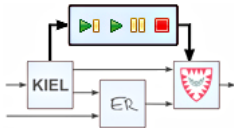


Model Execution and Meta Layout in Eclipse

Snapshots of the KIELER Project

Christian Motika and Miro Spönemann

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



Eclipse Summit, 08/27/2009

Part I: Model Execution in Eclipse

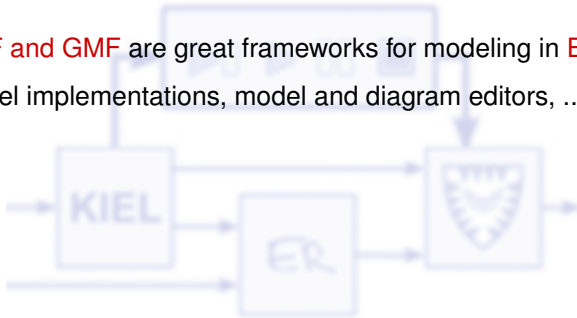
Motivation

- ▶ **EMF** and **GMF** are great frameworks for modeling in **Eclipse**.



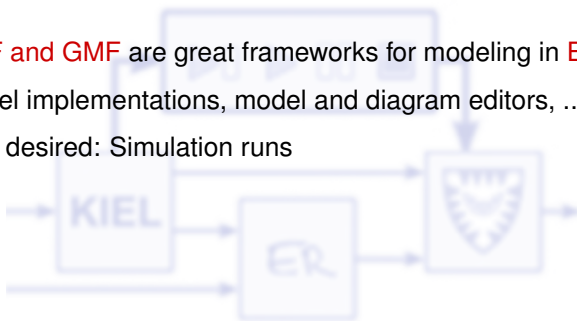
Motivation

- ▶ **EMF** and **GMF** are great frameworks for modeling in **Eclipse**.
- ▶ Model implementations, model and diagram editors, ...



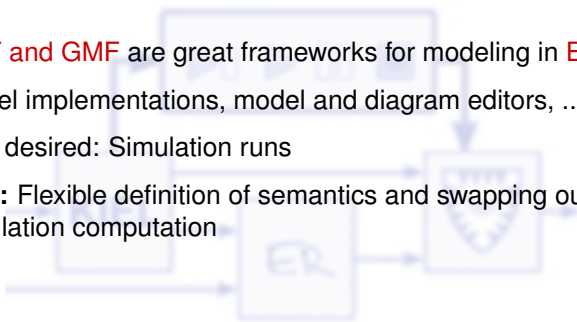
Motivation

- ▶ **EMF and GMF** are great frameworks for modeling in **Eclipse**.
- ▶ Model implementations, model and diagram editors, ...
- ▶ Also desired: Simulation runs



Motivation

- ▶ **EMF and GMF** are great frameworks for modeling in **Eclipse**.
- ▶ Model implementations, model and diagram editors, ...
- ▶ Also desired: Simulation runs
- ▶ **Idea:** Flexible definition of semantics and swapping out of simulation computation



Motivation

- ▶ **EMF and GMF** are great frameworks for modeling in **Eclipse**.
- ▶ Model implementations, model and diagram editors, ...
- ▶ Also desired: Simulation runs
- ▶ **Idea:** Flexible definition of semantics and swapping out of simulation computation
- ▶ **Solution proposed:** **KIELER Execution Manager**

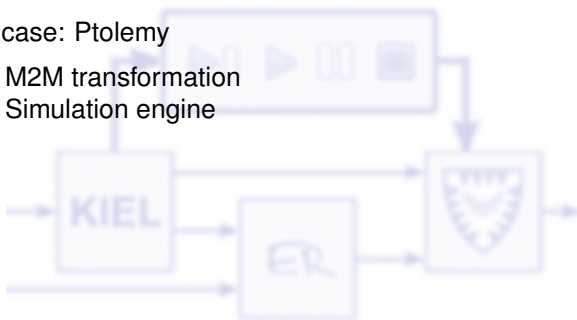
Overview

► KIELER Execution Manager



Overview

- ▶ KIELER Execution Manager
- ▶ Use case: Ptolemy
 - ▶ M2M transformation
 - ▶ Simulation engine



Overview

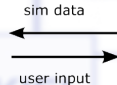
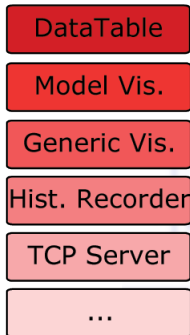
- ▶ KIELER Execution Manager
- ▶ Use case: Ptolemy
 - ▶ M2M transformation
 - ▶ Simulation engine
- ▶ Use case: Model Railway
 - ▶ Installation
 - ▶ Railway Controller DSL
 - ▶ SimpleRailCtrl editor (*DEMO*)

Overview

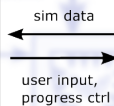
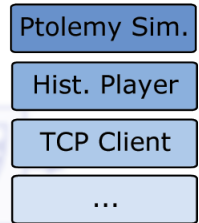
- ▶ KIELER Execution Manager
- ▶ Use case: Ptolemy
 - ▶ M2M transformation
 - ▶ Simulation engine
- ▶ Use case: Model Railway
 - ▶ Installation
 - ▶ Railway Controller DSL
 - ▶ SimpleRailCtrl editor (*DEMO*)
- ▶ Summary

KIEM Components

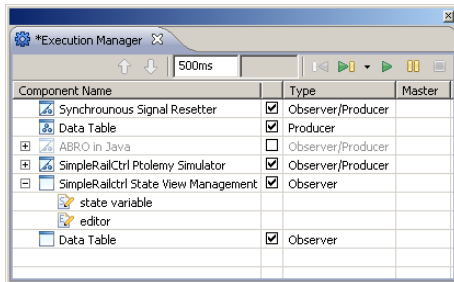
Data Observer



Data Producer



KIEM GUI and Threads

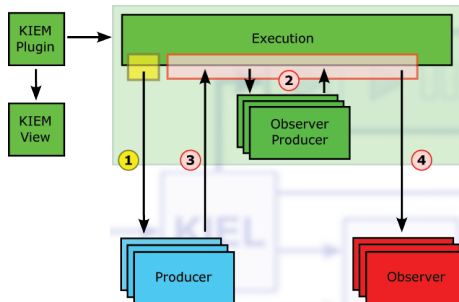


The screenshot shows the KIEM Execution Manager window. It has a title bar with a gear icon and the text '*Execution Manager'. Below the title bar is a toolbar with up/down arrows, a time field set to '500ms', and several status icons. The main area is a table with three columns: 'Component Name', 'Type', and 'Master'.

Component Name	Type	Master
Synchronous Signal Resetter	<input checked="" type="checkbox"/> Observer/Producer	
Data Table	<input checked="" type="checkbox"/> Producer	
ABRO in Java	<input type="checkbox"/> Observer/Producer	
SimpleRailCtrl Ptolemy Simulator	<input checked="" type="checkbox"/> Observer/Producer	
SimpleRailCtrl State View Management	<input checked="" type="checkbox"/> Observer	
state variable		
editor		
Data Table	<input checked="" type="checkbox"/> Observer	

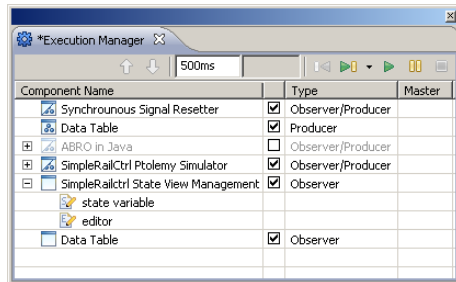
► DataObserver vs.
DataProducer

KIEM GUI and Threads



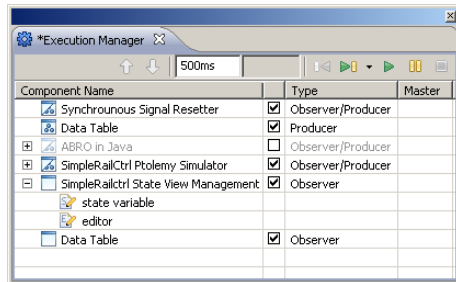
- DataObserver vs. DataProducer
- Scheduling & DataPool

KIEM GUI and Threads



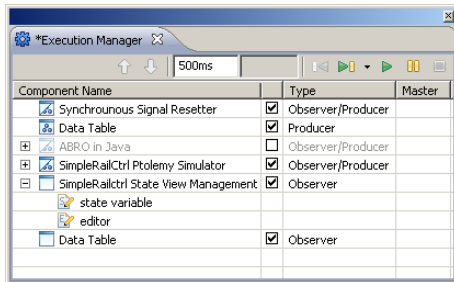
- DataObserver vs. DataProducer
- Scheduling & DataPool
- Properties

KIEM GUI and Threads



- ▶ DataObserver vs. DataProducer
- ▶ Scheduling & DataPool
- ▶ Properties
- ▶ Execution Buttons

KIEM GUI and Threads



- ▶ DataObserver vs. DataProducer
- ▶ Scheduling & DataPool
- ▶ Properties
- ▶ Execution Buttons
- ▶ Master

Simple Interface

```
1 public interface IDataComponent {  
2  
3     public void initialize() throws KiemInitializationException;  
4  
5     public void wrapup() throws KiemInitializationException;  
6  
7     public boolean isProducer();  
8  
9     public boolean isObserver();  
10  
11     public JSONObject step(JSONObject jsonObject)  
12                             throws KiemExecutionException;  
13  
14 }
```

Flexible Extensions

```
1 public abstract class DataComponent implements IDataComponent,  
2                                     IExecutableExtension {  
3  
4     public boolean isMultiInstantiable() {return false;}  
5  
6     public String[] provideFilterKeys() {return null;}  
7  
8     public KiemProperty[] provideProperties() {return null;}  
9  
10    public void checkProperties(KiemProperty[] properties)  
11                               throws KiemPropertyException {}  
12  
13    public String[] provideInterfaceKeys() {return null;}  
14  
15    public boolean isHistoryObserver() {return false;}  
16  
17    public boolean isDeltaObserver() {return false;}  
18  
19    public boolean isMaster() {return false;}  
20  
21 }
```

Overview

- ▶ KIELER Execution Manager
- ▶ Use case: Ptolemy
 - ▶ M2M transformation
 - ▶ Simulation engine
- ▶ Use case: Model Railway
 - ▶ Installation
 - ▶ Railway Controller DSL
 - ▶ SimpleRailCtrl editor (*DEMO*)
- ▶ Summary

What is Ptolemy?



- ▶ „The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems.“

Introduction to Ptolemy II, UC Berkeley

What is Ptolemy?



- ▶ „The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems.“

Introduction to Ptolemy II, UC Berkeley

- ▶ Executable models to describe behavior of reactive systems

What is Ptolemy?

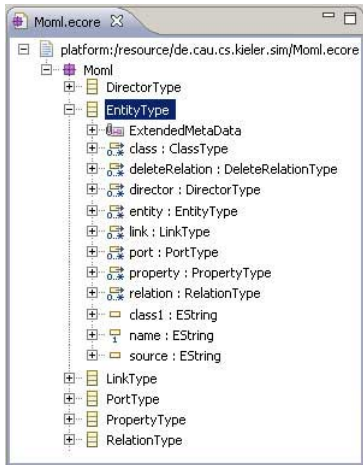


- ▶ „The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems.“

Introduction to Ptolemy II, UC Berkeley

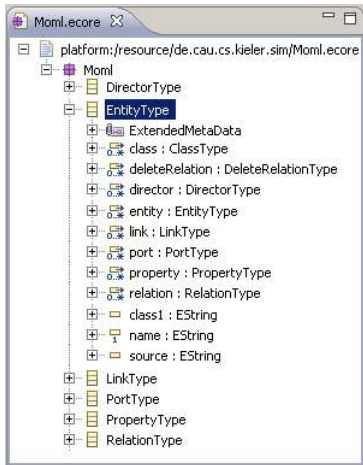
- ▶ Executable models to describe behavior of reactive systems
- ▶ Set of components interacting under a *model of computation*

Ptolemy EMF Model



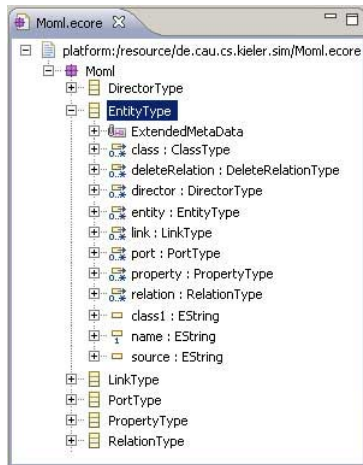
► Ptolemy models can be executable

Ptolemy EMF Model



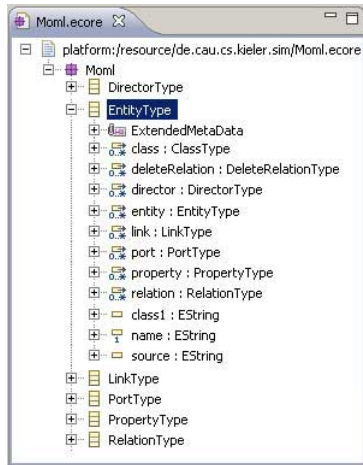
- ▶ Ptolemy models can be executable
- ▶ DTD of the Ptolemy XML representation (*MOML*)

Ptolemy EMF Model



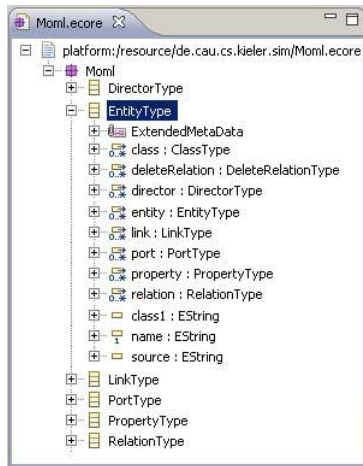
- ▶ Ptolemy models can be executable
- ▶ DTD of the Ptolemy XML representation (*MOML*)
 - ▶ Acquire EMF model

Ptolemy EMF Model



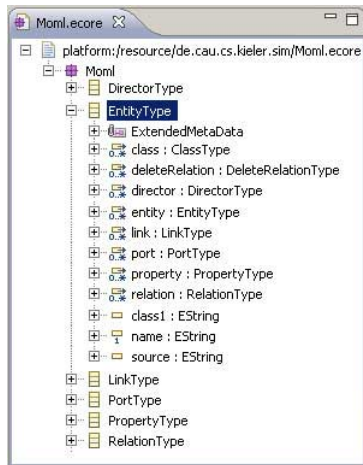
- ▶ Ptolemy models can be executable
- ▶ DTD of the Ptolemy XML representation (*MOML*)
 - ▶ Acquire EMF model
 - ▶ M2M transformation

Ptolemy EMF Model



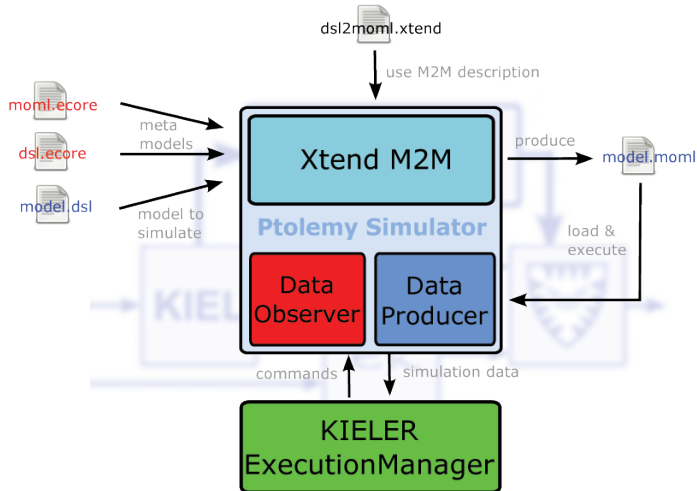
- ▶ Ptolemy models can be executable
- ▶ DTD of the Ptolemy XML representation (*MOML*)
 - ▶ Acquire EMF model
 - ▶ M2M transformation
 - ▶ Execute Ptolemy models

Ptolemy EMF Model

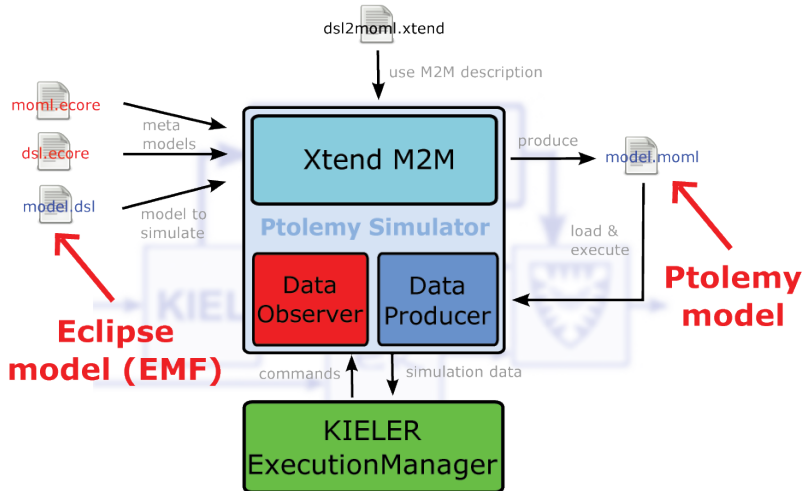


- ▶ Ptolemy models can be executable
- ▶ DTD of the Ptolemy XML representation (*MOML*)
 - ▶ Acquire EMF model
 - ▶ M2M transformation
 - ▶ Execute Ptolemy models
 - ▶ Back mapping of data/states

Simulation with Ptolemy



Simulation with Ptolemy



Overview

- ▶ KIELER Execution Manager
- ▶ Use case: Ptolemy
 - ▶ M2M transformation
 - ▶ Simulation engine
- ▶ Use case: Model Railway
 - ▶ Installation
 - ▶ Railway Controller DSL
 - ▶ SimpleRailCtrl editor (*DEMO*)
- ▶ Summary

Installation



- Standard model railway equipment

Installation



- Standard model railway equipment combined with

Installation



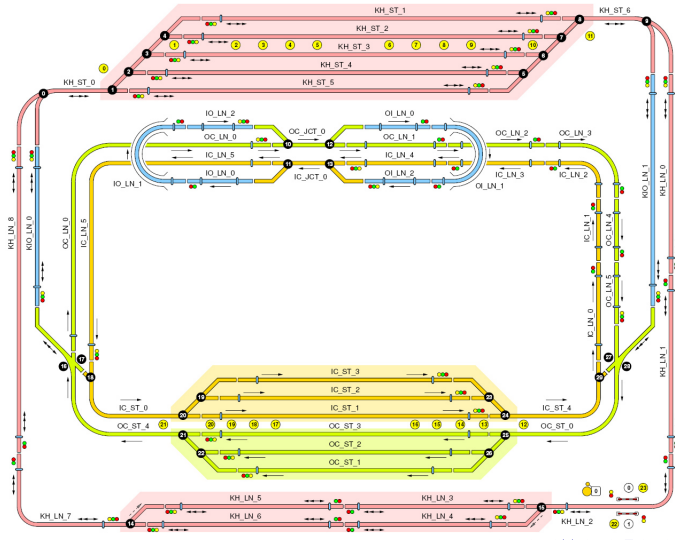
- ▶ Standard model railway equipment combined with
- ▶ Over 200 sensors and actuators

Installation

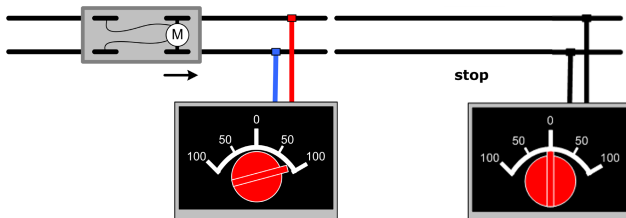


- ▶ Standard model railway equipment combined with
- ▶ Over 200 sensors and actuators
- ▶ Controlled by distributed computer system

Track Layout

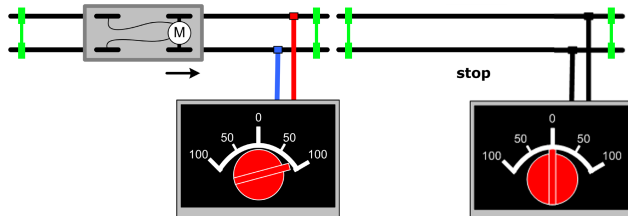


Train Movement



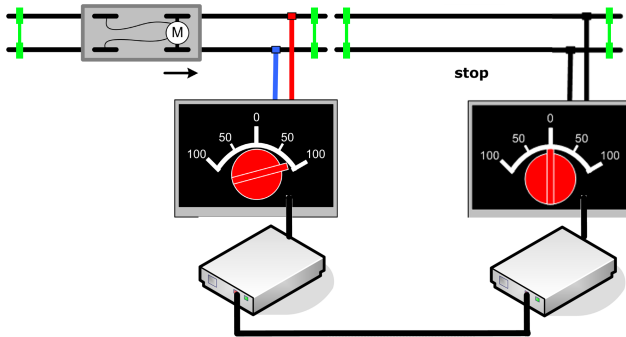
- Several track segments individually controlled

Train Movement



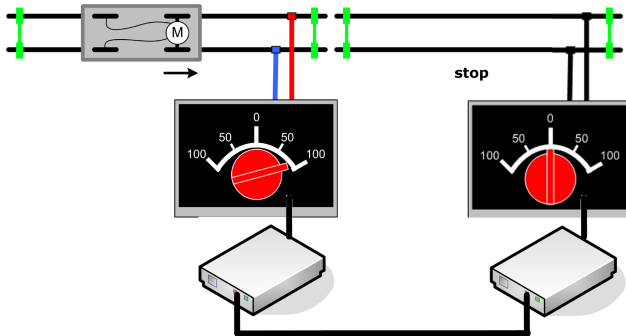
- ▶ Several track segments individually controlled
- ▶ Sensors allow detection of instantaneous train positions





Train Movement (cont'd)



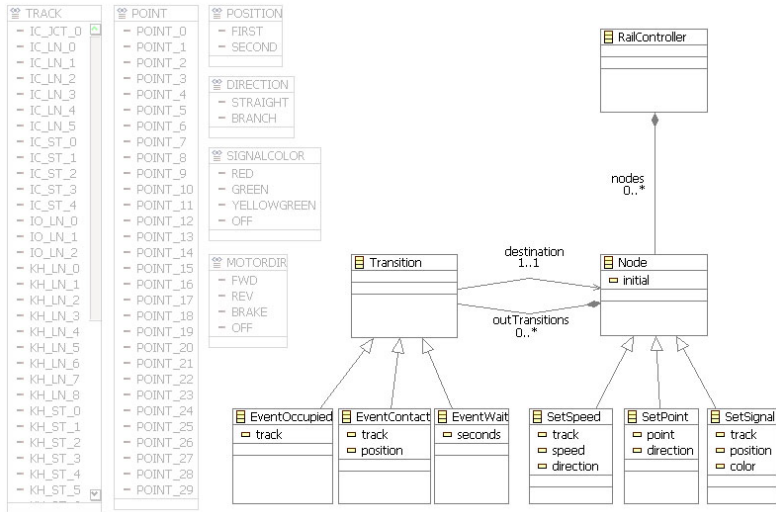
- Computers get sensor information and control voltage

Train Movement (cont'd)

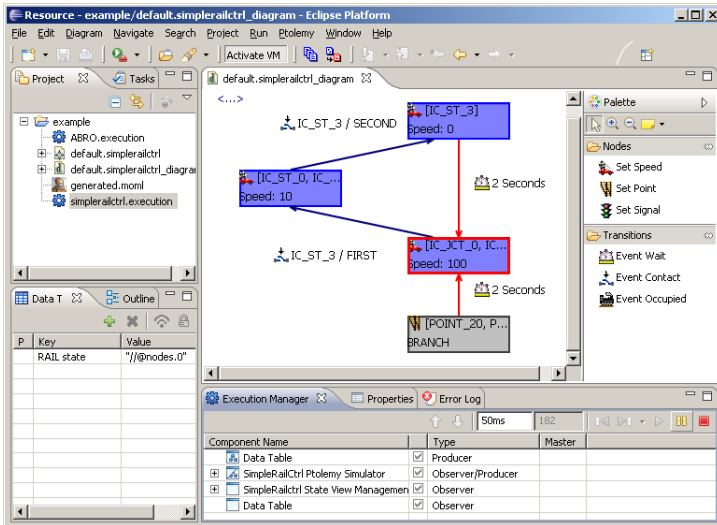


- ▶ Computers get sensor information and control voltage
 - ▶ ⇒ Actions:  SetSpeed,  SetPoint
 - ▶ ⇒ Trigger:  EventContact,  EventTimeout

EMF Meta Model



Generated GMF Editor



Eclipse Model Execution Demo



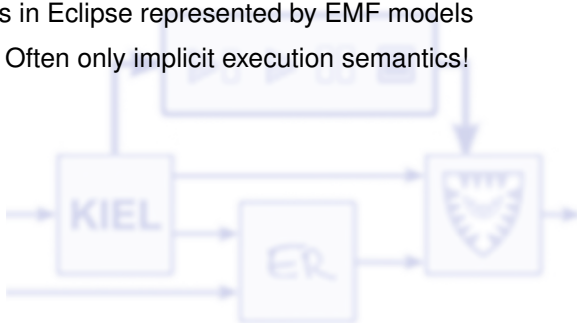
Summary

- ▶ DSLs in Eclipse represented by EMF models



Summary

- ▶ DSLs in Eclipse represented by EMF models
 - ▶ Often only implicit execution semantics!



Summary

- ▶ DSLs in Eclipse represented by EMF models
 - ▶ Often only implicit execution semantics!
- ▶ Ptolemy models can be executable



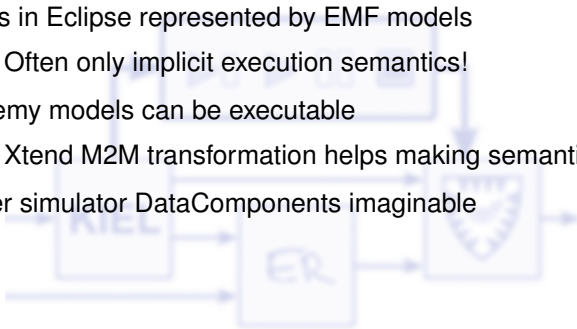
Summary

- ▶ DSLs in Eclipse represented by EMF models
 - ▶ Often only implicit execution semantics!
- ▶ Ptolemy models can be executable
 - ▶ Xtend M2M transformation helps making semantics explicit



Summary

- ▶ DSLs in Eclipse represented by EMF models
 - ▶ Often only implicit execution semantics!
- ▶ Ptolemy models can be executable
 - ▶ Xtend M2M transformation helps making semantics explicit
- ▶ Other simulator DataComponents imaginable



Summary

- ▶ DSLs in Eclipse represented by EMF models
 - ▶ Often only implicit execution semantics!
- ▶ Ptolemy models can be executable
 - ▶ Xtend M2M transformation helps making semantics explicit
- ▶ Other simulator DataComponents imaginable
- ▶ **KIELER Execution Manager** seamlessly integrates execution into the Eclipse RCP

To Go Further



UC BERKELEY, EECS DEPT.

Ptolemy webpage.

[http://ptolemy.eecs.berkeley.edu/.](http://ptolemy.eecs.berkeley.edu/)

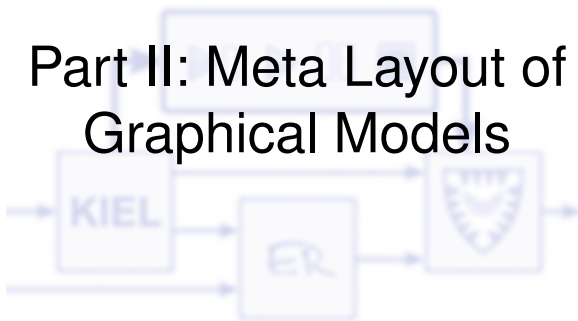


UNI KIEL, REAL-TIME AND EMBEDDED SYSTEMS GROUP.

KIELER webpage.

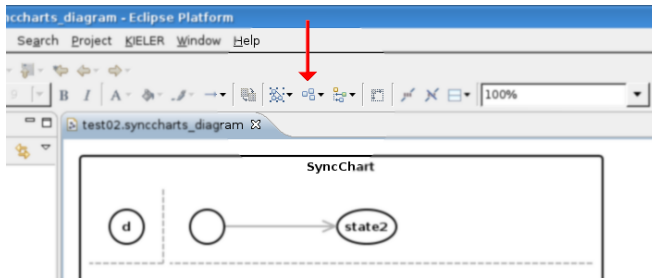
[http://www.informatik.uni-kiel.de/en/rtsys/kieler/.](http://www.informatik.uni-kiel.de/en/rtsys/kieler/)

Part II: Meta Layout of Graphical Models



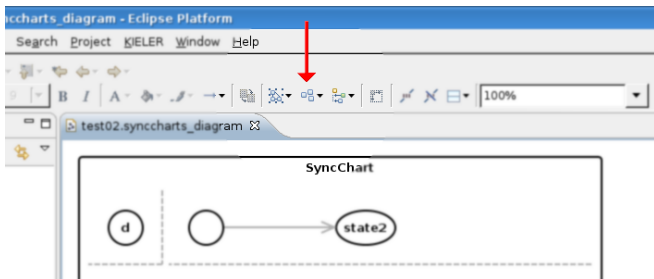
Automatic Layout in GMF

- GMF supports automatic layout...



Automatic Layout in GMF

- ▶ GMF supports automatic layout...



- ▶ ...but is not very flexible
 - ▶ No selection of different layout algorithms
 - ▶ No customization of layout options
 - ▶ No deep layout of compound structures

KIELER Meta Layout

- **Meta** Layout: Allow fully flexible automatic diagram layout



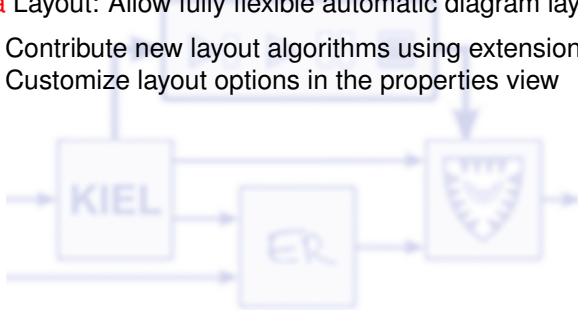
KIELER Meta Layout

- ▶ **Meta** Layout: Allow fully flexible automatic diagram layout
 - ▶ Contribute new layout algorithms using extension points



KIELER Meta Layout

- ▶ **Meta** Layout: Allow fully flexible automatic diagram layout
 - ▶ Contribute new layout algorithms using extension points
 - ▶ Customize layout options in the properties view



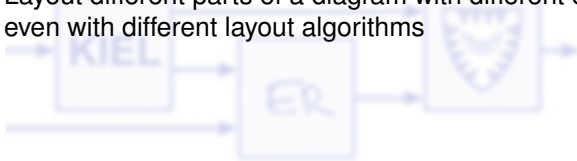
KIELER Meta Layout

- ▶ **Meta** Layout: Allow fully flexible automatic diagram layout
 - ▶ Contribute new layout algorithms using extension points
 - ▶ Customize layout options in the properties view
 - ▶ Layout compound structures recursively



KIELER Meta Layout

- ▶ **Meta** Layout: Allow fully flexible automatic diagram layout
 - ▶ Contribute new layout algorithms using extension points
 - ▶ Customize layout options in the properties view
 - ▶ Layout compound structures recursively
 - ▶ Layout different parts of a diagram with different options, or even with different layout algorithms



KIELER Meta Layout

- ▶ **Meta** Layout: Allow fully flexible automatic diagram layout
 - ▶ Contribute new layout algorithms using extension points
 - ▶ Customize layout options in the properties view
 - ▶ Layout compound structures recursively
 - ▶ Layout different parts of a diagram with different options, or even with different layout algorithms
- ▶ Development of special layout algorithms, e.g. for **data flow diagrams**

Layout Providers

Extension points are used to

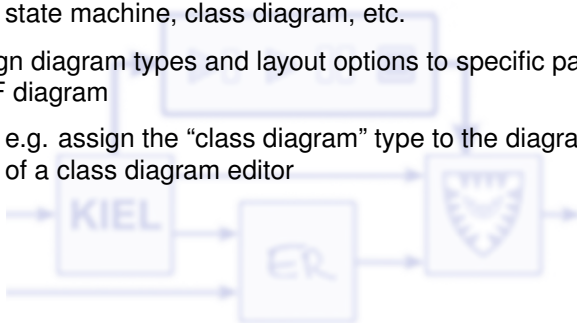
- ▶ define diagram types
 - ▶ state machine, class diagram, etc.



Layout Providers

Extension points are used to

- ▶ define diagram types
 - ▶ state machine, class diagram, etc.
- ▶ assign diagram types and layout options to specific parts of a GMF diagram
 - ▶ e.g. assign the “class diagram” type to the diagram edit part of a class diagram editor



Layout Providers

Extension points are used to

- ▶ define diagram types
 - ▶ state machine, class diagram, etc.
- ▶ assign diagram types and layout options to specific parts of a GMF diagram
 - ▶ e.g. assign the “class diagram” type to the diagram edit part of a class diagram editor
- ▶ contribute new layout algorithms
 - ▶ call these contributions **layout providers**

Layout Providers

Extension points are used to

- ▶ define diagram types
 - ▶ state machine, class diagram, etc.
- ▶ assign diagram types and layout options to specific parts of a GMF diagram
 - ▶ e.g. assign the “class diagram” type to the diagram edit part of a class diagram editor
- ▶ contribute new layout algorithms
 - ▶ call these contributions **layout providers**
- ▶ define which diagram types are supported by the layout provider

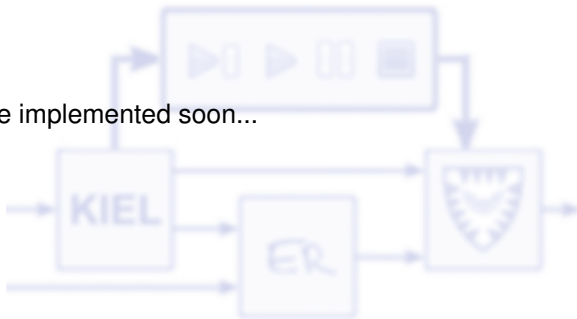
Layout Providers

Extension points are used to

- ▶ define diagram types
 - ▶ state machine, class diagram, etc.
- ▶ assign diagram types and layout options to specific parts of a GMF diagram
 - ▶ e.g. assign the “class diagram” type to the diagram edit part of a class diagram editor
- ▶ contribute new layout algorithms
 - ▶ call these contributions **layout providers**
- ▶ define which diagram types are supported by the layout provider
- ▶ define new layout options and specify which options are understood by a layout provider

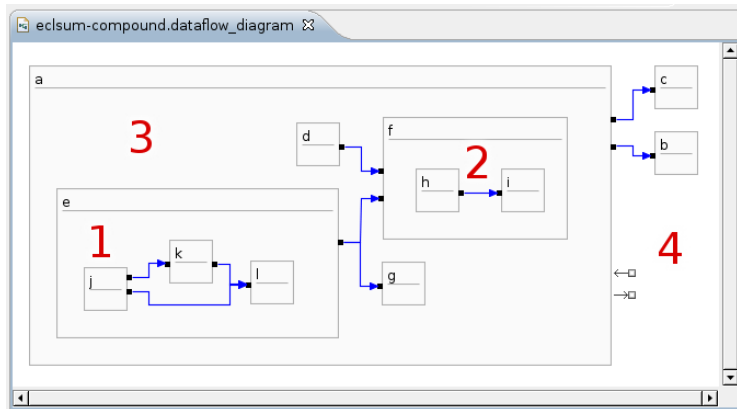
Custom Layout Options

- To be implemented soon...



Compound Structures

- Apply layout providers recursively



Meta Layout



- ▶ Layout providers work on an internal graph structure generated with EMF

Diagram Layout

- ▶ Layout providers work on an internal graph structure generated with EMF
- ▶ Need to map the contents of a diagram to the internal structure and back

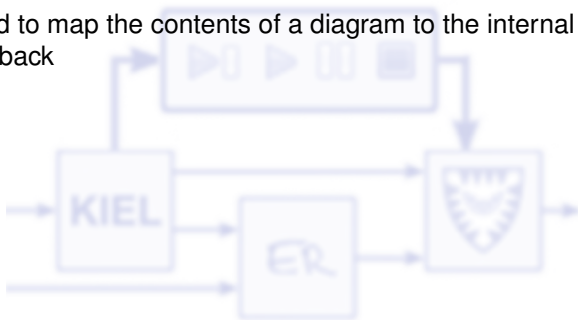


Diagram Layout

- ▶ Layout providers work on an internal graph structure generated with EMF
- ▶ Need to map the contents of a diagram to the internal structure and back
- ▶ Done by **Layout Managers** using the GEF command / request pattern

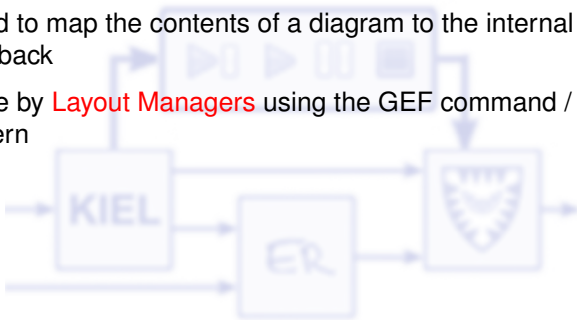


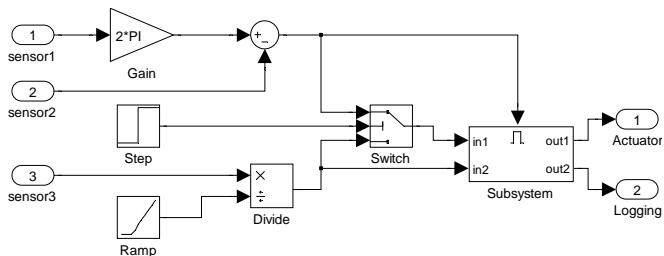
Diagram Layout

- ▶ Layout providers work on an internal graph structure generated with EMF
- ▶ Need to map the contents of a diagram to the internal structure and back
- ▶ Done by **Layout Managers** using the GEF command / request pattern
- ▶ Currently only a layout manager for GMF is implemented
 - ▶ Analyzes the edit part structure at runtime
 - ▶ Recursively go into the contents of an edit part to explore compound structures

Diagram Layout

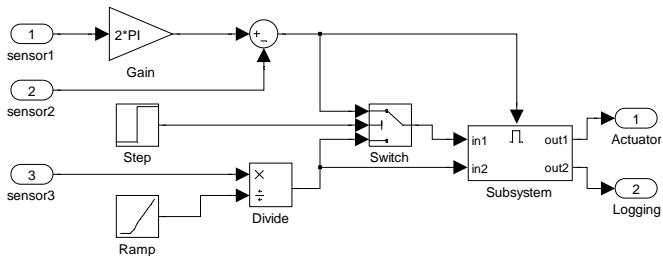
- ▶ Layout providers work on an internal graph structure generated with EMF
- ▶ Need to map the contents of a diagram to the internal structure and back
- ▶ Done by **Layout Managers** using the GEF command / request pattern
- ▶ Currently only a layout manager for GMF is implemented
 - ▶ Analyzes the edit part structure at runtime
 - ▶ Recursively go into the contents of an edit part to explore compound structures
- ▶ Extension to other diagram editor generation frameworks is possible

Data Flow Diagrams



- Operators exchange data through **ports**

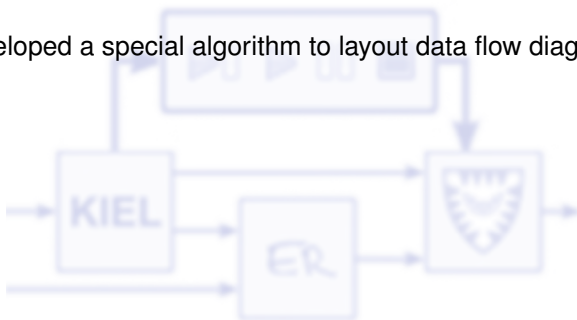
Data Flow Diagrams



- ▶ Operators exchange data through **ports**
- ▶ Layout algorithms must respect these ports when routing connections

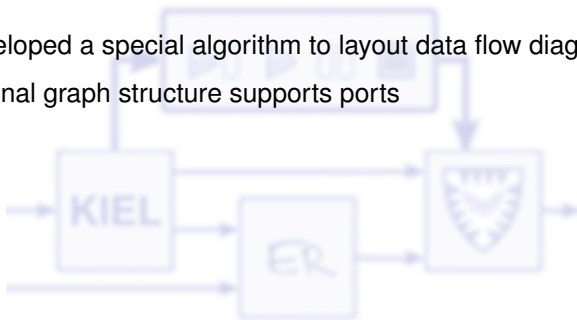
Data Flow Diagrams: Special Algorithm

- Developed a special algorithm to layout data flow diagrams



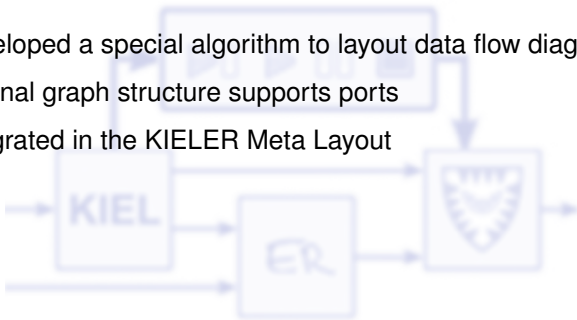
Data Flow Diagrams: Special Algorithm

- ▶ Developed a special algorithm to layout data flow diagrams
- ▶ Internal graph structure supports ports



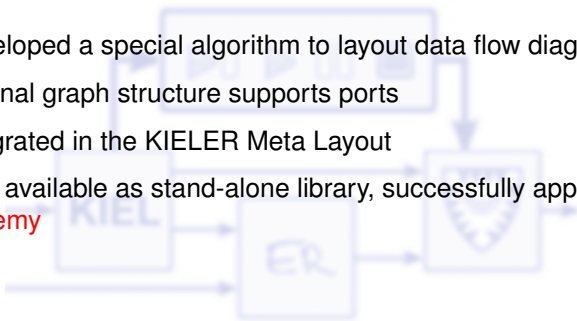
Data Flow Diagrams: Special Algorithm

- ▶ Developed a special algorithm to layout data flow diagrams
- ▶ Internal graph structure supports ports
- ▶ Integrated in the KIELER Meta Layout



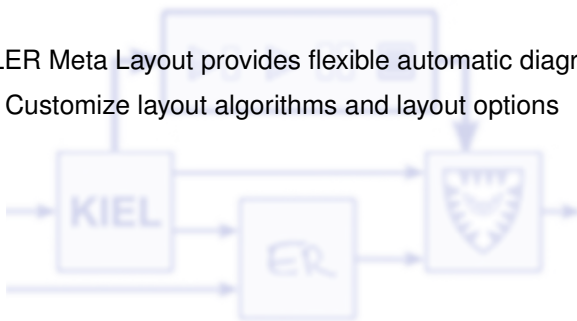
Data Flow Diagrams: Special Algorithm

- ▶ Developed a special algorithm to layout data flow diagrams
- ▶ Internal graph structure supports ports
- ▶ Integrated in the KIELER Meta Layout
- ▶ Also available as stand-alone library, successfully applied to **Ptolemy**



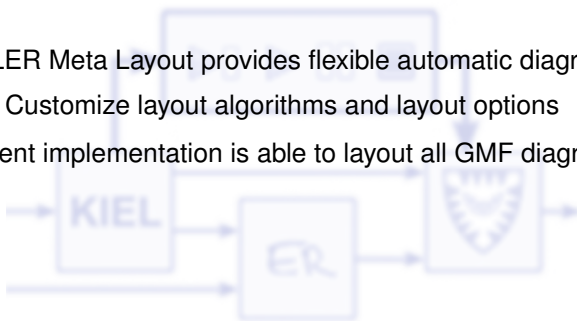
Summary

- ▶ KIELER Meta Layout provides flexible automatic diagram layout
 - ▶ Customize layout algorithms and layout options



Summary

- ▶ KIELER Meta Layout provides flexible automatic diagram layout
 - ▶ Customize layout algorithms and layout options
- ▶ Current implementation is able to layout all GMF diagrams



- ▶ KIELER Meta Layout provides flexible automatic diagram layout
 - ▶ Customize layout algorithms and layout options
- ▶ Current implementation is able to layout all GMF diagrams
- ▶ Implemented a special layout algorithm for data flow diagrams

