# KIEL

## Textual and Graphical Representations of Statecharts

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

Steffen Prochnow, Claus Traulsen, Reinhard von Hanxleden

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

SYNCHRON'05, November 2005

# Contents

# Introduction

**Motivation of the project:**

- 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 of the project:**

- 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 of the project:**

- 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
   $\Longrightarrow$ let the computer help more!
2. graphical languages appealing, but not effective enough
   $\Longrightarrow$ should develop and consciously use secondary notations!
3. graphical languages good for understanding structures, but bad for analyzing dynamics
   $\Longrightarrow$ use dynamic charts!

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

# Outline

Introduction

Statechart Layout

Visualizing Complex Behaviors

Creating Graphical Models

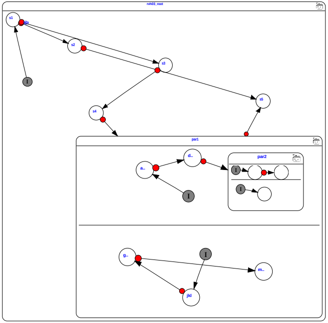
Summary and Conclusions

# The KIEL Statechart Layouter

**K**iel **I**ntegrated **E**nvironment for **L**ayout

- 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

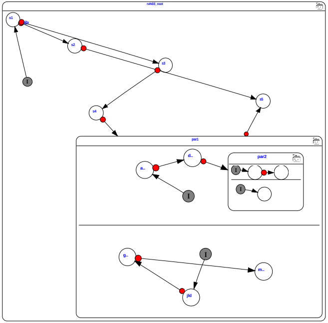
# Example from KIEL



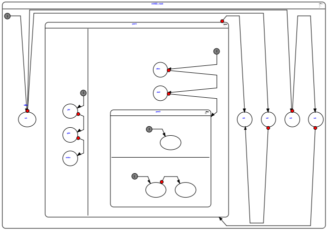

(a) Original Layout

Figure: Auto-layout from KIEL, 2003

# Example from KIEL



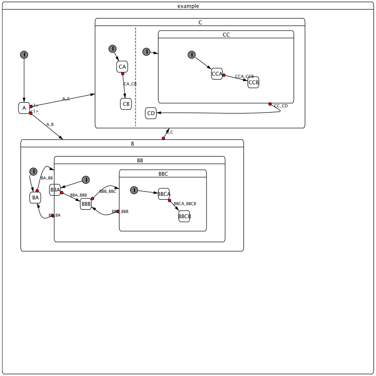

(a) Original Layout



(b) After Auto-layout

Figure: Auto-layout from KIEL, 2003

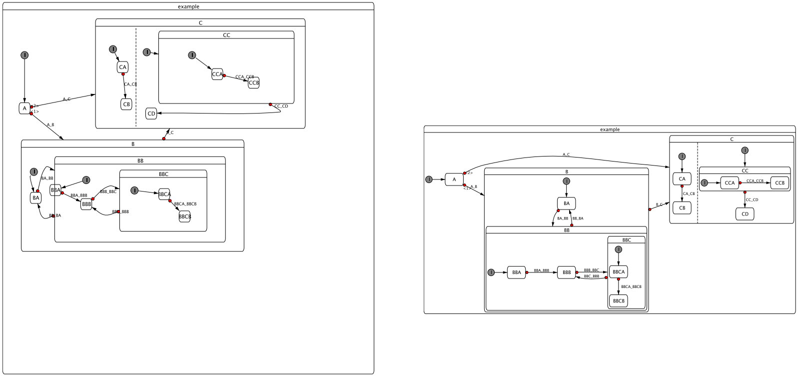
# Example from KIEL




(a) Original Layout

Figure: Auto-layout from KIEL, 2005

# Example from KIEL



(a) Original Layout

(b) After auto-layout

Figure: Auto-layout from KIEL, 2005

# 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

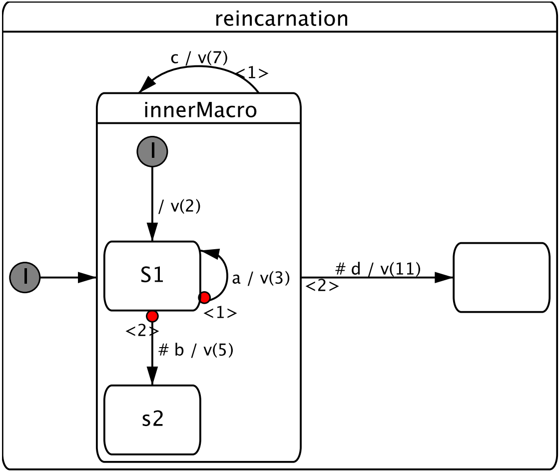
  **Macro Step:** accumulates micro steps of same instances
- trace playback, forward/backward simulation
- simulate Statecharts according to different semantics:

  **SSM:** internal simulator (according by André)

  **Stateflow:** using the Stateflow API

# 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

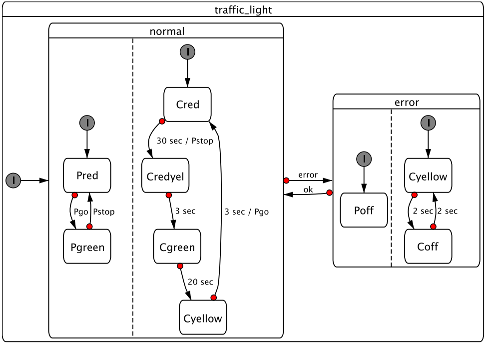
# Dynamic Statecharts

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

# 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:**

❶ move cursor to position

❷ 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 $\texttt{file}_{\text{ABRO}}$ $\texttt{file}_{\text{ABRO}'}$

**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: **KI**EL 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: **KI**EL 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

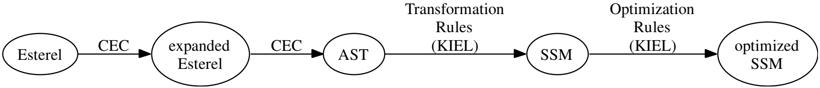
KIEL using KIT

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

Demo

# Esterel in KIEL

The Transformation:

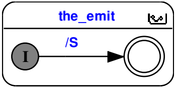


# Production Rules

**The emit statement**

| |
|---|
| `emit` $S$ |

$\Longrightarrow$

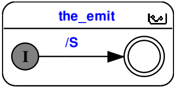


# Production Rules

## The emit statement

| emit $S$ |
|---|

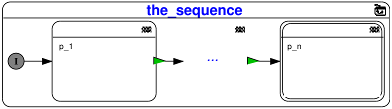
$\Longrightarrow$




## The sequence statement

| $p_1 ; \ldots ; p_n$ |
|---|

$\Longrightarrow$

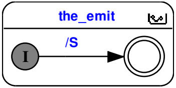


# Production Rules

## The emit statement

| |
|---|
| emit $S$ |

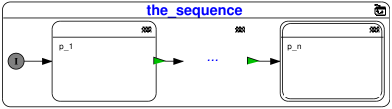
$\Longrightarrow$




## The sequence statement

| |
|---|
| $p_1 ; \ldots ; p_n$ |

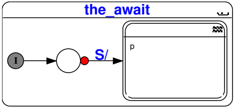
$\Longrightarrow$




## The await statement

| |
|---|
| await $S$ do $p$ end |

$\Longrightarrow$

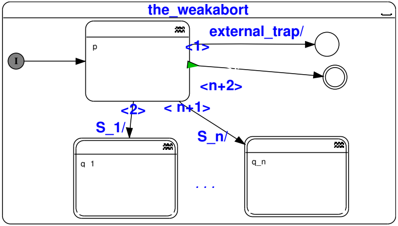


# Production Rules

**The weak abort statement**

```
weak abort p when
    case S_1 do q_1
    ⋮
    case S_n do q_n
end abort
```

⟹




+19 further rules

# Optimization Rules

Motivation

- automatic synthesis produces "verbose" modules
- however, also human modelers (esp. novices) may produce sub-optimal models

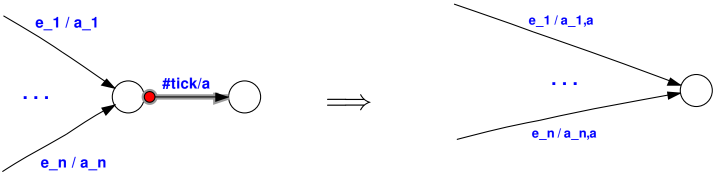
Note: what an optimal model is might be a matter of style, but automatic optimization rules can lead to a more consistent modeling style.

In total only five kinds of rules

- flatten hierarchy
- remove simple states
- remove conditional states
- combine terminal states
- remove normal termination

# Optimization Rules

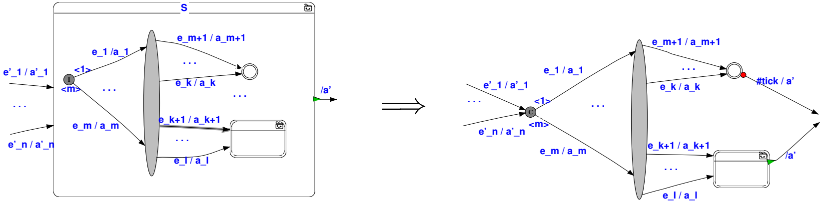
## Remove Simple States




Applicable for transient states

# Optimization Rules

## Flatten Hierarchy




Applicable if

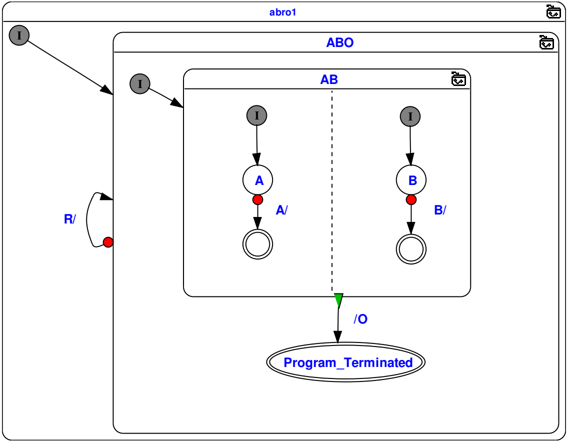
- no abort originate from S
- S has no local signals

# Transformation Example (Roundtrip)

$$\text{SSM} \xrightarrow[\text{Esterel Studio}]{\text{Generation}} \text{Esterel} \xrightarrow[\text{KIEL}]{\text{Transformation}} \text{SSM}' \xrightarrow[\text{KIEL}]{\text{Optimization}} \text{SSM}''$$

# Transformation Example (Roundtrip)

SSM $\xrightarrow[\text{Esterel Studio}]{\text{Generation}}$ Esterel $\xrightarrow[\text{KIEL}]{\text{Transformation}}$ SSM$'$ $\xrightarrow[\text{KIEL}]{\text{Optimization}}$ SSM$''$



# Transformation Example (Roundtrip)

SSM —Generation / Esterel Studio→ Esterel —Transformation / KIEL→ SSM′ —Optimization / KIEL→ SSM″

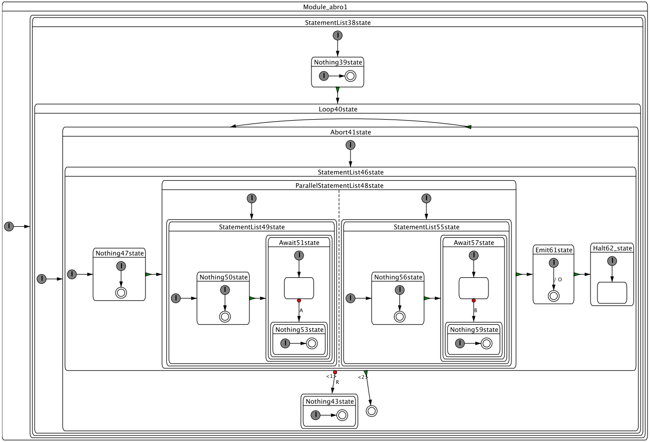
```
module abro1:
input A  ;
input B  ;
input R  ;
output O  ;

nothing;
loop
 % state ABO
 abort
  nothing;
  % state AB
  [
   nothing;
   % state A
   await case [A] do
   nothing
   end await
```

⋮

```
   ||
    nothing;
    % state B
    await case [B] do
    nothing
    end await
   ];
   emit O;
   halt
  when [R]
  do
   nothing
  end abort
 end loop
end module
```

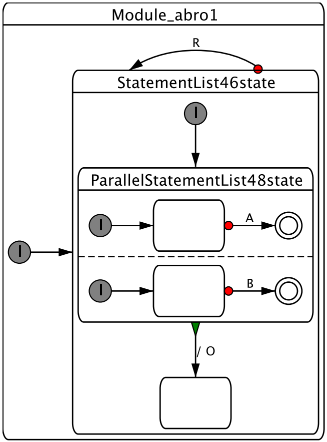
# Demo: Transformation Example (Roundtrip)

$$\text{SSM} \xrightarrow[\text{Esterel Studio}]{\text{Generation}} \text{Esterel} \xrightarrow[\text{KIEL}]{\text{Transformation}} \text{SSM}' \xrightarrow[\text{KIEL}]{\text{Optimization}} \text{SSM}''$$



# Demo: Transformation Example (Roundtrip)

SSM $\xrightarrow[\text{Esterel Studio}]{\text{Generation}}$ Esterel $\xrightarrow[\text{KIEL}]{\text{Transformation}}$ SSM′ $\xrightarrow[\text{KIEL}]{\text{Optimization}}$ SSM″



# Testing by Roundtrip

has been done for all basic blocks

Drawbacks

- does still not assure that *all* programs are transformed correctly
- relies one the correctness of the transformation from SSMs to Esterel

# A Formal Proof

Idea:

- consider a state based semantics for Esterel (e. g. Tardieu)
- each simple state of a stable configuration matches a pause in the Esterel program.
- use this to define a simulation relation between states of the Esterel program and stable states of the derived chart
- show by structural induction, that this relation is a bisimulation

$\leadsto$ hence the observable behavior is the same

# A Formal Proof

Problems:

- parts of SSM lack (to our knowledge) a nice formalization
- extend the formal definition to valued signals, history, ...
- the proof itself is not hard but cumbersome (and has still do be done)

- traps need special treatment, because they have to be expressed by abort in SSM. This can be done on the Esterel level before the transformation. (suggested by Klaus Schneider)

# Outline

# 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”
- see also DATE’06 publication on KIEL

# Outlook on KIEL

- examine further layout schemes
- refine secondary notations for Statecharts (et al.)
- checking of syntactical/semantical properties (subsetting of Statecharts)
- prove equivalent behavior of synthesized SSMs
- 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