Statechart Development Beyond WYSIWYG

Steffen Prochnow and Reinhard von Hanxleden

Department of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
Christian-Albrecht-Universität zu Kiel

MODELS'07, Nashville TN
September 2007

Contents

Introduction

Motivation

- Realistic Statecharts possess high complexity
- Construction, modification, and revision management of graphical models are burdensome
- Graphical WYSIWYG editors limit the productivity editing Statecharts

Introduction

Motivation

- Realistic Statecharts possess high complexity
- Construction, modification, and revision management of graphical models are burdensome
- Graphical WYSIWYG editors limit the productivity editing Statecharts

Purpose

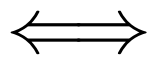
- Analysis of the graphical editing process
- Identification of generic Statecharts editing patterns
- Development of alternative and progressive Statechart construction paradigms
- Establishment of this paradigms in a highly configurable tool
- Validation of operativeness of the tool

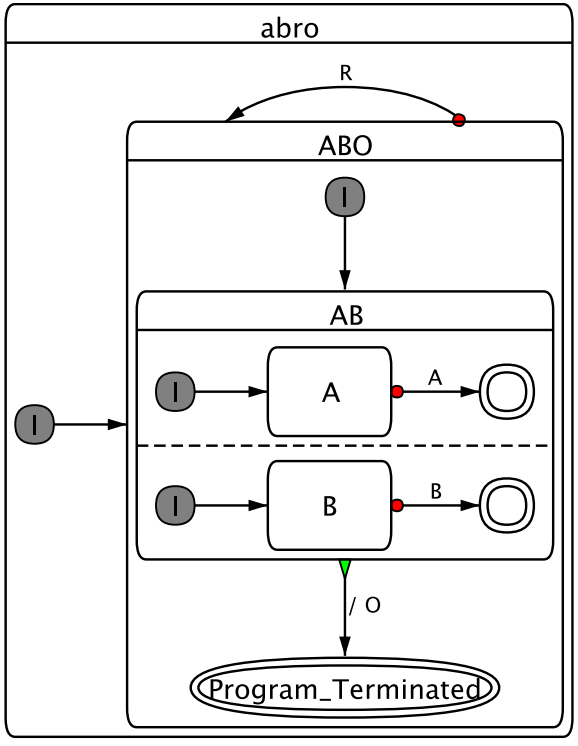
Example: ABRO/ABCRO

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

Example: ABRO/ABCRO

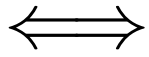
```
module ABRO:
input A, B, R;
output O;
loop
  [
    await A
  ||
    await B
  ];
  emit O;
each R
end module
```

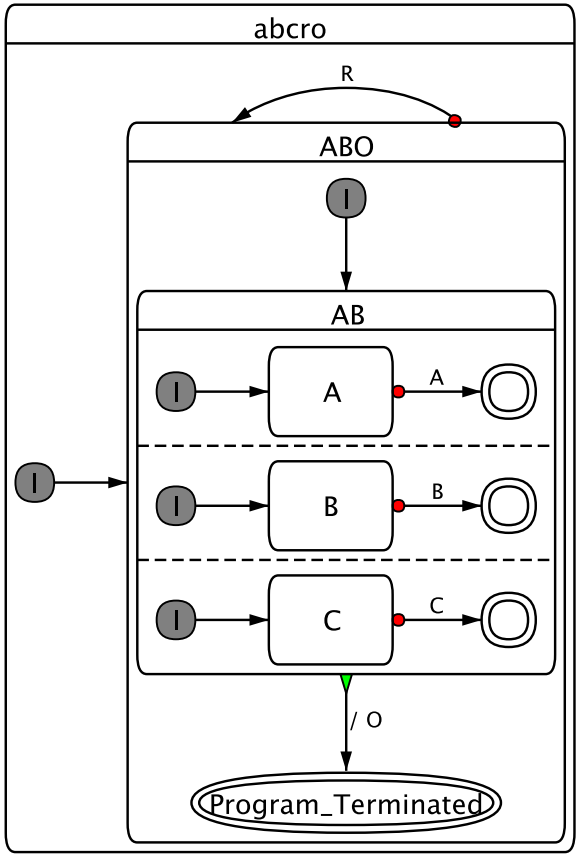




Example: ABRO/ABCRO

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





Which is faster?

Editing Esterel:

1. move cursor to position
2. type ", C"
3. move cursor to position
4. type "`|| await C`"

Which is faster?

Editing Esterel:

1. move cursor to position
2. type ", C"
3. move cursor to position
4. type "`|| await C`"

Editing Safe State Machine:

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 more tracable?

Textual:

```
1,2c1,2
< module ABRO:
< input A, B, R;
---
> module ABCRO:
> input A, B, C, R;
5c5
< [ await A || await B ];
---
> [ await A || await B || await C
      ];
```

Which is more tracable?

Textual:

```
1,2c1,2
< module ABRO:
< input A, B, R;
---
> module ABCRO:
> input A, B, C, R;
5c5
< [ await A || await B ];
---
> [ await A || await B || await C
      ];
```

Graphical:

```
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
243c243
< [E]
---
> [V105 E E]
344a345,519
> NODE init.2    init
> ATTRIB
> {[]
> ""
>
> {0
> 0
> }
> {
> {"helvetica"
> 12
> 1
> "Blue"
> }
> {"helvetica"
> 12
> 1
> "Blue"
> }}<false>
> }
> AT    170 35
> END # of init.2
>
> NODE state.4 state
> ATTRIB
> {""
> <halt>
> <no>
> ""
>
> {30
> 30
> 0
> 0
> 1
> "1 0 0 1 0 0"
> 0
> 0
> 0
> 0
> 0
> 0
> "1 0 0 1 0 0"
> }
> {[]
> []
> []
> }
> {[]
> []
> []
> }
> {<false>
> ""
> }
> {
> {"helvetica"
> 12
> 1
> "Blue"
> }
> {"Black"
> 1
> "none"
> 0
> }"White"
>
> {"helvetica"
> 12
> 4
> "Blue"
> }
> {"helvetica"
> 12
> 4
> "Blue"
> }
> {"helvetica"
> 12
> 4
> "Blue"
```

(Only first 100 of 287 lines shown)

Outline

Creating Graphical Models

- WYSIWYG editors to create graphical models
- Some editors offer alignment tools
- Creating graphical models slow
- In initial phase, often resort to paper and pencil
- Maintaining graphical models time consuming

Creating Graphical Models

> *I quite often spend an hour or two just moving boxes and wires around, with no change in functionality, to make it that much more comprehensible when I come back to it.*
>
> One practitioner, according to Petre

Marian Petre.
Why looking isn't always seeing: readership skills and graphical programming.
Communications of the ACM, 38(6):33–44, June 1995.

What is the Problem?

Non-linearity

- Text: 1-D
- Graphics: 2-D

What is the Problem?

Non-linearity

- Text: 1-D
- Graphics: 2-D

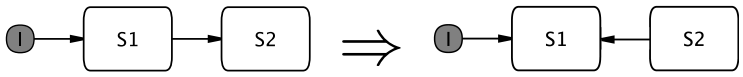
Context entanglement

- Transitions
- State hierarchy, concurrency

Editing Schemata

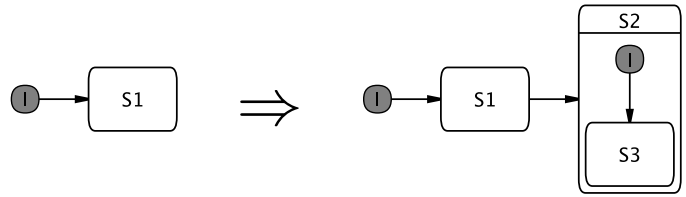
- Nine main editing schemata of categories:
  - *Creation*, *modification*, and *deletion* of Statechart elements

(a) Insertion of a simple successor state.

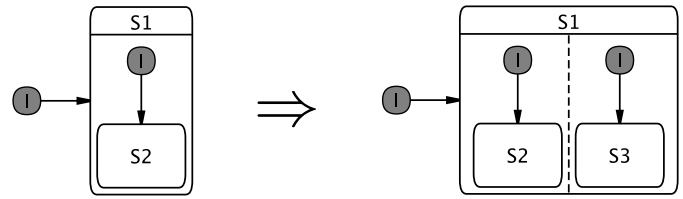


(b) Modification of transition direction.

(c) Deletion of a Statechart element.



(d) Insertion of hierarchical successor state.



(e) Insertion of a parallel region.

Figure: Exemplary generic editing schemata derived from a typical editing process using WYSIWYG editors.

Editing Action Sequence

1. If needed, create free space.
2. Focus on a Statechart element for modification resp. supplementation.
3. Apply an editing schema.
4. If needed, rearrange the modified chart to improve readability.

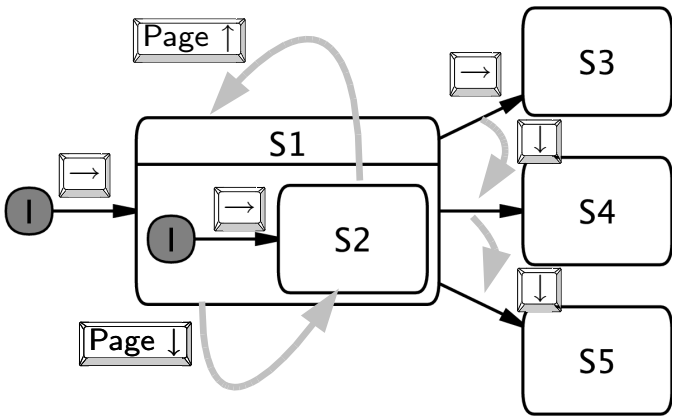
Editing Action Sequence

1. If needed, create free space.
2. Focus on a Statechart element for modification resp. supplementation.
3. Apply an editing schema.
4. If needed, rearrange the modified chart to improve readability.

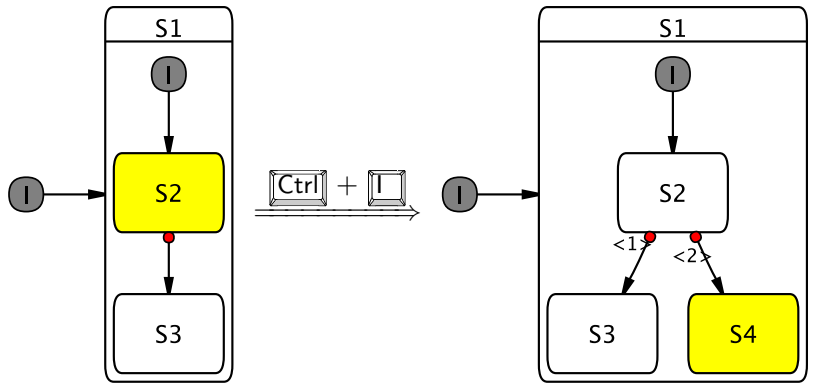
Enhancements in Statechart Editing

1. Quick-and-dirty graphical model (WYSIWYG)
   - After creation: automated layout of graphical elements
2. Macro-based modeling
   - Direct manipulation of Statechart structure using keyboard macros
   - Employs Statechart production rules
   - Ensures syntax-consistency
   - Automated layout of graphical elements
3. Text-based modeling
   - Modeler creates only Statechart structure
   - Automated placement of graphical elements
   - Separate content from layout

Macro-Based Modeling



(a) Navigation with key strokes.



(b) Example of applying the "insert simple successor state" schema.

Figure: Editing actions and navigation using the macro-based modeling approach.

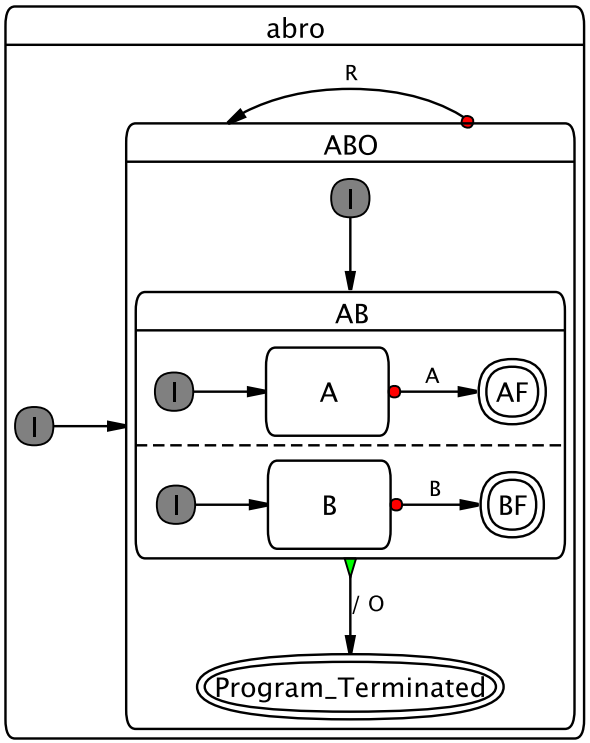
Text-Based Modeling

KIel Statechart extension of doT:

- Implicit declarations in *dot*,
- Hierarchy construction in *Argos*,
- Orthogonal construction in *Esterel*, and
- Ability to describe different Statechart dialects

```
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"];
 };
};
```

(a) KIT description representation.



(b) SSM representation.

Outline

The KIEL Modeling Tool

Kiel Integrated Environment for Layout

The KIEL Modeling Tool

Kiel Integrated Environment for Layout

- KIEL macro editor and the KIT editor are implemented
- Element placement: automatic layout of Statecharts
- Several layout heuristics

The KIEL Modeling Tool

Kiel Integrated Environment for Layout

- KIEL macro editor and the KIT editor are implemented
- Element placement: automatic layout of Statecharts
- Several layout heuristics

Beyond:

- Interfaces to Esterel Studio, ArgoUML, Stateflow
- Convert textual Esterel to graphical SyncCharts

The KIEL Modeling Tool

Kiel Integrated Environment for Layout

- KIEL macro editor and the KIT editor are implemented
- Element placement: automatic layout of Statecharts
- Several layout heuristics

  Beyond:
- Interfaces to Esterel Studio, ArgoUML, Stateflow
- Convert textual Esterel to graphical SyncCharts
- Micro-step simulation and visualization
- Supports dynamic Statecharts
- Support for robustness analysis
- URL: `http://rtsys.informatik.uni-kiel.de/~rt-kiel`

Screen-shot

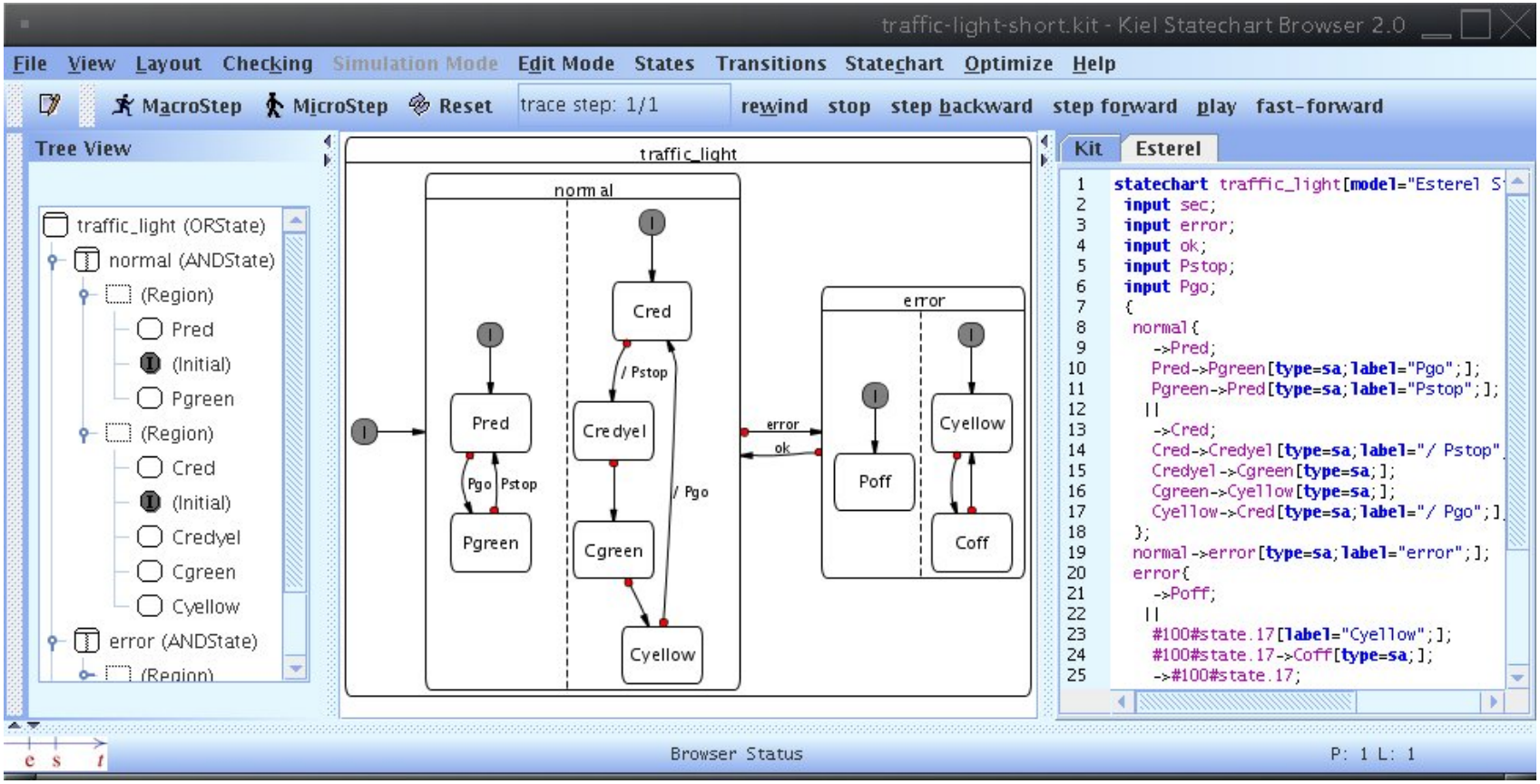


Figure: Screenshot of KIEL displaying the Statechart tree-structure, the graphical model, and the KIT editor.

Outline

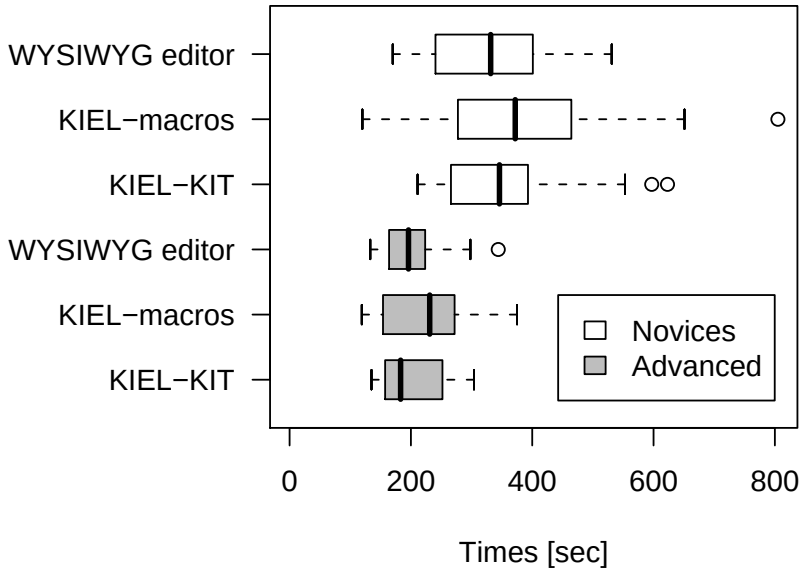
The Experiment

- Goals:
  1. Investigation of differences in editing using an WYSIWYG Statechart editor, the KIEL macro editor, and the KIT editor
  2. Comparison of the readability of Statechart layouts created by the KIEL layouter and other Statechart layouts
- Two groups:
  1. Novices: basic knowledge concerning Statecharts
  2. Advanced: practical experiences modeling Statecharts (using co-notation)
- Subjects: 24 resp. 19 graduate-level students attending the lecture "Model-Based Design and Distributed Real-Time Systems"

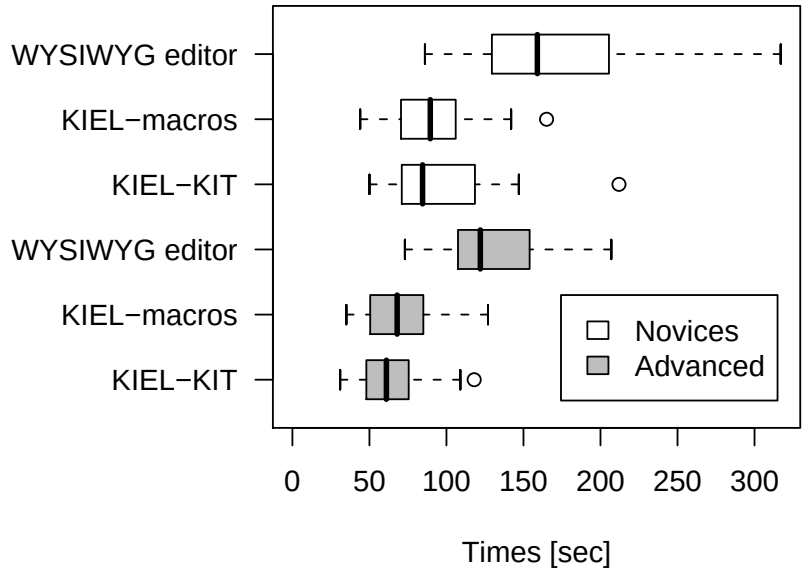
Hypothesis

1. *Statechart Creation:*
   - Novices will need less time to create a Statechart using the WYSIWYG editor.
   - Advanced will need less time using the KIT editor.
2. *Statechart Modification:* Statechart modification using the KIT editor or the KIEL macro editor is faster than using the WYSIWYG editor.
3. *Aesthetics:* We expect the best scores for Statecharts laid out according certain layout styles realized by the KIEL Statechart layouter.
4. *Comprehension:* Well arranged Statecharts laid out by the KIEL Statechart layouter are faster understandable.

Results (Editing)

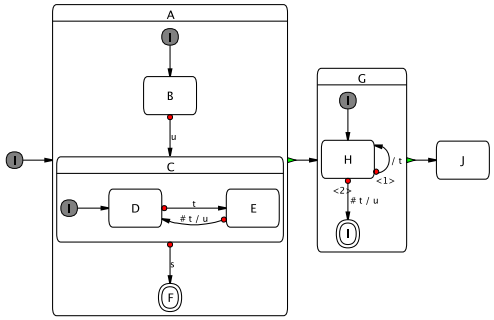


(a) Distribution of times for creating a new Statechart.

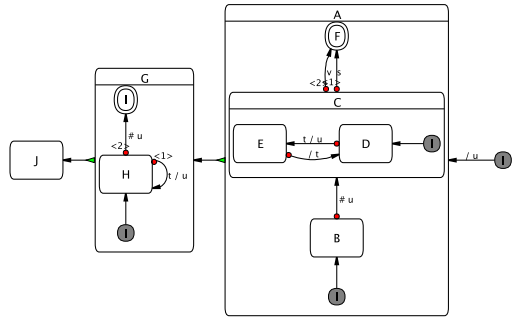


(b) Distribution of times for modifying an existing Statechart.

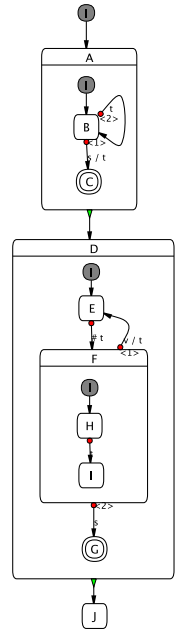
Statechart layouts



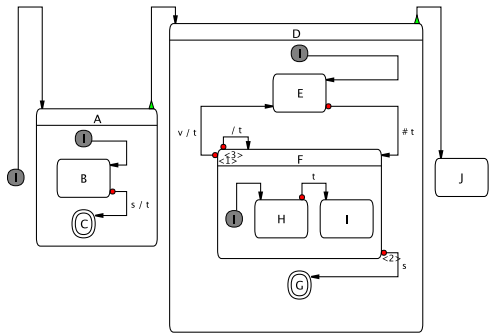
(a) Alternating dot layout (ADL).



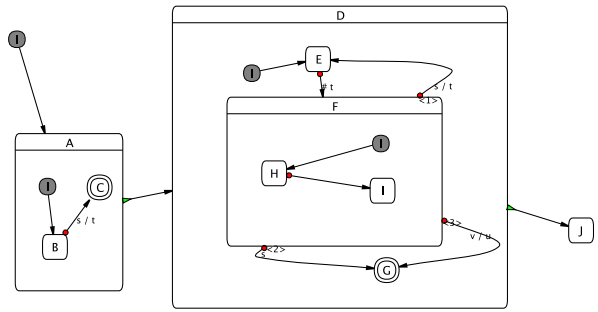
(b) ADL backwards (ADBL).



(c) Linear layer layout (LLL).

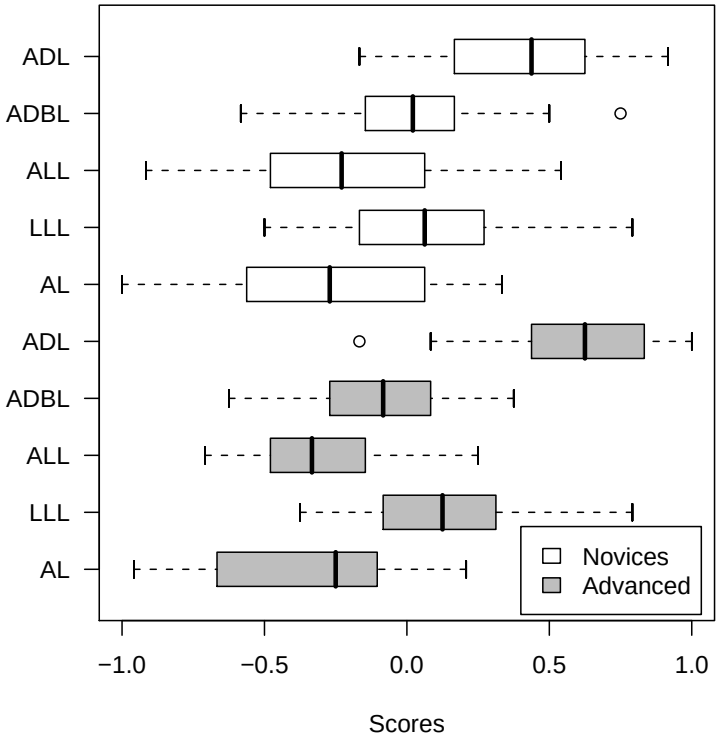


(d) Alternating linear layout (ALL).

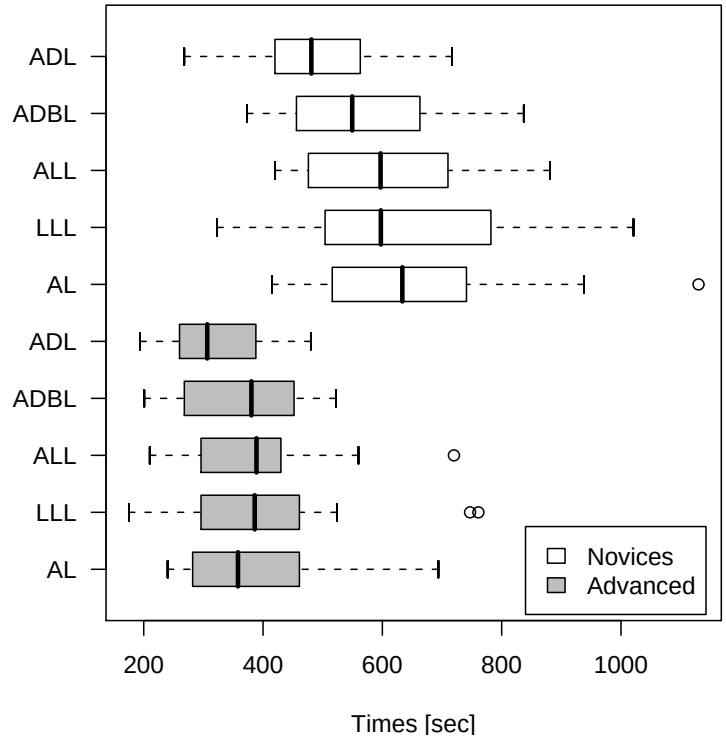


(e) Arbitrary layout (AL).

Results (Layout)



(a) Distribution of subjective Statechart layout scores.



(b) Distribution of Statechart comprehension times.

Outline

Summary and Further Work

Summary

- Alternatives to WYSIWYG graphical modeling techniques
  - Language KIT to describe topological Statechart structures
  - Macro-based modeling using Statechart production rules to ensure syntax-consistency
- Use of an automated layout method to arrange Statecharts
- Has been used successfully in teaching "System Modeling and Synchronous Languages"
- Experimental evaluation supports
  - Usability of the editing methods and
  - Readability of computed Statechart layouts

Summary and Further Work

Summary

- Alternatives to WYSIWYG graphical modeling techniques
  - Language KIT to describe topological Statechart structures
  - Macro-based modeling using Statechart production rules to ensure syntax-consistency
- Use of an automated layout method to arrange Statecharts
- Has been used successfully in teaching "System Modeling and Synchronous Languages"
- Experimental evaluation supports
  - Usability of the editing methods and
  - Readability of computed Statechart layouts

Further Work

- Simultaneous display: indexing mechanism between editing views
- Apply graphical model synthesis, layout and simultaneous display to data-flow languages (e. g. Matlab SIMULINK, SCADE/LUSTRE)

Summary and Further Work

Summary

- Alternatives to WYSIWYG graphical modeling techniques
  - Language KIT to describe topological Statechart structures
  - Macro-based modeling using Statechart production rules to ensure syntax-consistency
- Use of an automated layout method to arrange Statecharts
- Has been used successfully in teaching "System Modeling and Synchronous Languages"
- Experimental evaluation supports
  - Usability of the editing methods and
  - Readability of computed Statechart layouts

Further Work

- Simultaneous display: indexing mechanism between editing views
- Apply graphical model synthesis, layout and simultaneous display to data-flow languages (e. g. Matlab SIMULINK, SCADE/LUSTRE)

thanks!
questions or comments?

Appendix: Implementation

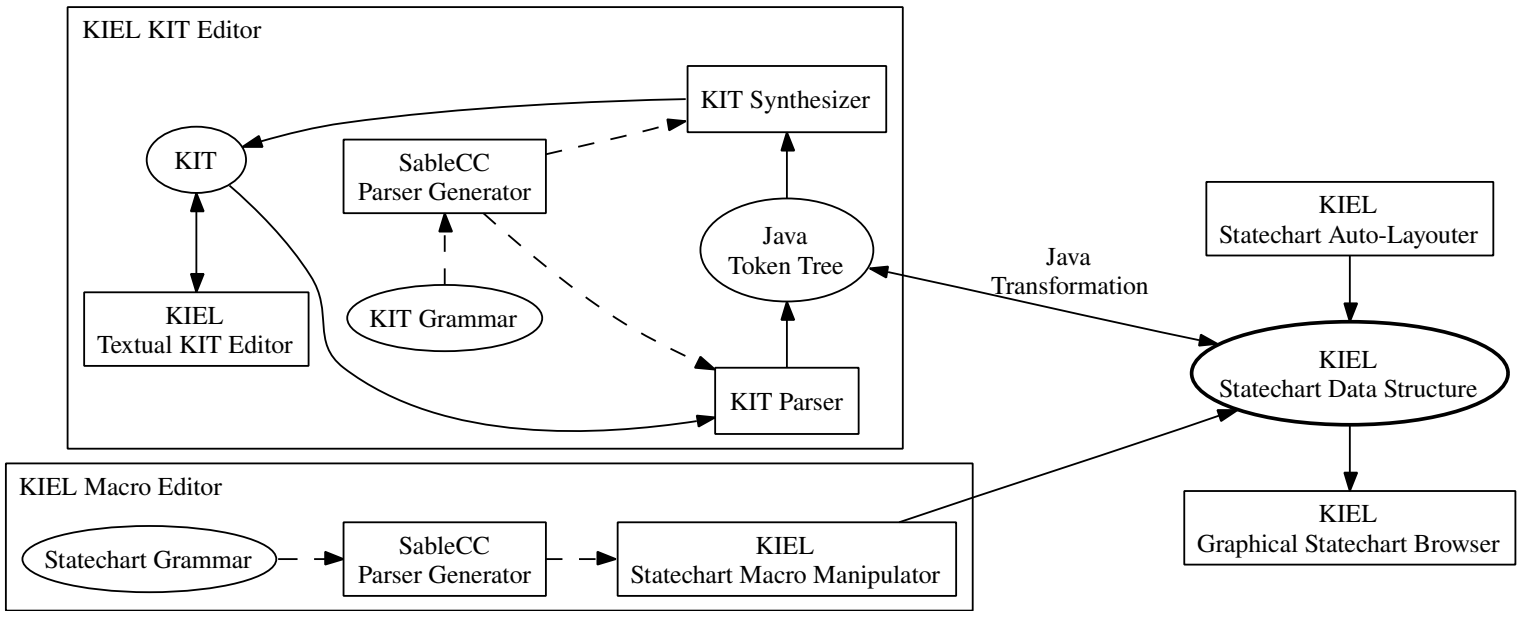


Figure: Integration of the KIT editor and the KIEL macro editor into KIEL. The solid lines characterize the information flow during runtime, the dashed lines represent dependencies during compile-time of KIEL.