Actor-Oriented Design and The Ptolemy II framework

http://ptolemy.eecs.berkeley.edu/
Ptolemy II objectives

- Supports modeling, simulation and design of concurrent systems
- Promotes component-based modeling, where a component represents a domain-specific entity and it is called an actor
- Provides widely-used models of interaction between components, called models of computation
- Advocates a programming discipline called Actor-Oriented Programming
- Focuses on flexibility
- Encourages experimentation with designs
Actor Oriented Design

- Actors are conceptually concurrent (no predefined order of execution)
- Actors interact by sending messages through channels
- An actor implements an execution interface
- Dedicated components are responsible for data transfer between actors and for executing the actors
- Actors can be hierarchically composed
AOD versus OOD* (I)

Object orientation:

- **Class name**
- **Data**
- **Methods**

What flows through an object is sequential control.

Actor orientation:

- **Actor name**
- **Data (state)**
- **Parameters**
- **Ports**

What flows through an object is streams of data.

*Edward Lee, UC Berkeley
Identified limitations of object orientation:

- Says little or nothing about concurrency and time
- Concurrency typically expressed with threads, monitors, semaphores
- Components tend to implement low-level communication protocols
  - Re-use potential is disappointing

**Object oriented**

<table>
<thead>
<tr>
<th>TextToSpeech</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize(): void</td>
</tr>
<tr>
<td>notify(): void</td>
</tr>
<tr>
<td>isReady(): boolean</td>
</tr>
<tr>
<td>getSpeech(): double[]</td>
</tr>
</tbody>
</table>

**Actor oriented**

Actor oriented interface definition says “Give me text and I’ll give you speech”

OO interface definition gives procedures that have to be invoked in an order not specified as part of the interface definition.
Examples of Actor-Oriented component frameworks*

- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- OPNET (Opnet Technologies)
- Polis & Metropolis (UC Berkeley)
- Gabriel, Ptolemy, and Ptolemy II (UC Berkeley)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Easy5 (Boeing)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- …

* Edward Lee, UC Berkeley
Live examples of actor-oriented designs

- Matlab/Simulink: Automotive controllers
- Ptolemy: Finite State Machine example
Ptolemy model example (I)
Ptolemy model example (II)
Ptolemy model example (III)
Ptolemy model example (IV)

- ParameterSet
- On
- Off
- guard: off_isPresent
- fareUpdate
- tariff
- guard: on_isPresent && off_isPresent
- output: fareUpdate = startingFareEUR
- guard: speed_isPresent && speed < spdDownThr
- output: tariff = this.getDirector().getModelTime().toString() + ": Time"
- fareUpdate
- guard: speed_isPresent && speed > spdUpThr
- output: tariff = this.getDirector().getModelTime().toString() + ": Distance"
- spdDownThr: 2
- spdUpThr: 8.0
Ptolemy model example (V)
Ptolemy Kernel: abstract syntax

- Clustered graph
- Entities
- Ports
- Relations
- Attributes
Ptolemy model example
MoML: XML schema for Ptolemy models

```xml
<entity name="tm3" class="ptolemy.actor.TypedCompositeActor">
    <property name="Continuous Director" class="ptolemy.domains.continuous.kernel.ContinuousDirector">
    </property>
    <entity name="TaxiMeter" class="ptolemy.actor.TypedCompositeActor">
        <property name="ParameterSet" class="ptolemy.actor.parameters.ParameterSet">
            <property name="fileOrURL" class="ptolemy.data.expr.FileParameter" value="fareDefinition_Salzburg.txt"/>
        </property>
        <port name="speed" class="ptolemy.actor.TypedIOPort">
            <property name="input"/>
            <property name="_type" class="ptolemy.actor.TypeAttribute" value="double"/>
        </port>
        <port name="fare" class="ptolemy.actor.TypedIOPort">
            <property name="output"/>
        </port>
    </entity>
    <entity name="MeteringMode" class="ptolemy.domains.modal.modal.ModalModel">
        <port name="speed" class="ptolemy.domains.modal.modal.ModalPort">
            <property name="input"/>
            <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true"/>
        </port>
        <link port="MeteringMode.fareUpdate" relation="relation5"/>
        <link port="Accumulator.input" relation="relation5"/>
    </entity>
</entity>
```
Hierarchical composition
Kernel classes

**NamedObj**

**Entity**

- **+Entity()**
- **+Entity(name : String)**
- **+Entity(w : Workspace, name : String)**
- **+connectedPorts() : Enumeration**
- **+connectionsChanged(p : Port)**
- **+getPort(name : String) : Port**
- **+getPorts() : Enumeration**
- **+linkedRelations() : Enumeration**
- **+newPort(name : String) : Port**
- **+removeAllPorts()**
- **+_addPort(p : Port)**
- **+_removePort(p : Port)**

**Port**

- **+_container : Entity**
- **+_relationsList : CrossRefList**
- **+Port()**
- **+Port(w : Workspace)**
- **+Port(container : Entity, name : String)**
- **+connectedPorts() : Enumeration**
- **+isLinked(r : Relation) : boolean**
- **+isOpaque() : boolean**
- **+linkedRelations() : Enumeration**
- **+link(r : Relation)**
- **+numLinks() : int**
- **+setContainer(c : Entity)**
- **+unlink(r : Relation)**
- **+unlinkAll()**
- **+_link(r : Relation)**

**Relation**

- **+_portList : CrossRefList**
- **+Relation()**
- **+Relation(name : String)**
- **+Relation(w : Workspace, name : String)**
- **+linkedPorts() : Enumeration**
- **+linkedPorts(except : Port) : Enumeration**
- **+numLinks() : int**
- **+unlinkAll()**
- **+_checkPort(p : Port)**
- **+_getPortList() : CrossRefList**
Actor package: producer/consumer components

- Services in the infrastructure:
  - broadcast
  - multicast
  - busses
  - mutations
  - clustering
  - parameterization
  - typing
  - polymorphism

Basic Transport:

![Diagram of Basic Transport]

- send(0,t)
- receiver.put(t)
- get(0)
- token t
- Actor
- IOPort
- IORelation
- Receiver (inside port)
Object Model for Executable Components

```
+fire()
+initialize()
+postfire() : boolean
+prefire() : boolean
+preinitialize()
+stopFire()
+terminate()
+wrapup()
```

```
+getDirector() : Director
+getExecutiveDirector() : Director
+getManager() : Manager
+inputPortList() : List
+newReceiver() : Receiver
+outputPortList() : List
```

```
+fire()
+initialize()
+postfire() : boolean
+prefire() : boolean
+preinitialize()
+stopFire()
+terminate()
+wrapup()
```

Director

AtomicActor

ComponentEntity

CompositeEntity

CompositeActor

Actor execution interface

- Preinitialization
  - Type inference, scheduling, etc.
- Initialization
  - Set initial state, initial output values, etc.
- Execution
  - Iterate: prefire, fire, postfire
- Finalization
Polymorphic Components - Component Library Works Across Data Types and Domains

- **Data polymorphism:**
  - Add numbers (int, float, double, Complex)
  - Add strings (concatenation)
  - Add composite types (arrays, records, matrices)
  - Add user-defined types

- **Behavioral polymorphism:**
  - In dataflow, add when all connected inputs have data
  - In a time-triggered model, add when the clock ticks
  - In discrete-event, add when any connected input has data, and add in zero time
  - In process networks, execute an infinite loop in a thread that blocks when reading empty inputs
  - In CSP, execute an infinite loop that performs rendezvous on input or output
  - In push/pull, ports are push or pull (declared or inferred) and behave accordingly
  - In real-time CORBA, priorities are associated with ports and a dispatcher determines when to add

* Edward Lee, UC Berkeley

By not choosing among these when defining the component, we get a huge increment in component re-usability. But how do we ensure that the component will work in all these circumstances?
Actor View of Producer/Consumer Components

The send() and get() methods on ports are polymorphic. Their implementation is provided by an object implementing the Receiver interface. The Receiver is supplied by the director and implements the communication semantics of a model of computation.

Basic Transport:

send(0,t) receiver.put(t) get(0)
token t

E1 P1 R1 P2 E2

* Edward Lee, UC Berkeley

Stefan Resmerita, WS2015
Object Model for Communication Infrastructure

+ get() : Token
+ getContainer() : IOPort
+ hasRoom() : boolean
+ hasToken() : boolean
+ put(t : Token)
+ setContainer(port : IOPort)
These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is supplied by the director, not by the component.

Recall: Behavioral polymorphism is the idea that components can be defined to operate with multiple models of computation and multiple middleware frameworks.

* Edward Lee, UC Berkeley
Example Using AO Classes

This model illustrates the mechanisms in Ptolemy II for defining classes and subclasses with inheritance.

This actor is a class definition, indicated by the blue halo. It is ignored by the director and serves as a declaration. To create an instance of this class, right click on the class definition and select "Create Instance" (or type Ctrl-N). To see the class definition, look inside.

* Edward Lee, UC Berkeley
Aspect oriented programming in actor oriented design

- Using the Decorator pattern
- Automotive example:

To use the model, switch the aspects in the relation between the LeadingCar and the FollowingCar:
1. Right-click on the FollowingCar input port
2. Customize -> Configure
3. Enable one of the four Attack Models by ticking the box and setting the inputPort to 'in'.

This model shows a simple adaptive cruise control system, illustrating model-integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.