulation state
 - ② user requests
- a dynamic extension to semantic focus-and-context representation (Köth)
- Views:
 - associated with deepest hierarchy levels of macro states
 - all simple states of this level share one view
 - each view shows complete system



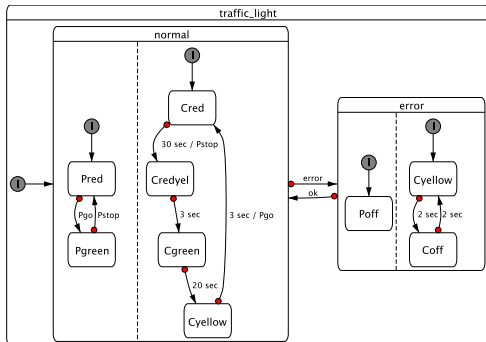
Oliver Köth.

Semantisches Zoomen in Diagrammeditoren am Beispiel von UML.

Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2001.

Demo: Example of Dynamic Statecharts

static view, after layout (deep layout):



Statistics for this example:

- 15 states
- 10 state configurations
- 2 views

Outline

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

Summary and Conclusions

Creating Graphical Models

Approaches:

- ① quick-and-dirty graphical model (WYSIWYG)
 - import from Esterel Studio, Matlab/Simulink/Stateflow
 - KIEL statechart editor
- ② textual languages
 - KIT: Statechart description language
 - Esterel

Characteristics:

- synthesize graphical model
- automated model-derivation
- configurability
- scalability
- separate content from layout (compare with \LaTeX)

Textual Languages describing Statecharts

Advantages:

- ① editing speed
- ② configuration resp. revision management (traceability)
- ③ model synthesis

Which is faster?

textual Environment:

- ① move cursor to position
- ② type `"|| await C"`

Which is faster?

textual Environment:

- 1 move cursor to position
- 2 type "|| await C"

graphical Environment:

- 1 make room: shift neighbor states, enlarge parent state
- 2 click on "add state"
- 3 move mouse to location and place new state
- 4 click on "add state"
- 5 move mouse to location and place new state
- 6 double click on new state, toggle terminal field
- 7 click on "initial state"
- 8 move mouse to location and place new initial state
- 9 click on "transition"
- 10 move mouse to location of initial state
- 11 press left mouse button and keep pressed until reaching state
- 12 click on "transition"
- 13 move mouse to location of state
- 14 press left mouse button and keep pressed until reaching terminal state
- 15 double click on transition
- 16 write "C" in trigger field
- 17 press "OK"
- 18 click on "delimiter line"
- 19 move mouse to location and place delimiter line

Which is traceable?

diff file_{ABRO} file_{ABRO'}

textual: only 4 lines

```
8c8
< [ await A || await B || await C ];
---
> [ await A || await B ];
```

graphical: 12 of 287 lines

```
1c1
< # Model of type Document saved by /home/esterel/EsterelStudio-5.2/bin/estudio.exe
[11/18/2005 10:39:01]
---
> # Model of type Document saved by /home/esterel/EsterelStudio-5.2/bin/estudio.exe
[11/18/2005 10:40:03]
161c161
< {115
---
> {295
227c227
< AT 107 145
---
> AT 197 145
```

KIT: Statechart Description Language

- KIT: **KIEL** statechart extension of do**T**
- describes topological statecharts structure
- extensible superset of known statechart dialects
- extends the dot specification language by all statechart specific components:
 - signals/events, variables
 - state properties
 - pseudostates
 - transition properties

KIEL using KIT

- easy transformation using java parser generator
- synthesizing statechart layout
- model according component representation

KIT: Statechart Description Language

- KIT: **KIEL** statechart extension of do**T**
- describes topological statecharts structure
- extensible superset of known statechart dialects
- extends the dot specification language by all statechart specific components:
 - signals/events, variables
 - state properties
 - pseudostates
 - transition properties

KIEL using KIT

- easy transformation using java parser generator
- synthesizing statechart layout
- model according component representation

Demo

Outline

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

Summary and Conclusions

Summary

The KIEL Prototype

- automatic layout of Statecharts
- several layout heuristics

Summary

The KIEL Prototype

- automatic layout of Statecharts
- several layout heuristics
- interfaces to Esterel Studio and Stateflow
- supports dynamic Statecharts

Summary

The KIEL Prototype

- automatic layout of Statecharts
- several layout heuristics
- interfaces to Esterel Studio and Stateflow
- supports dynamic Statecharts
- easy textual modeling
- transformation of Esterel to SSM
- representation and simulation of Statecharts according to miscellaneous modeling tools

Summary

The KIEL Prototype

- automatic layout of Statecharts
- several layout heuristics
- interfaces to Esterel Studio and Stateflow
- supports dynamic Statecharts
- easy textual modeling
- transformation of Esterel to SSM
- representation and simulation of Statecharts according to miscellaneous modeling tools
- has been used successfully in teaching “System Modeling and Synchronous Languages”

Outlook on KIEL

- examine further layout schemes
- refine secondary notations for Statecharts (et al.)
- checking of syntactical/semantical properties (subsetting of Statecharts)
- cognitive experiments

thanks!

questions or comments?

Appendix: Secondary Notation

(see presentation of Reinhard von Hanxleden at SYNCHRON'03)

- Typically not part of notation
- Provide additional hints to reader
 - Adjacency
 - Clustering
 - White space
 - Labeling . . .
- Effectively result in language sub-setting

Appendix: Example of poor SN

```
while ((used!=1) || (a[0] !=1)) { if (a[0] & 0x1)
{ k=1; for (c = 0; c <= used; c++)
{ a[c] = 3 * a[c] + k; k = a[c] / 10; a[c] = a[c] % 10;}
if (a[used]) { used++; if (used >= 72)
{ printf ("Run out of space\n"); exit(1);}} }
else {k = 0; for (c = used - 1; c >= 0; c--)
{ a[c] = a[c] + 10*k; k = a[c] & 0x1; a[c] = a[c] >>1;}
if (a[used - 1] == 0) used--; }count++; }
```

Appendix: Example of better SN

```
while ((used!=1) || (a[0] != 1)) {
  if (a[0] & 0x1) {
    k=1;
    for (c = 0; c <= used; c++) {
      a[c] = 3 * a[c] + k;
      k = a[c] / 10;
      a[c] = a[c] %10;
    }
    if (a[used]) {
      used++;
      if (used >= 72) {
        printf ("Run out of space\n");
        [...]
      }
    }
  }
}
```

Appendix: The Proposal

- ① Develop catalogue of efficient secondary notations for Statecharts (Style Guide, Normal Forms)
- ② Provide support for conformance checking (Style Checker)
- ③ Provide support for generating conformant diagrams (Pretty Printer)

Appendix: Secondary Notations for Statecharts

Placement of initial and final state

Goal: Aid identification of initial/final state

Example: Top/left, bottom/right, respectively

Placement of remaining states

Goal: Support understanding of state sequencing

Example: Minimize back transitions

Shape of transitions

Goal: State sequencing; prominent source/sink states

Example: Clock-wise orientation

Appendix: Secondary Notations for Statecharts

Placement of labels

Goal: Easy matching of labels and transitions

Example: Left of transition, relative to direction

Exploitation of symmetry

Goal: Highlight design regularities

Example: parallelism

Appendix: Secondary Notation in KIEL

- Place initial states top/left
- Place final states bottom/right
- Clock-wise orientation of transitions
- Consistent placement of labels
- Try to put successive states adjacently
- minimize back transitions

Appendix: More Information

- KIEL URL:
`http://www.informatik.uni-kiel.de/~rt-kiel`