Executing SyncCharts with Ptolemy

Christian Motika

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

SYNCHRON Workshop 2010
Frejús, 30.11.2010
KIENER Semiotics

Semantics

Syntax

Pragmatics

Automatic Layout

Dynamic Views

View Management

Semiotics
KIILER Semiotics

Semiotics

Execution Manager
CodeGen / Sim
UML State Machines
Ptolemy (KlePto)
Semantics
SyncCharts
Data Flow
Actor Oriented
UML
Syntax

Pragmatics

Dynamic Views

Automatic Layout

View Management
KIILER Semiotics

Semiotics

Execution Manager

Semantics

CodeGen/Sim

SyncCharts

Data Flow

Syntax

Actor Oriented

UML

Ptolemy (KlePto)

UML State Machines

Synchronous C/Java

SCC

CEC

Pragmatics

Automatic Layout

Dynamic Views

View Management

Christian Motika

Executing SyncCharts with Ptolemy
Overview

- KIELER
  - Overview
  - SyncCharts
Overview

- KIELER
  - Overview
  - SyncCharts

- Ptolemy
  - Heterogenous Modeling
  - ModalModel and SR Domain
Overview

- KIELER
  - Overview
  - SyncCharts
- Ptolemy
  - Heterogenous Modeling
  - ModalModel and SR Domain
- KIELER leveraging Ptolemy
  - Simulation Approach
  - Transformations
  - Eclipse Integration
Overview

- KIELER
  - Overview
  - SyncCharts

- Ptolemy
  - Heterogenous Modeling
  - ModalModel and SR Domain

- KIELER leveraging Ptolemy
  - Simulation Approach
  - Transformations
  - Eclipse Integration

- Summary
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
- Modeling platform and test bed
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
- Modeling platform and test bed
  - Improve pragmatics
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
- Modeling platform and test bed
  - Improve pragmatics
- Open source and Eclipse based (plug-ins)
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
- Modeling platform and test bed
  - Improve pragmatics
- Open source and Eclipse based (plug-ins)
- General concepts:
What is KIELER?

- Kiel Integrated Environment for Layout Eclipse Rich Client
- Modeling platform and test bed
  - Improve pragmatics
- Open source and Eclipse based (plug-ins)
- General concepts:
  - Generic approaches
  - Symbiosis w/ Eclipse technologies (e.g., EMF, GMF, TMF, Xpand, Xtend)
  - Interfaces to other tools (Ptolemy, Papyrus)
SyncCharts

- Statechart dialect
- Mealy machine

Charles André, Computing SyncCharts Reactions, 2003
SyncCharts

- Statechart dialect
- Mealy machine with
  - Parallelism, hierarchy, compound events, broadcast

Charles André, Computing SyncCharts Reactions, 2003
SyncCharts

- Statechart dialect
- Mealy machine with
  - Parallelism, hierarchy, compound events, broadcast
- Graphical notation for the Esterel synchronous language

Charles André, Computing SyncCharts Reactions, 2003
SyncCharts

- Statechart dialect
- Mealy machine with
  - Parallelism, hierarchy, compound events, broadcast
- Graphical notation for the Esterel synchronous language
- Synchrony hypothesis
  - Discrete ticks
  - Computations take no time

Charles André, Computing SyncCharts Reactions, 2003
Abstract Syntax (EMF)
Overview

- KIELER
  - Overview
  - SyncCharts
- Ptolemy
  - Heterogenous Modeling
  - ModalModel and SR Domain
- KIELER leveraging Ptolemy
  - Simulation Approach
  - Transformations
  - Eclipse Integration
- Summary
Ptolemy

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

Introduction to Ptolemy II, UC Berkeley
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
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
- Ptolemy models are a set of interacting components → Actor-Oriented Design
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

- Ptolemy models are a set of interacting components \(\rightarrow\) **Actor-Oriented Design**

- Constructed under a **model of computation** (MoC)
Ptolemy Actor Example

Introduction to Ptolemy II, UC Berkeley, 2008
Model of Computation

- Defines interaction of system components
Model of Computation

- Defines interaction of system components
  - Semantics of a model
Model of Computation

- Defines interaction of system components
  - Semantics of a model
- Ptolemy Model can have more than one MoC
Model of Computation

- Defines interaction of system components
  - Semantics of a model
- Ptolemy Model can have more than one MoC
- MoC domains/directors:
  - Process Networks (PN)
Model of Computation

- Defines interaction of system components
  - Semantics of a model
- Ptolemy Model can have more than one MoC
- MoC domains/directors:
  - Process Networks (PN)
  - Continuous Time (CT)
Model of Computation

- Defines interaction of system components
  - Semantics of a model
- Ptolemy Model can have more than one MoC
- MoC domains/directors:
  - Process Networks (PN)
  - Continuous Time (CT)
  - Finite State Machines (FSM)
  - Synchronous Reactive (SR)
Model of Computation

- Defines interaction of system components
  - Semantics of a model
- Ptolemy Model can have more than one MoC
- MoC domains/directors:
  - Process Networks (PN)
  - Continuous Time (CT)
  - Finite State Machines (FSM)
  - Synchronous Reactive (SR)
  - ...

Christian Motika
Executing SyncCharts with Ptolemy
ModalModel Domain

- Entities not actors but states
ModalModel Domain

- Entities not actors but states
- Execution: Strictly ordered sequence of state transitions
ModalModel Domain

- Entities not actors but states
- Execution: Strictly ordered sequence of state transitions
- Build-in expression language to evaluate guards
ModalModel Domain

- Entities not actors but states
- Execution: Strictly ordered sequence of state transitions
- Build-in expression language to evaluate guards
- Refinements (multiple)
ModalModel Domain

- Entities not actors but states
- Execution: Strictly ordered sequence of state transitions
- Build-in expression language to evaluate guards
- Refinements (multiple)
- Reset and preemptive transitions
Synchronous Reactive Domain

- Zero-Delay blocks
Synchronous Reactive Domain

- Zero-Delay blocks
- Instantaneous communication
Synchronous Reactive Domain

- Zero-Delay blocks
- Instantaneous communication
- Feedback
Synchronous Reactive Domain

- Zero-Delay blocks
- Instantaneous communication
- Feedback
- Fixed point $\Leftrightarrow$ Stable state
Synchronous Reactive Domain

- Zero-Delay blocks
- Instantaneous communication
- Feedback
- Fixed point $\iff$ Stable state
- Values from flat lattice
Synchronous Reactive Domain

- Zero-Delay blocks
- Instantaneous communication
- Feedback

- Fixed point $\Leftrightarrow$ Stable state
- Values from flat lattice
- Determinism $\Leftrightarrow$ Unique solution
Overview

- KIELER
  - Overview
  - SyncCharts
- Ptolemy
  - Heterogenous Modeling
  - ModalModel and SR Domain
- KIELER leveraging Ptolemy
  - Simulation Approach
  - Transformations
  - Eclipse Integration
- Summary
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  Mealy machine
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel

- Interesting:
  - Implicit broadcast vs. explicit signal representation
  - Signal coherence (must/cannot analysis)
  - Transition priorities
  - Normal termination
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality

- Interesting:
  - Implicit broadcast vs. explicit signal representation
  - Signal coherence (must/cannot analysis)
  - Transition priorities
  - Normal termination
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality ↔ Concurrent Actors (inherent)
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality ↔ Concurrent Actors (inherent)
  - Hierarchy

Interesting:
- Implicit broadcast vs. explicit signal representation
- Signal coherence (must/cannot analysis)
- Transition priorities
- Normal termination
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality ↔ Concurrent Actors (inherent)
  - Hierarchy ↔ Compound Actors, state refinements
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality ↔ Concurrent Actors (inherent)
  - Hierarchy ↔ Compound Actors, state refinements
  - Compound events

Interesting:
- Implicit broadcast vs. explicit signal representation
- Signal coherence (must/cannot analysis)
- Transition priorities
- Normal termination
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine ↔ ModalModel
  - Orthogonality ↔ Concurrent Actors (inherent)
  - Hierarchy ↔ Compound Actors, state refinements
  - Compound events ↔ Expression language

- Interesting:
Mapping SyncCharts to Ptolemy:

- Mealy machine $\leftrightarrow$ ModalModel
- Orthogonality $\leftrightarrow$ Concurrent Actors (inherent)
- Hierarchy $\leftrightarrow$ Compound Actors, state refinements
- Compound events $\leftrightarrow$ Expression language

Interesting:

- Implicit broadcast vs. explicit signal representation
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine $\leftrightarrow$ ModalModel
  - Orthogonality $\leftrightarrow$ Concurrent Actors (inherent)
  - Hierarchy $\leftrightarrow$ Compound Actors, state refinements
  - Compound events $\leftrightarrow$ Expression language

- Interesting:
  - Implicit broadcast vs. explicit signal representation
  - Signal coherence (must/cannot analysis)
Ptolemy Simulation Engine

- Mapping SyncCharts to Ptolemy:
  - Mealy machine $\leftrightarrow$ ModalModel
  - Orthogonality $\leftrightarrow$ Concurrent Actors (inherent)
  - Hierarchy $\leftrightarrow$ Compound Actors, state refinements
  - Compound events $\leftrightarrow$ Expression language

- Interesting:
  - Implicit broadcast vs. explicit signal representation
  - Signal coherence (must/cannot analysis)
  - Transition priorities
Mapping SyncCharts to Ptolemy:
- Mealy machine $\leftrightarrow$ ModalModel
- Orthogonality $\leftrightarrow$ Concurrent Actors (inherent)
- Hierarchy $\leftrightarrow$ Compound Actors, state refinements
- Compound events $\leftrightarrow$ Expression language

Interesting:
- Implicit broadcast vs. explicit signal representation
- Signal coherence (must/cannot analysis)
- Transition priorities
- Normal termination
Transformation Example: Parallelism and Signals

parallelism

Interface: L,

s1 \xrightarrow{L} s2

s3 \xrightarrow{L} s4
Transformation Example: Parallelism and Signals
Transformation Example: Hierarchy

```
interface

s1

s2

s3
```

```
file://C/Program%20Files/eclipse-galileo-x64/generated.mxml

hierarchy_918156682_region_1

O_COMBINE

SR Director

s1_200586989

s2_200586688

O

hierarchy_200586888_region_1

O_COMBINE

SR Director

s3_565826381

output: O=1
```
Schematic Overview

- PtolemyMM
- SyncChartsMM
- M2M description
- SyncChart 2 Ptolemy
- Xtend M2M
- Ptolemy Simulator
  - Data Producer
  - Data Observer
  - metamodels
  - model to simulate
  - simulation data, model outputs
  - commands, model inputs
- Execution Manager

Christian Motika
Executing SyncCharts with Ptolemy
Architecture and User Interface
Architecture and User Interface

![Diagram showing the architecture and user interface of Ptolemy II Environment.]
KI ELER KlePto Simulation Demo
Summary

- KIELER
Summary

- KIELER
- Ptolemy
Summary

- KIELER
- Ptolemy
- KIELER leveraging Ptolemy
  - KlePto concept
Summary

- KIELER
- Ptolemy
- KIELER leveraging Ptolemy
  - KlePto concept
  - Construct runnable Ptolemy models for EMF based models (Xtend)
Summary

- KIELER
- Ptolemy
- KIELER leveraging Ptolemy
  - KlePto concept
  - Construct runnable Ptolemy models for EMF based models (Xtend)
  - Ptolemy integration in Eclipse
Summary

- KIELER
- Ptolemy
- KIELER leveraging Ptolemy
  - KlePto concept
  - Construct runnable Ptolemy models for EMF based models (Xtend)
  - Ptolemy integration in Eclipse
  - Infrastructure for interactive model execution
Summary

- KIELER
- Ptolemy
- KIELER leveraging Ptolemy
  - KlePto concept
  - Construct runnable Ptolemy models for EMF based models (Xtend)
  - Ptolemy integration in Eclipse
  - Infrastructure for interactive model execution
  - Also: Visualization, stepwise transformation, model checking, online debugging, regression tests, validation, ...
To Go Further

**ANDRÉ, C.**

Computing SyncCharts reactions.


**MOTIKA, C., FUHRMANN, H., AND VON HANXLEDEN, R.**

Semantics and execution of domain specific models.


**UC BERKELEY, EECS DEPT.**

Ptolemy webpage.

http://ptolemy.eecs.berkeley.edu/.

**UNI KIEL, REAL-TIME AND EMBEDDED SYSTEMS GROUP.**

KI ELER webpage.

http://www.informatik.uni-kiel.de/en/rtsys/kieler/.
Thank you for your attention and participation!

Any questions or suggestions?
public class DataComponent extends JSONObjectDataComponent
    implements IJSONObjectDataComponent {

    boolean doneI;

    public void initialize() {
        doneI = false;
    }

    public boolean isObserver() { return true; }
    public boolean isProducer() { return true; }

    public JSONObject step(JSONObject jSONObject)
        throws KiemExecutionException {
        JSONObject returnObj = new JSONObject();
        if (!doneI && jSONObject.has("I")
            && (JSONSignalValues.isPresent(jSONObject.get("I")))) {
            //change state to doneI when signal I is present
            doneI = true;
            //output signal O
            returnObj.accumulate("O", JSONSignalValues.newValue(true));
        }
        return returnObj;
    }
}
Synchronous/Reactive Modeling Example: TrafficLight

This model illustrates a typical design pattern where the top level is a DE model of the physical environment for a system under design. The next level down is a modal model fashioned after the statecharts model at the right. Open the TrafficLight actor to see how it is implemented.

The PoissonClock actor occasionally injects an Error signal. The Error condition then lasts 5 seconds, as determined by the TimedDelay actor.

Authors: Reinhard von Hanxleden, Huining Feng, and Edward A. Lee
Synchronous/Reactive Modeling Example: TrafficLight

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.

Note that we are following the design of the Statecharts model shown on the top level, but there is a flaw in the 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 otherwise arranged. 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.
Synchronous/Reactive Modeling Example: TrafficLight

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 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 Wireless Deployment example uses the same instances.
Synchronous/Reactive Modeling Example: TrafficLight

This state machine controls the car lights. It uses the count variable to stay red for three seconds and to stay green for two seconds.
M2M Transformation Results
Transformation 1: Simple States
Transformation 2: Transitions (Priorities)
Transformation 3: Parallelism and Signals
Transformation 4: Inputs and Outputs
Transformation 5: Hierarchy
```java
public class TestLoadPtolemyModel {
    public static void main(String[] args) {
        URI momlFile = URI.createFileURI(new File("kielerio.moml").getAbsolutePath());
        MoMLParser parser = new MoMLParser();
        NamedObj ptolemyModel = null;
        //load & parse model
        ptolemyModel = parser.parse(null, new URL(momlFile.toString()));
        //execute model
        CompositeActor actor = ((CompositeActor) ptolemyModel);
        //create a manager
        Manager manager = actor.getManager();
        if (manager == null) {
            manager = new Manager(actor.workspace(), "kieler manager");
            actor.setManager(manager);
        }
        // run the model
        if (manager != null) {
            List<Actor> children = actor.getChildren();
            manager.initialize();
            for (int i = 0; i < 100; i++) {
                manager.iterate();
                // LISTEN FOR OUTPUT TO KIELER HERE //
            }
            manager.wrapup();
        }
    }
}
```
public class TestLoadPtolemyModel {
    public static void main(String[] args) {
        //load & parse model
        //execute model
        //create a manager

        //insert user data
        Iterator<Object> childrenIterator = actor.containedObjectsIterator();
        while (childrenIterator.hasNext()) {
            Object child = childrenIterator.next();
            //search for KielerIO ports
            if (child instanceof KielerIO) {
                KielerIO kielerIO = (KielerIO)child;
                System.out.println(kielerIO.getSignalName());
                kielerIO.setValue(2);
                kielerIO.setPresent(true);
                kielerIO.setPermanent(true);
            }
        }

        // run the model
    }
}
public class KielerIO extends TypedAtomicActor {
    public Parameter value;
    public TypedIOPort signal;

    public void setValue(int value) {
        this.value.setExpression(value+"");
    }

    public void fire() throws IllegalActionException {
        if (trigger.getWidth() > 0) {
            if (trigger.hasToken(0)) {
                trigger.get(0);
            }
        }

        if (present.getValueAsString().equals("true")) {
            int tokenValue = Integer.valueOf(value.getValueAsString());
            signal.send(0, new IntToken(tokenValue));
            if (permanent.getValueAsString().equals("false")) {
                this.setPresent(false);
            }
        }
        super.fire();
    }
}
Moml.ecore - Meta Model (EMF)
Determine Signal Assignment

- Signal Coherence Law
  - A Signal can be either present or absent within a tick but not both at the same time.

- Ensure by lattice and ternary logic

![Diagram]

- Use extended fixed point iteration:
  1. Start with unknown local and output signals
  2. Determine which signals must be emitted (set to present)
     → SR director fixed point iteration (send token)
  3. Determine which signals cannot be emitted (set to absent)
     → FSM director (send clear)
Linear Scheduler
Scheduling

listP := dataComponentList.select(type == producerOnly)
listAll := dataComponentList
prod := listP.getFirst()
listP.removeFirst()
not listP.empty()
comp := listAll.getFirst()
listAll.removeFirst()
listP.empty()
prod.asyncStartStep()
comp.syncStep()
comp.type == observerProducer
comp.type == observerOnly
comp.syncWaitEndStep()
comp.type == producerOnly
comp.asyncStep()
comp.isReady()
not comp.isReady()
not listAll.empty()
listAll.empty()
Ptolemy Meta Model (EMF)