

# 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



**KIELER**  
Kiel Integrated Environment  
for Layout Eclipse RichClient

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

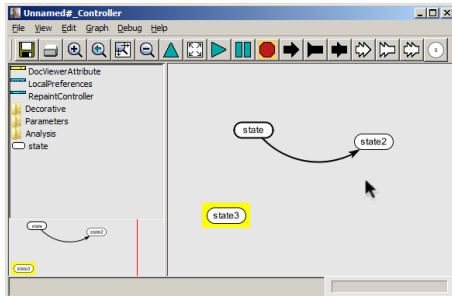
Ramp ActorB Display2

Expression2 in + 1 Starver2

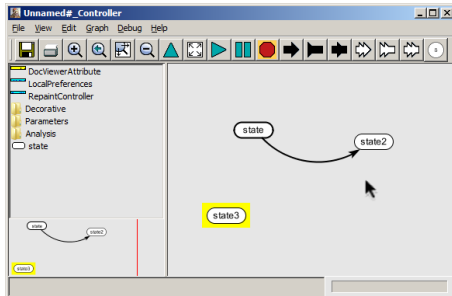
KIELER layout done in 397ms (Graph conversion 247ms, Algorithm 88ms, MoMLChanges 48ms).

# Graphical Modeling

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

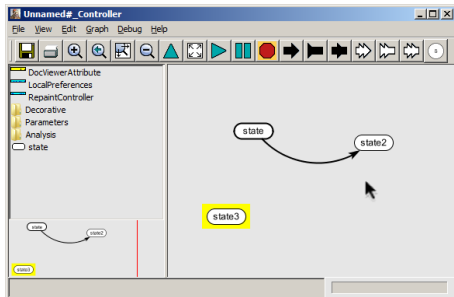


# Graphical Modeling



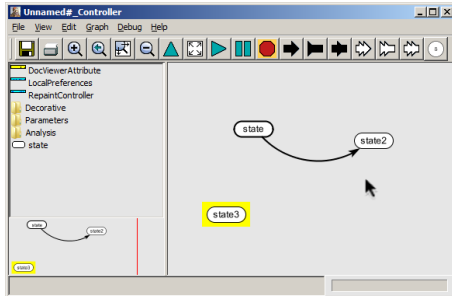
- ▶ 😊 **Short learning curve**  
 (palette, Drag&Drop)
- ▶ 😊 **Readability**  
 (inspecting/understanding,  
 mental map)

# Graphical Modeling



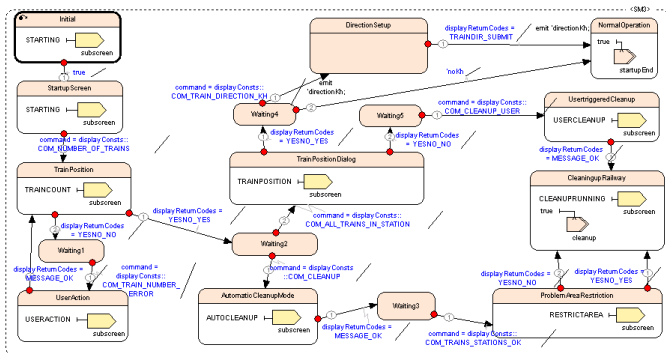
- ▶ 😊 **Short learning curve**  
(palette, Drag&Drop)
- ▶ 😊 **Readability**  
(inspecting/understanding,  
mental map)
- ▶ 😊 **Visualization of  
dynamics: Simulation**

# Graphical Modeling



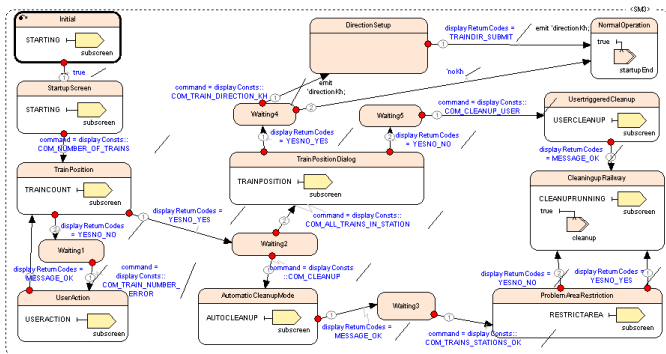
- ▶ 😊 Short learning curve (palette, Drag&Drop)
- ▶ 😊 Readability (inspecting/understanding, mental map)
- ▶ 😊 Visualization of dynamics: Simulation
- ▶ 😊 Validation (detect obvious model errors)

## Graphical Modeling (cont'd)



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

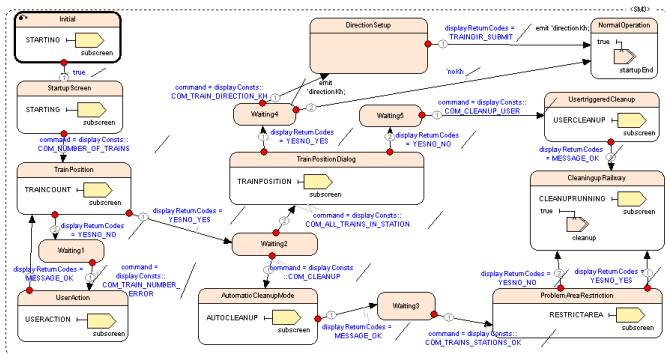
# Graphical Modeling (cont'd)



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

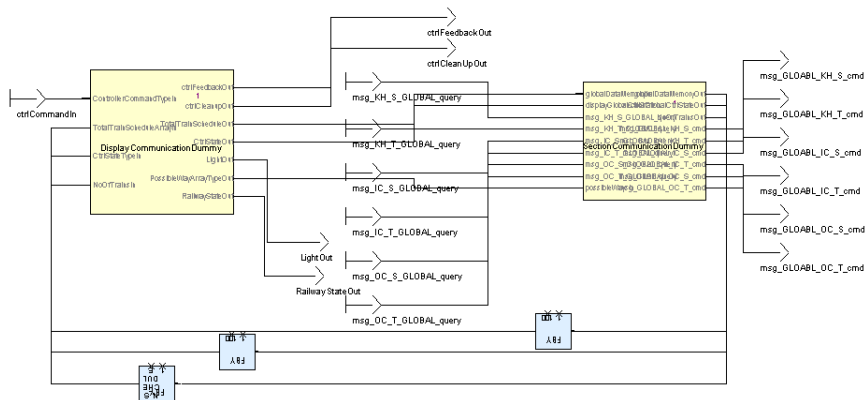


## Graphical Modeling (cont'd)



- ▶ 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

**File** View GUI Graph Debug Help

SR Director

- Pedst: 1
- Cars: 1
- Pgrm: 0
- Cylt: 1
- Cjgn: 0

**Legend:**

- Pedestrian
- Cars

The colors of the lights above are set when the Deformable actors at the left receive this activation via 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.

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 this Petri net model into a model that can be fed into a model checker to verify correctness.

Using the Formal Model Converter with the model checker NuSMV:

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

```
EF ( TrafficLightCarsLightNormalState = Cjgn & TrafficLightPedestrianLightNormalState = Pgrm )
```

 and run the NuSMV conversion (2.3) (does not seem to be able to handle the fact that in the generated model (shown in this image, see) MCDL2 (Clock) has a single state and the value is assigned by the control flow, which is not modeled and hence the action ASIGNER from MCDL2 (Clock); i.e., remove all text from "ASIGNER" with the first "next"; Then NuSMV manages to check the model and finds the property to hold.
- Using the RTModelCodeGenerator in the model checker Real Time Monitor. Open the model hierarchy at TrafficLight actor (editor/codegen/realtime/actor/real) and follow the instructions there.

Authors: Christoph Petric, Cheng and Shu and A. Lee

# 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 about 'SelfVariable' actors. The bottom window, titled 'Top-level model of the traffic light controller', shows a statechart with 'normal' and 'error' states. It includes a legend for 'Pred', 'Pgrn', 'Cred', 'Cyel', and 'Cgrn' and a note about the design flaw.

**Top-level model of the traffic light controller**  
where there are two states, an error state and a normal state. Look inside the states to see the implementations.

```

    graph LR
      normal((normal)) -- "guard: Ok_isPresent" --> error((error))
      error -- "guard: Error_isPresent" --> normal
  
```

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 SR Director generates the control signals for the car stoplights under normal operating conditions. The NormalIP 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 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.

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

Note that we are looking at top level, but there is a deployment model. At this 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 multi-windowed Ptolemy II IDE interface. The top-left window shows a high-level block diagram of the 'SR Director' and 'TrafficLight' components. The top-right window shows a similar diagram for the 'SR Director' with explanatory text: "The NormalIC actor generates the control signals for the car stoplights under normal operating conditions. The NormalIP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation." The middle window shows the 'Normal Model Converter' with a state diagram and text: "Top-level model where there are normal state. Lo implementations". The bottom window shows a detailed state machine for the 'CarLightNormal' actor, with text: "This model just blinks the yellow lights on the car control and turns off the pedestrian lights." The state machine includes states like 'YellowOn' and transitions based on inputs like 'Sec', 'Error', and 'Cred'.

Note that we are looking top level, but there is a deployment model level, and internally contain concurrent operations. Light should be concurrent and should internal states. This way, the car light and pedestrian light separate hardware.

# Graphical Modeling (cont'd) - Model Browsing

The SR Director 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.

Top-level where the normal s implements

Note that we top level, but a deployment level, and its pedestrian light should states. This 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.

# 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 implements

Note that we go top level, but a deployment level, and its pedestrian light should implement. This is separate har

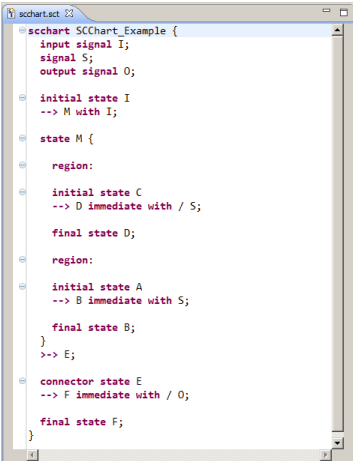
This state machine controls the car lights. It uses the count variable to stay red for three seconds and to stay green for two.

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;

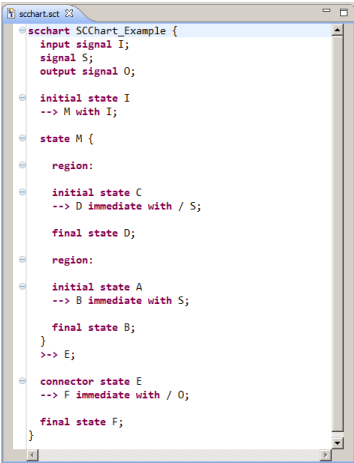
    final state B;
  }
  >> E;

  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;

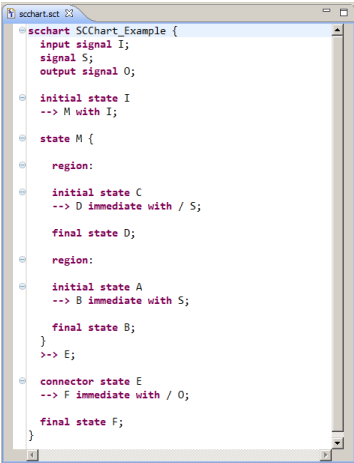
    final state B;
  }
  >> E;

  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;

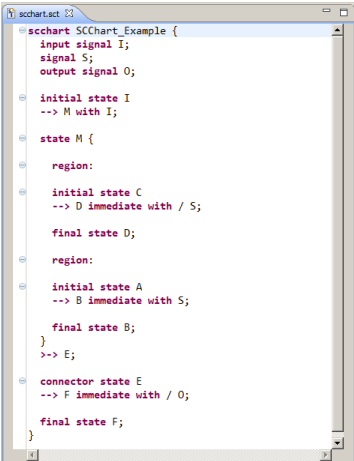
    final state B;
  }
  >> E;

  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;

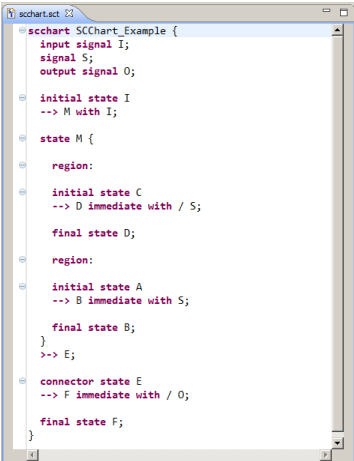
    final state B;
  }
  >> E;

  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;

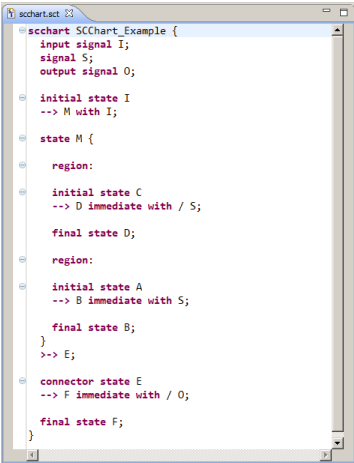
    final state B;
  }
  >> E;

  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.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;

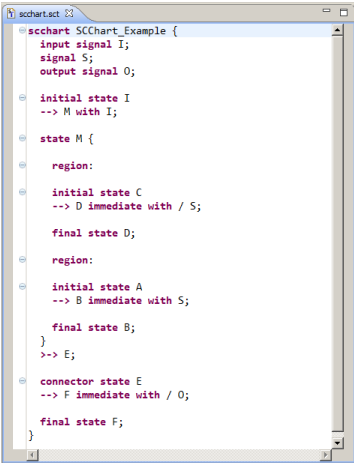
    final state B;
  }
  >> E;

  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)

# 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;

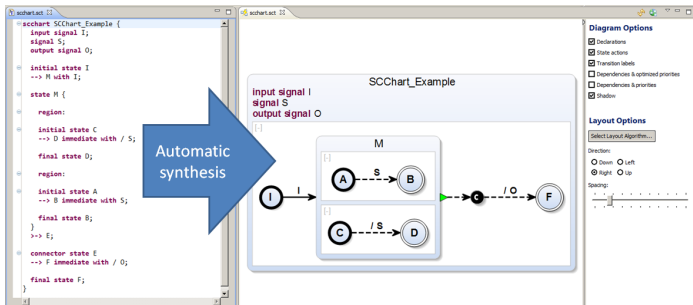
    final state B;
  }
  >> E;

  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

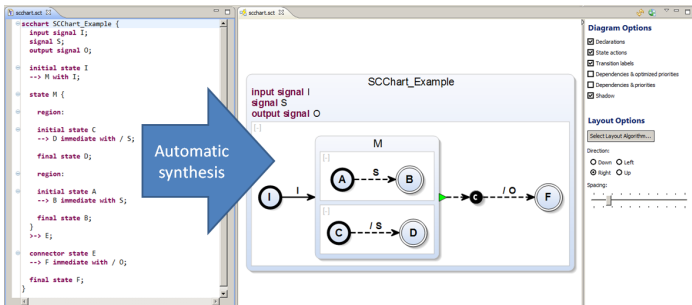
# Contribution



- ▶ Get all benefits from graphical modeling

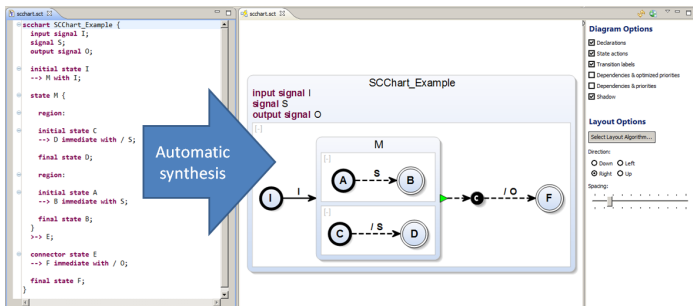


# Contribution



- ▶ 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

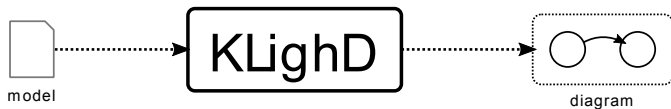
# Synthesis of Diagrams



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

- ▶ Input: Model (dsl, xml, moml)

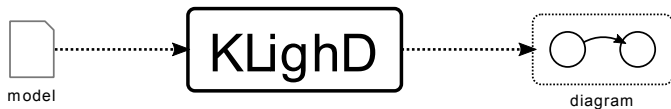
## Synthesis of Diagrams



[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

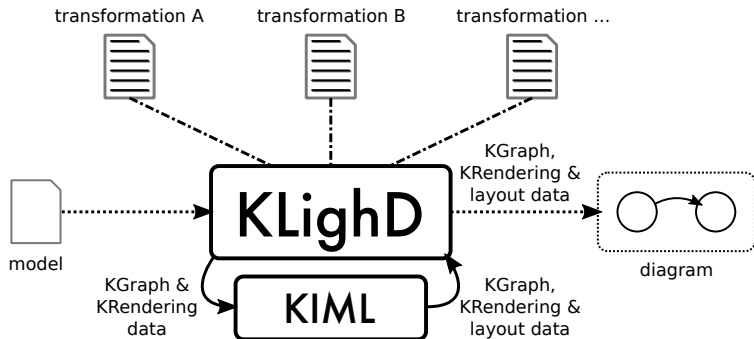


[C. Schneider et al., VL/HCC'13]

- ▶ 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)

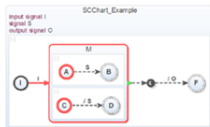


# KLighD Architecture



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

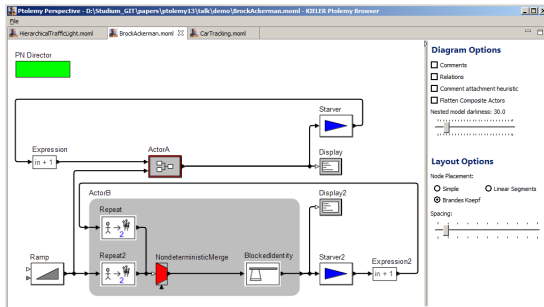
# KLighD Demo



## DEMO

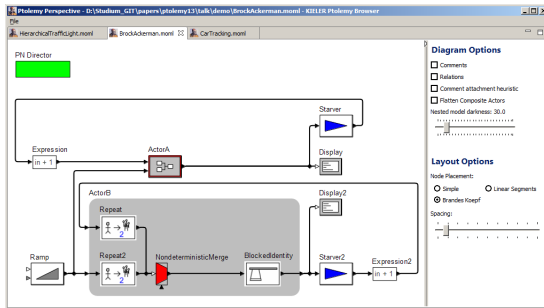


# Ptolemy Case Study: Model Browsing



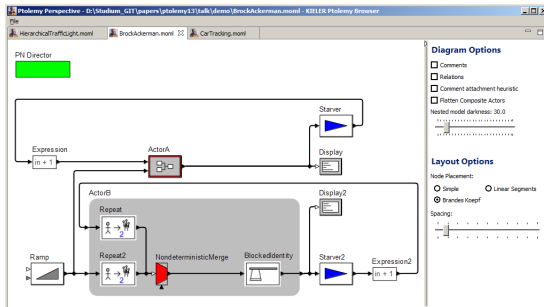
- 😊 **Readability** (normalized diagrams with fixed layout settings, configurable settings)

# Ptolemy Case Study: Model Browsing



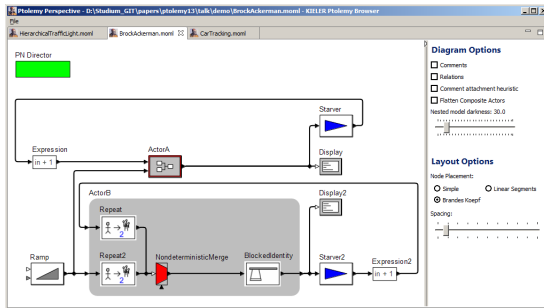
- ▶ 😊 **Readability** (normalized diagrams with fixed layout settings, configurable settings)
  - ▶ 😊 **Hierarchy** (no new windows, inner and outer ports)

# Ptolemy Case Study: Model Browsing



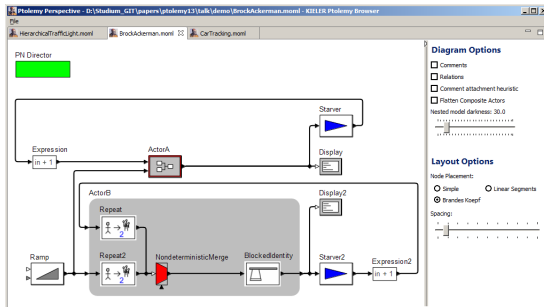
- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
  - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
  - ▶ ☺ **Large models** (Focus&Context, collapse & expand)

# Ptolemy Case Study: Model Browsing



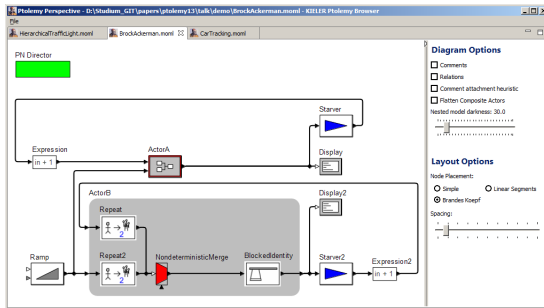
- ▶ ☺ **Readability** (normalized diagrams with fixed layout settings, configurable settings)
  - ▶ ☺ **Hierarchy** (no new windows, inner and outer ports)
  - ▶ ☺ **Large models** (Focus&Context, collapse & expand)
  - ▶ ☺ **Complex models** (filter details, e.g., transition labels)

# 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 definition of the SCChart\_Example component. On the right, a diagram view shows the visual representation of this component, including its ports and internal state machine structure. A right-hand sidebar contains configuration options for the diagram.

```

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;
}
    
```

The diagram view shows the SCChart\_Example component with the following structure:

- Ports:** Input signal I, signal S, and output signal O.
- Initial State:** I.
- State M:** A region containing states A, B, C, and D.
  - State A transitions to state B on signal S.
  - State C transitions to state D on signal S.
- Connector State E:** A state that transitions to state F on signal O.
- Final State:** F.

The right sidebar contains the following options:

- Diagram Options:**
  - Declarations
  - State actions
  - Transition labels
  - Dependencies & optimized priorities
  - Dependencies & priorities
  - Shadow
- Layout Options:**
  - Select Layout Algorithm...
- Direction:**
  - Down
  - Left
  - Right
  - Up
- Spacing:** A slider control.

# 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

## 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!**

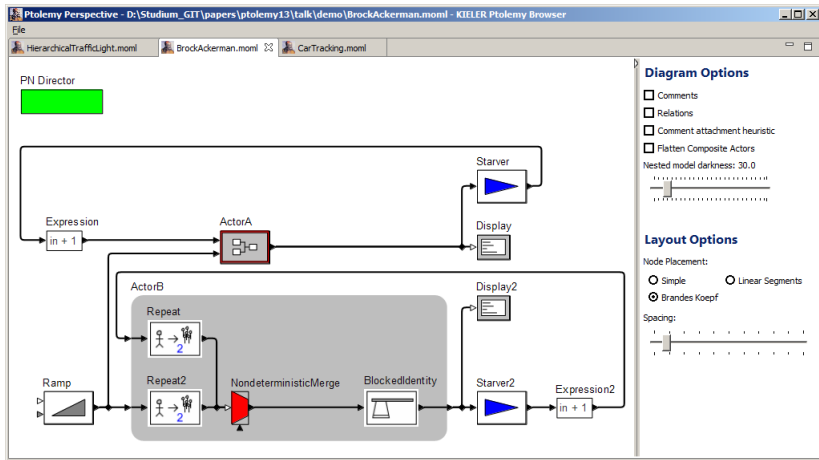
**Any questions or suggestions?**

# Ptolemy Case Study: Model Browsing

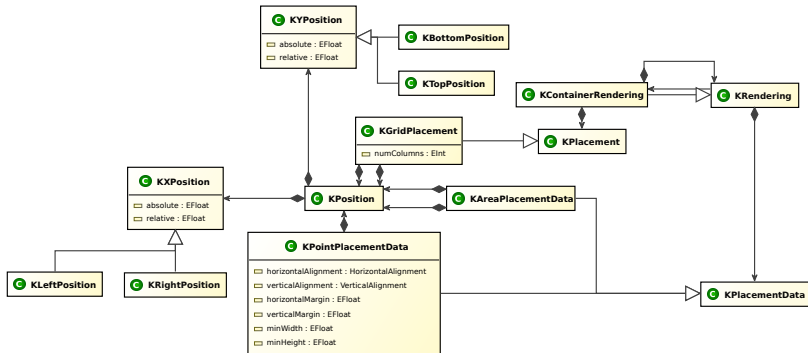
The screenshot displays the Ptolemy Perspective interface. The main window shows a hierarchical diagram of a traffic light system. The top-level component is 'TrafficLight', which contains an 'error' node and a sub-diagram 'SR Director'. The 'SR Director' contains an 'error' node, a 'PedestrianLightNormal' 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. The status bar at the bottom shows 'Christian Motika' and 'Light-Weight Synthesis of Ptolemy Diagrams with KIELER'.



# Ptolemy Case Study: Model Browsing (cont'd)



# General Data Structure Visualization



# KLighDning - Collaborative Browser-Based Viewer

