

Light-Weight Synthesis of Ptolemy Diagrams with KIELER

Ulf Rüegg, Christian Schneider, Christoph Daniel Schulze,
Miro Spönemann, Christian Motika, and Reinhard von Hanxleden

Real-Time Systems and Embedded Systems Group
Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Germany



10th Ptolemy Miniconference
Berkeley, 7 Nov. 2013

Ptolemy & KIELER

file: D:/Studium_GIT/papers/ptolemy13/talk/demo/BrockAckerman.moml

File View Edit Graph Debug Help

Find:

Library Tree

- Utilities
- Directors
- Actors
- MoreLibraries
- UserLibrary

PN Director Author: Edward A. Lee

This model illustrates the well-known Brock-Ackerman anomaly. The two composite actors ActorA and ActorB implement exactly the same (nondeterministic) input/output relation. That is, given any two input sequences at the two input ports, the possible output sequences from each actor are the same. However, when wired as shown into two feedback loops, the two actors do not behave the same way. In particular, the upper feedback loop has more possible outputs than the bottom one.

Starver Expression in + 1 ActorA Display

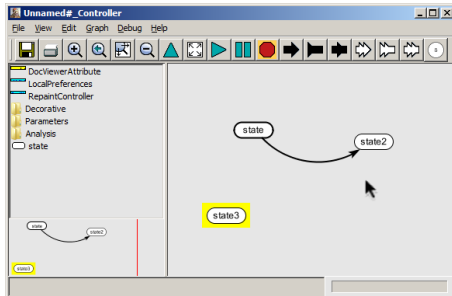
Ramp ActorB Display2

Expression2 in + 1 Starver2

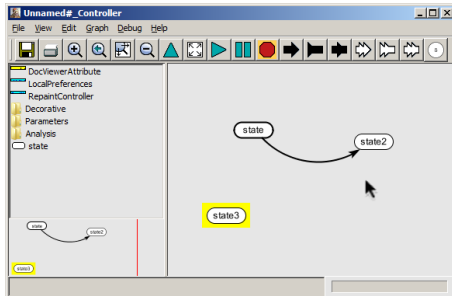
KIELER layout done in 397ms (Graph conversion 247ms, Algorithm 88ms, MoMLChanges 48ms).

Graphical Modeling

- ▶ 😊 Short learning curve
 (palette, Drag&Drop)

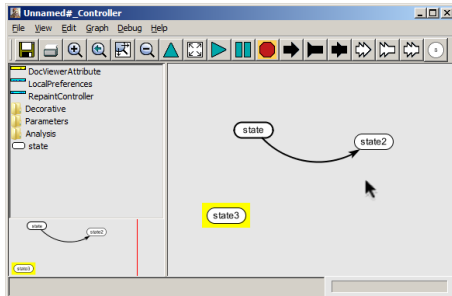


Graphical Modeling



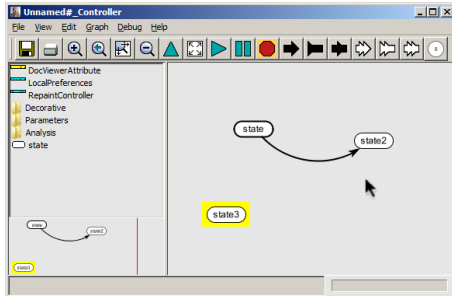
- ▶ 😊 Short learning curve
(palette, Drag&Drop)
- ▶ 😊 Readability
(inspecting a graphical model, mental map)

Graphical Modeling



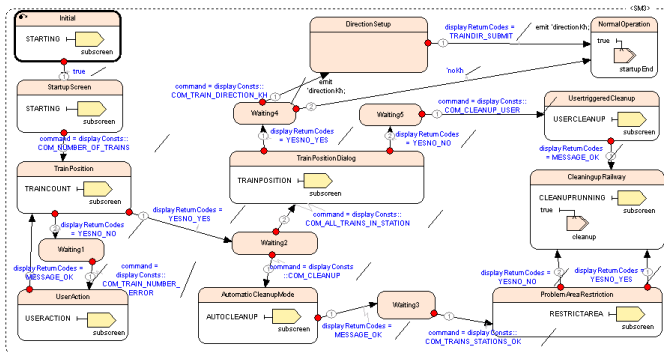
- ▶ 😊 **Short learning curve**
(palette, Drag&Drop)
- ▶ 😊 **Readability**
(inspecting a graphical model, mental map)
- ▶ 😊 **Visualization of dynamics: Simulation**

Graphical Modeling



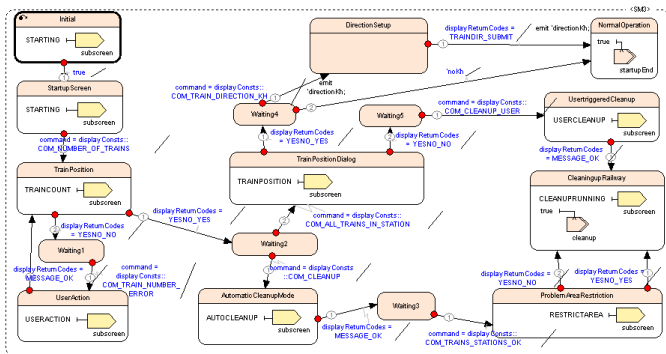
- ▶ 😊 Short learning curve (palette, Drag&Drop)
- ▶ 😊 Readability (inspecting a graphical model, mental map)
- ▶ 😊 Visualization of dynamics: Simulation
- ▶ 😊 Validation (detect obvious model errors)

Graphical Modeling (cont'd)



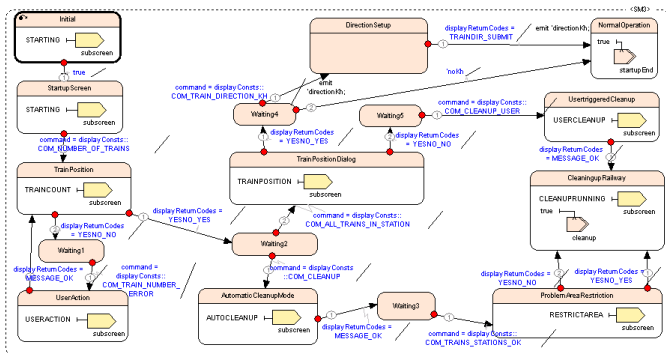
- ▶ Widely used in today's industrial tool chains (e.g., SCADE) and academia (e.g., Ptolemy)

Graphical Modeling (cont'd)



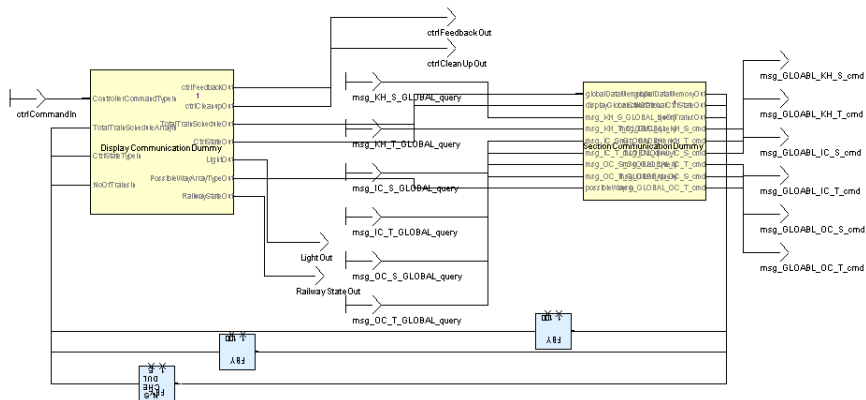
- ▶ Widely used in today's industrial tool chains (e.g., SCADE) and academia (e.g., Ptolemy)
- ▶ ☹ **Readability** (overview gets lost quickly)

Graphical Modeling (cont'd)



- ▶ Widely used in today's industrial tool chains (e.g., SCADE) and academia (e.g., Ptolemy)
- ▶ ☹ **Readability** (overview gets lost quickly)
- ▶ ☹ **Maintenance** (requires lots of manual effort)

Graphical Modeling (cont'd)



► ☹ **Readability** (layout critical for understanding semantics)

Graphical Modeling (cont'd) - Model Browsing

The screenshot shows the KIELER IDE interface for a traffic light control system model. The main window displays a hierarchical model browser with the following components:

- SR Director**: A green rectangular component.
- Clock**: A component with a clock icon.
- Decision**: A component with a decision diamond icon.
- Formal Model Converter**: A component with a document icon and a red 'X'.
- TrafficLight**: A central component with multiple output ports.
- SelfObservable**: A component with a circular icon.
- Pedestrian**: A component with a circular icon.
- Cars**: A component with a circular icon.

The right-hand pane contains the following text:

The colors of the lights above are set when the Deformable actors at the left receive. This activates the execution.

There is a corresponding WirelessDeployment model that models the same traffic light control algorithm with wireless communication between the car light and the pedestrian light. Changes to the normal operation logic of either light here will be reflected automatically in the deployment model.

This model illustrates a typical design pattern where the top level is a SR model of the physical environment for a system under design. The next level down is a modal model behavioral after the standardly mode at the light. Open the TrafficLight actor to see how it is implemented.

In order to simulate correct and erroneous behavior, we use a Decision FSMActor to switch between normal and abnormal cases.

Author: Christoph Petrich, Cheng and Shi and A. Lee

Below the diagram, there are several text blocks providing instructions and notes:

Our design should make it impossible to have the car light and pedestrian light be green at the same time (this might lead to accidents). We can use Formal Model Converter to convert the Petri nets model into a model that can be fed into a model checker to verify correctness.

Using the Formal Model Converter as the model checker NGEM:

- Follow the same instructions as for the PetriNetControl demo, except that:
 - copy the following CTL formula in the Temporal Formula field:


```
EF ( TrafficLight.CarsLightNormalState = Green & TrafficLight.PedestrianLightNormalState = PGreen )
```

 and change the NGEM version to 2.0.0. Click and open to the state to verify the fact that in the generated model (shown in this image, see), MODEL Checker has a single state and the state is assigned by the command line. We'll create and verify the assertion ASSESS from MODEL Checker (i.e., remove all text from "ASSESS" with the first "next"). Then NGEM manages to check the model with the property in fact.
- Using the RTModelCovGenerator as the model checker Reach Tool:
 - Open the model Hierarchy of TrafficLight actor (patern/covgen/trafficLight/actor) and follow the instructions there.

Graphical Modeling (cont'd) - Model Browsing

The screenshot displays two windows from a modeling tool. The top window, titled 'SR Director', shows a formal model with components like 'Clock', 'TrafficLight', and 'Pedestrian'. It includes a legend for 'Pedestrian' (red dot) and 'Cars' (yellow dot) and a note: 'The colors of the lights above are set when the SelfVariable actors at the left execute. This announces the execution.' Below this is a note: 'There is a corresponding Wireless Deployment model that models the more traffic light control algorithms with wireless communication between the car light and the pedestrian light. Changes to the normal operation logic of either light here will be reflected automatically in the deployment model.'

The bottom window, titled 'Top-level model of the traffic light controller', shows a statechart with two states: 'normal' and 'error'. Transitions are labeled with guards: 'guard: Ok_isPresent' from normal to error, and 'guard: Error_isPresent' from error to normal. On the right side of the statechart, there is a vertical list of variables: Pred, Pgrn, Cred, Cyel, Cgrn, and Cym, each with a right-pointing arrow.

Below the statechart, a note reads: 'Note that we are following the design of the Statecharts model shown on the top level, but there is a flaw in that design that shows up when constructing a deployment model. The flaw is that the Error and Ok states are at the top level, and internally contain concurrent operations of the car light and the pedestrian light. It should be other way around. The car light and pedestrian light should be concurrent, and should internally each have Error and Ok states. This way, the car light and pedestrian light can be deployed in separate hardware.'

Graphical Modeling (cont'd) - Model Browsing

The screenshot displays a Ptolemy II IDE window titled "File: /D:/NewDev.../papers/ptolemy13/Tools/Classes/NormalTrafficLightControl.pmodel". The interface is divided into several panes:

- Top-Left Pane:** A hierarchical model browser showing the project structure. The selected path is "Normal Model Compiler".
- Top-Right Pane:** A diagram of the "SR Director" actor. It contains a "TrafficLight" actor and is connected to "Pred: 1", "Cred: 1", "Cred: 0", and "Cred: 0" signals.
- Bottom-Left Pane:** A diagram of the "Formal Model Compiler" actor. It shows a state transition diagram with states "normal" and "guards".
- Bottom-Right Pane:** A detailed diagram of the "CarLightNormal" and "PedestrianLightNormal" actors. It shows inputs like "Sec", "Error", "Ok", "Cred", "Cred", "Cred", and "Cred", and outputs like "Cred", "Cred", "Cred", and "Cred".

Annotations:

- Top-Left:** "Top-level model where there are normal state. Lo implementations"
- Top-Right:** "The NormalC actor generates the control signals for the car stoplights under normal operating conditions. The NormalP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation."
- Bottom-Right:** "The CarLightNormal and PedestrianLightNormal actors here are instances of actor-oriented classes defined in other files. If you open the actors, you will open the other files. If you change the design, then all other instances of this class will see the change. In particular, the WirelessDeployment example uses the same instances."
- Bottom-Left:** "Note that we are looking at the top level, but there is a deployment model. At the top level, internally contain concurrent operations of the car light and the pedestrian light. It should be other way around. The car light and pedestrian light should be concurrent, and should internally each have Error and Ok states. This way, the car light and pedestrian light can be deployed in separate hardware."

Graphical Modeling (cont'd) - Model Browsing

The screenshot displays a Ptolemy II IDE window titled "Normal Model Controller" showing a hierarchical model browser. The browser tree on the left includes "SR Director", "Normal Model Controller", "CarLightNormal", and "PedestrianLightNormal". The main workspace shows a state transition diagram for the "normal" state, with inputs "Sec", "Error", and "OK", and outputs "Cyl", "Cred", and "Cym".

Top-level model where there are normal state. Lo implementations

Note that we are follow top level, but there is a deployment model. level, and internally contain concurrent operation pedestrian light. It should be other way around light should be concurrent, and should internal states. This way, the car light and pedestrian li separate hardware.

The NormalC actor generates the control signals for the car stoplights under normal operating conditions. The NormalP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation.

The CarLightNormal and PedestrianLightNormal actors

This model just blinks the yellow lights on the car control and turns off the pedestrian lights.

Graphical Modeling (cont'd) - Model Browsing

The screenshot shows the KIELER IDE with three windows open:

- Top-left window:** Displays a Ptolemy diagram of the SR Director actor. It shows a hierarchy of components including 'Clock', 'Decisions', and 'TrafficLight'. Annotations state: "The NormalIC actor generates the control signals for the car stoplights under normal operating conditions. The NormalIP actor reacts to these control conditions to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation."
- Bottom-left window:** Shows a state machine diagram for the SR Director. Annotations include: "Top-level where the normal s implements" and "Note that we top level, but a deployment level, and its pedestrian light should states. This is separate har".
- Bottom-right window:** Displays a state machine for the CarLightNormal actor. Annotations state: "This state machine controls the car lights. It uses the count variable to stay red for three seconds and to stay green for two." and "on the car control and turns off the pedestrian lights."

Graphical Modeling (cont'd) - Model Browsing

The NormalC actor generates the control signals for the car stoplights under normal operating conditions. The NormalP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation.

Top-level where normal s imple

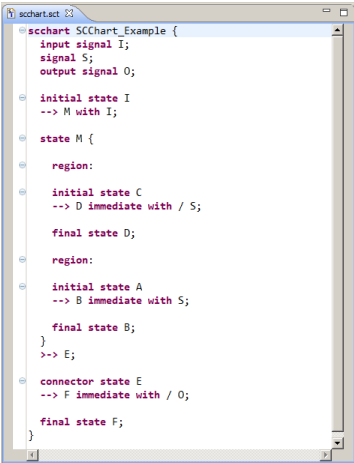
Note that we top level, but a deployment level, and ins pedestrian li light should i states. This s separate har

This state machine controls the car lights. It uses the count variable to stay red for three seconds and to stay green for two.

on the car control and turns off the pedestrian lights.

- ▶ ☹ Fokus&Context, inner and outer ports vs. performance

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

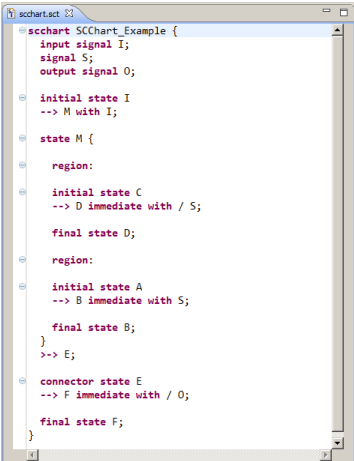
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

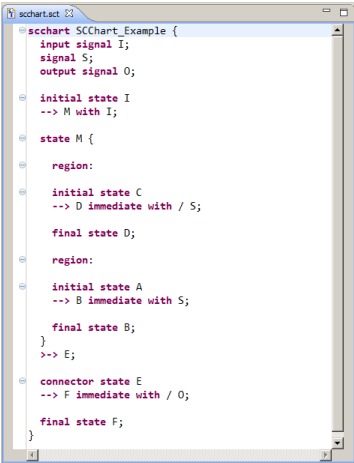
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

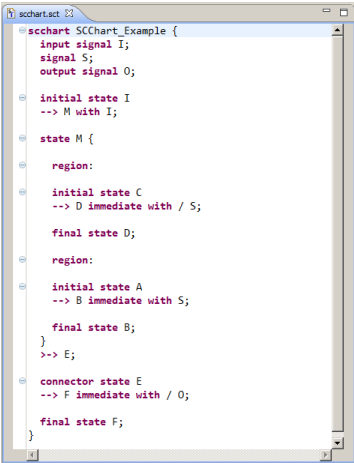
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)
 - ▶ 😊 **Handling** (model diffs, version control, tool interchange)

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

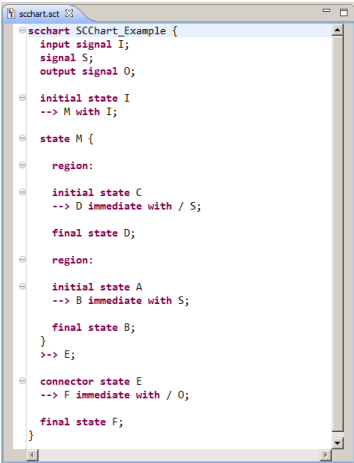
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)
 - ▶ 😊 **Handling** (model diffs, version control, tool interchange)
 - ▶ 😊 **Readability, Maintenance** (formatter)

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)
 - ▶ 😊 **Handling** (model diffs, version control, tool interchange)
 - ▶ 😊 **Readability, Maintenance** (formatter)
 - ▶ 😞 **Readability** (harder to inspect a larger model, mental map)

Graphical Modeling vs. Textual Modeling

```

scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

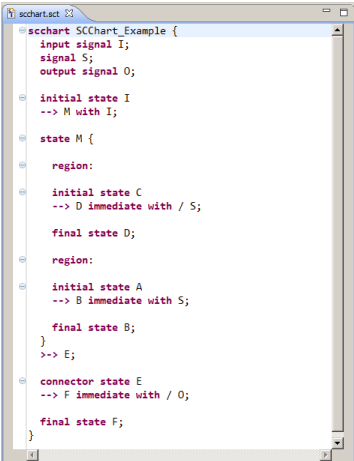
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
    
```

- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)
 - ▶ 😊 **Handling** (model diffs, version control, tool interchange)
 - ▶ 😊 **Readability, Maintenance** (formatter)
 - ▶ 😞 **Readability** (harder to inspect a larger model, mental map)
 - ▶ 😞 **Validation** (harder to see model errors)

Graphical Modeling vs. Textual Modeling



```
scchart.sct 23
scchart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:

    initial state C
    --> D immediate with / S;

    final state D;

    region:

    initial state A
    --> B immediate with S;

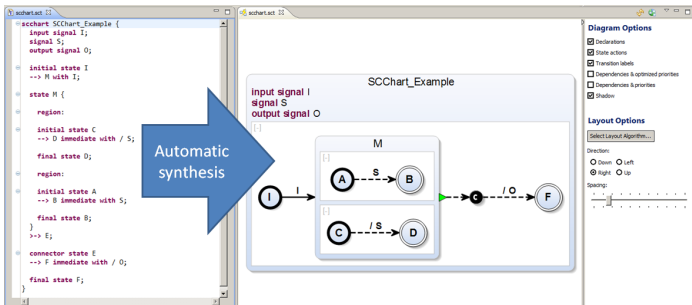
    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
```

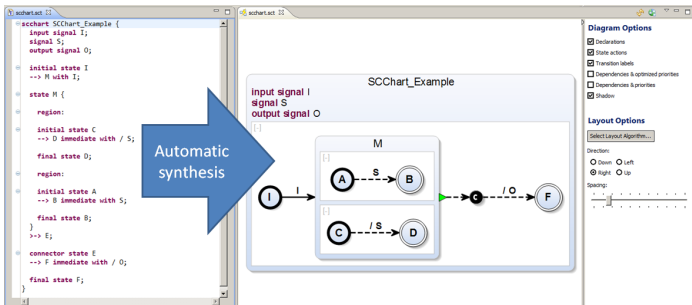
- ▶ Emerging trend: *Textual Modeling*
 - ▶ Concrete syntax is text
 - ▶ 😊 **Rapid Development** (good tooling support, e.g., Xtext)
 - ▶ 😊 **Handling** (model diffs, version control, tool interchange)
 - ▶ 😊 **Readability, Maintenance** (formatter)
 - ▶ 😞 **Readability** (harder to inspect a larger model, mental map)
 - ▶ 😞 **Validation** (harder to see model errors)
 - ▶ 😞 **Visualization** of dynamics: Simulation

Contribution



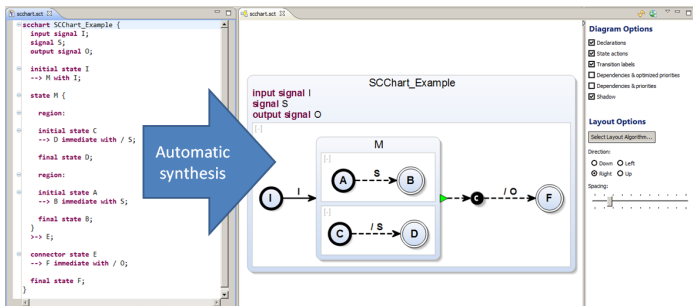
- ▶ Get all benefits from graphical modeling

Contribution



- ▶ Get all benefits from graphical modeling
- ▶ Preserve all the benefits from textual modeling

Contribution

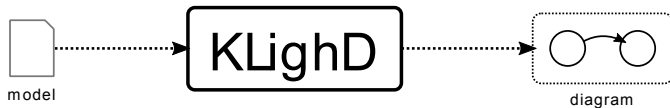


- ▶ Get all benefits from graphical modeling
- ▶ Preserve all the benefits from textual modeling
- ▶ ⇒ **Automatic synthesis of diagrams:**
KIELER Light-Weight Diagram (KLightD)

Overview

- ▶ Foundations & Concept
- ▶ Demo
- ▶ Case Study Results

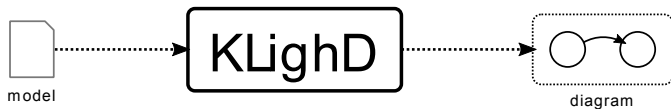
Synthesis of Diagrams



[C. Schneider et al., VL/HCC'13]

- ▶ Input: Model (dsl, xml, moml)

Synthesis of Diagrams



[C. Schneider et al., VL/HCC'13]

- ▶ Input: Model (dsl, xml, moml)
- ▶ Output: Configurable diagram

Synthesis of Diagrams



[C. Schneider et al., VL/HCC'13]

- ▶ Input: Model (dsl, xml, moml)
- ▶ Output: Configurable diagram
 - ▶ Diagram options: E.g., show transition labels

Synthesis of Diagrams



[C. Schneider et al., VL/HCC'13]

- ▶ Input: Model (dsl, xml, moml)
- ▶ Output: Configurable diagram
 - ▶ Diagram options: E.g., show transition labels
 - ▶ Layout options: E.g., direction or spacing

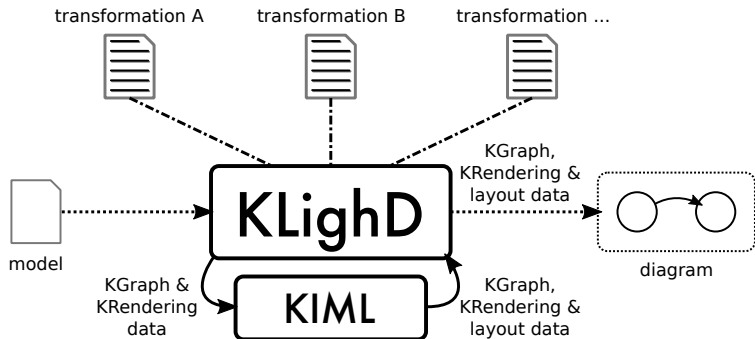
Synthesis of Diagrams



[C. Schneider et al., VL/HCC'13]

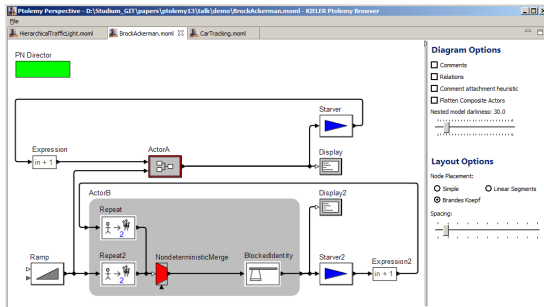
- ▶ Input: Model (dsl, xml, moml)
- ▶ Output: Configurable diagram
 - ▶ Diagram options: E.g., show transition labels
 - ▶ Layout options: E.g., direction or spacing
- ▶ Requirement: Automatic Layout (→ KIML)

KLighD Architecture



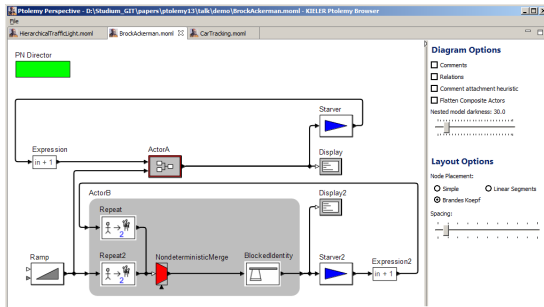
[C. Schneider et al., VL/HCC'13]

Ptolemy Case Study: Model Browsing



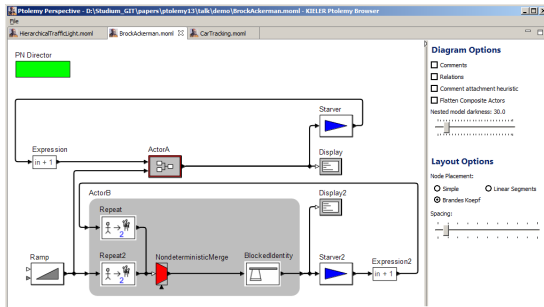
- 😊 **Readability** (normalized diagrams with fixed layout settings, configurable settings)

Ptolemy Case Study: Model Browsing



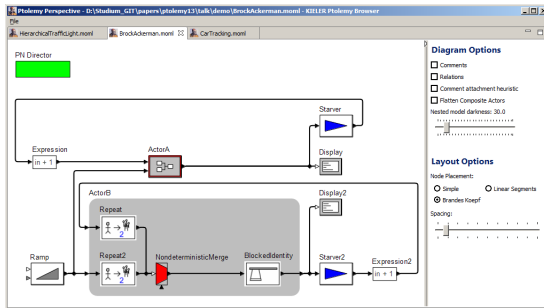
- ▶ 😊 **Readability** (normalized diagrams with fixed layout settings, configurable settings)
 - ▶ 😊 **Hierarchy** (no new windows, inner and outer ports)

Ptolemy Case Study: Model Browsing



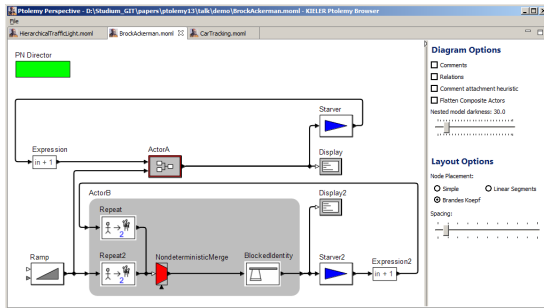
- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
 - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
 - ▶ ☺ **Large models** (Focus&Context, collapse & expand)

Ptolemy Case Study: Model Browsing



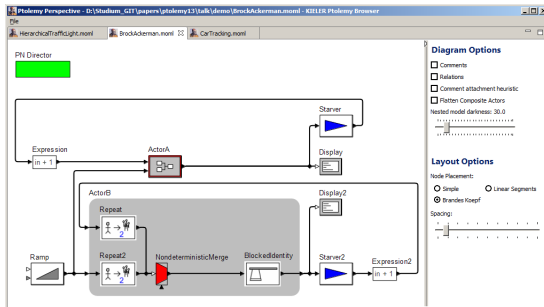
- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
 - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
 - ▶ ☺ **Large models** (Focus&Context, collapse & expand)
 - ▶ ☺ **Complex models** (filter details, e.g., transition labels)

Ptolemy Case Study: Model Browsing



- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
 - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
 - ▶ ☺ **Large models** (Focus&Context, collapse & expand)
 - ▶ ☺ **Complex models** (filter details, e.g., transition labels)
- ▶ ☺ **Maintenance / Handling** (create/edit the model in Vergil, generate it, use a textual DSL, ...)

Ptolemy Case Study: Model Browsing



- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
 - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
 - ▶ ☺ **Large models** (Focus&Context, collapse & expand)
 - ▶ ☺ **Complex models** (filter details, e.g., transition labels)
- ▶ ☺ **Maintenance / Handling** (create/edit the model in Vergil, generate it, use a textual DSL, ...)
- ▶ ☺ **Light-Weight** (no editing, no transactions → just transient views)

Model Editing & Simulation

The screenshot displays the scharact IDE interface. On the left, a code editor shows the following code for `SCChart_Example`:

```

schart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:
    initial state C
    --> D immediate with / S;

    final state D;

    region:
    initial state A
    --> B immediate with S;

    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
    
```

On the right, a diagram view shows the state machine structure. It features an initial state `I` with an outgoing transition labeled `I` to a region `M`. Inside `M`, there are two parallel regions. The top region contains states `A` and `B` with a transition labeled `S`. The bottom region contains states `C` and `D` with a transition labeled `/ S`. From state `E`, there is a transition labeled `/ O` to the final state `F`. The diagram also lists the signals: `input signal I`, `signal S`, and `output signal O`.

On the far right, there are two panels: **Diagram Options** and **Layout Options**. The **Diagram Options** panel includes checkboxes for `Declarations`, `State actions`, `Transition labels`, `Dependencies & optimized priorities`, `Dependencies & priorities`, and `Shadow`. The **Layout Options** panel includes a `Select Layout Algorithm...` button and radio buttons for `Direction` (Down, Left, Right, Up) and a `Spacing` slider.

Model Editing & Simulation

```

schart SCChart_Example {
  input signal I;
  signal S;
  output signal O;

  initial state I
  --> M with I;

  state M {
    region:
    initial state C
    --> D immediate with / S;

    final state D;

    region:
    initial state A
    --> B immediate with S;

    final state B;
  }
  >> E;

  connector state E
  --> F immediate with / O;

  final state F;
}
        
```

SCChart_Example

input signal I
 signal S
 output signal O

Diagram Options

- Declarations
- State actions
- Transition labels
- Dependencies & optimized priorities
- Dependencies & priorities
- Shadow

Layout Options

Select Layout Algorithm...

Direction:

Down Left

Right Up

Spacing:

Summary and Outlook

- ▶ Models are created once but read many times
- ▶ Large and complex, hierarchical models are hard to read and maintain

Summary and Outlook

- ▶ Models are created once but read many times
- ▶ Large and complex, hierarchical models are hard to read and maintain
- ▶ Automatic light-weight diagrams
 - ▶ help browsing/reading
 - ▶ and maintaining models

Summary and Outlook

- ▶ Models are created once but read many times
- ▶ Large and complex, hierarchical models are hard to read and maintain
- ▶ Automatic light-weight diagrams
 - ▶ help browsing/reading
 - ▶ and maintaining models
- ▶ Models can be textual or graphical
- ▶ Even general data structures (&dynamics) can be visualized

To Go Further



KLAUSKE, L. K., SCHULZE, C. D., SPÖNEMANN, M., AND VON HANXLEDEN, R.

Improved layout for data flow diagrams with port constraints.

In Proceedings of the 7th International Conference on the Theory and Application of Diagrams (DIAGRAMS'12) (2012), vol. 7352 of LNAI, Springer, pp. 65–79.



SCHNEIDER, C., SPÖNEMANN, M., AND VON HANXLEDEN, R.

Just model! – Putting automatic synthesis of node-link-diagrams into practice.

In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13) (San Jose, CA, USA, 15–19 Sept. 2013).

With accompanying poster.



SUGIYAMA, K., TAGAWA, S., AND TODA, M.

Methods for visual understanding of hierarchical system structures.

IEEE Transactions on Systems, Man and Cybernetics 11, 2 (Feb. 1981), 109–125.



UNI KIEL, REAL-TIME AND EMBEDDED SYSTEMS GROUP.

KIELER webpage.

<http://www.informatik.uni-kiel.de/en/rtsys/kieler/>.



VON HANXLEDEN, R., LEE, E. A., MOTIKA, C., AND FUHRMANN, H.

Multi-view modeling and pragmatics in 2020 — position paper on designing complex cyber-physical systems.

In Proceedings of the 17th International Monterey Workshop on Development, Operation and Management of Large-Scale Complex IT Systems, LNCS (Oxford, UK, Dec. 2012), vol. 7539.

**Thank you for your attention and
participation!**

Any questions or suggestions?

Ptolemy Case Study: Model Browsing

The screenshot displays the Ptolemy Perspective interface. The main window shows a hierarchical diagram of a traffic light system. The top-level component is 'TrafficLight', which contains an 'error' node and a sub-diagram 'SR Director'. The 'SR Director' contains an 'error' node, a 'PedestrianLightNormal' component, and a 'CarLightNormal' component. The 'PedestrianLightNormal' component has outputs for 'Pred' and 'Pgm'. The 'CarLightNormal' component has outputs for 'Cred', 'Cyel', and 'Cgm'. The 'error' node in 'SR Director' has two guards: 'Guard Ok_IsPresent' and 'Guard Error_IsPresent'. The 'Clock' and 'Decision' components are connected to the 'error' node in 'TrafficLight'. The 'Set/variable' components are connected to the outputs of the 'PedestrianLightNormal' and 'CarLightNormal' components. The interface includes a 'Diagram Options' panel on the right with checkboxes for 'Comments', 'Relations', 'Comment attachment heuristic', and 'Flatten Composite Actors', and a 'Layout Options' panel with radio buttons for 'Simple' and 'Linear Segments', and a 'Spacing' slider.

Ptolemy Perspective - D:\Studium_GIT\papers\ptolemy13\talk\demo\HierarchicalTrafficLight.moml - KIELER Ptolemy Browser

File

HierarchicalTrafficLight.moml | BrockAckerman.moml | CarTracking.moml

SR Director

- Pred: 1
- Pgm: 0
- Cred: 1
- Cyel: 1
- Cgm: 0

TrafficLight

Clock

Decision

error

Guard Ok_IsPresent

Guard Error_IsPresent

SR Director

- initialState: true

Ok

Error

PedestrianLightNormal

Pred

Pgm

CarLightNormal

Cred

Cyel

Cgm

Sec

Set/variable

Set/variable2

Set/variable3

Set/variable4

Set/variable5

Set/variable6

Diagram Options

- Comments
- Relations
- Comment attachment heuristic
- Flatten Composite Actors

Nested model darkness: 30.0

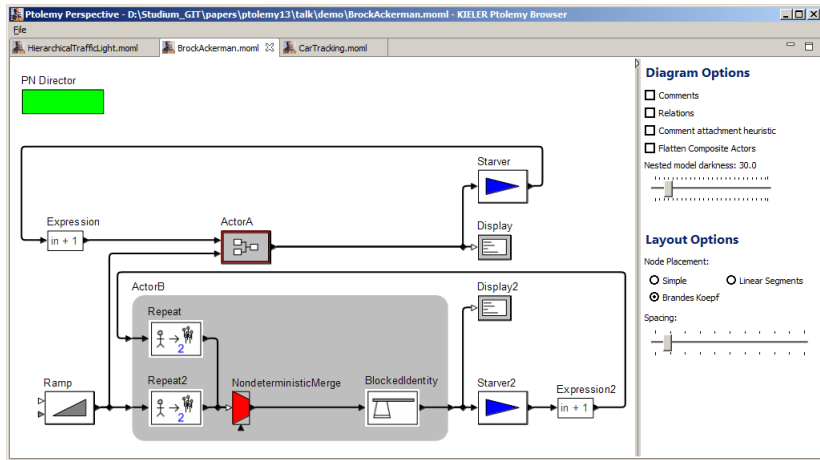
Layout Options

Node Placement:

- Simple
- Linear Segments
- Brandes Koepf

Spacing:

Ptolemy Case Study: Model Browsing (cont'd)



KLighDning - Collaborative Browser-Based Viewer

