Actor-Oriented Design and The Ptolemy II framework

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

Ptolemy II objectives

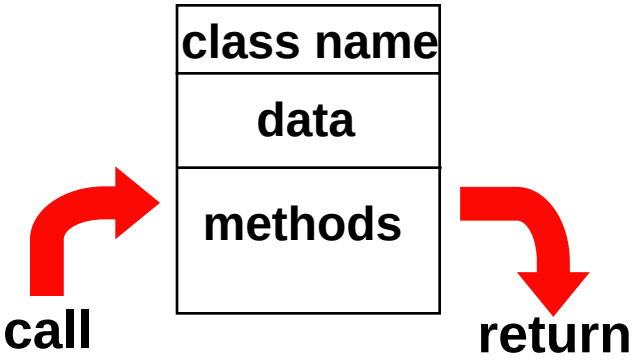
- Supports modeling, simulation and design of concurrent systems
- Promotes component-based modeling, where a component represents a domain-specific entity and it is called an **actor**
- Provides widely-used models of interaction between components, called **models of computation**
- Advocates a programming 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

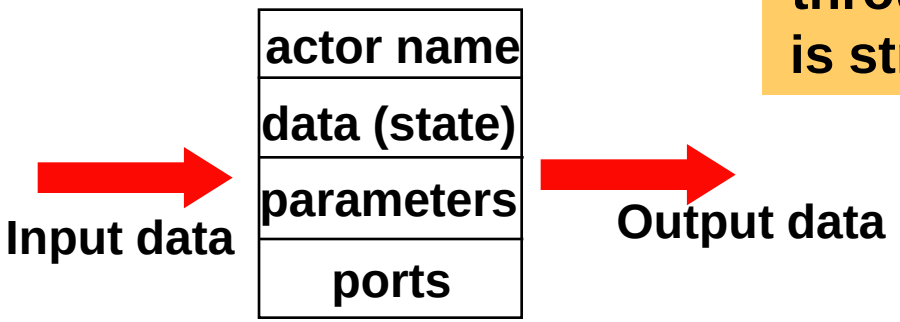
AOD versus OOD* (I)

Object orientation:



What flows through an object is sequential control

Actor orientation:



What flows through an object is streams of data

* Edward Lee, UC Berkeley

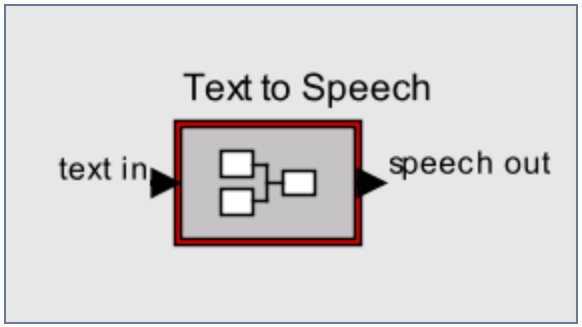
AOD versus OOD* (II)

Object oriented

TextToSpeech
initialize(): void notify(): void isReady(): boolean getSpeech(): double**[]**

OO interface definition gives procedures that have to be invoked in an order not specified as part of the interface definition.

Actor oriented



actor-oriented interface definition says "Give me text and I'll give you speech"

- 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

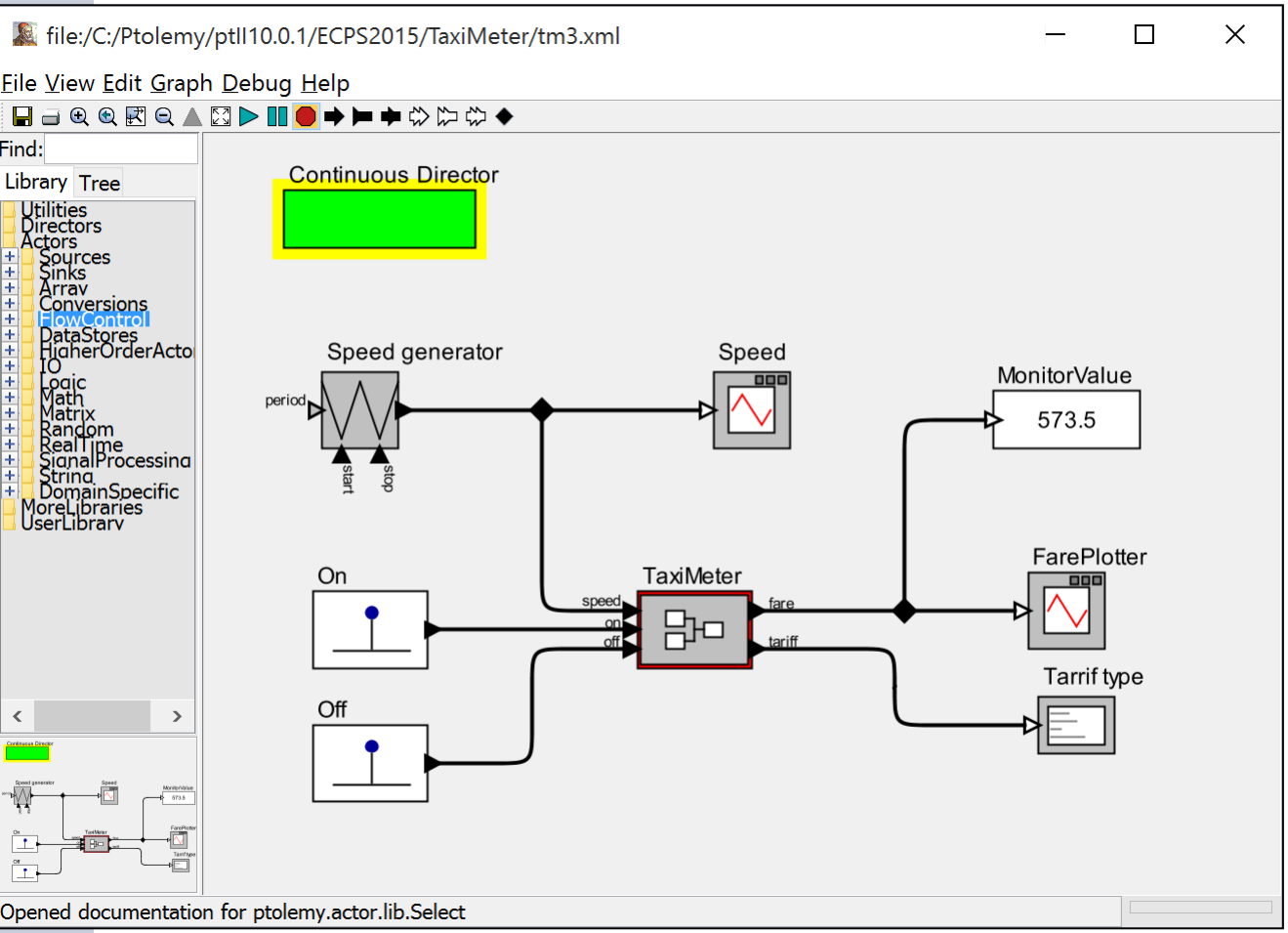
* Edward Lee, UC Berkeley

Examples of Actor-Oriented component frameworks*

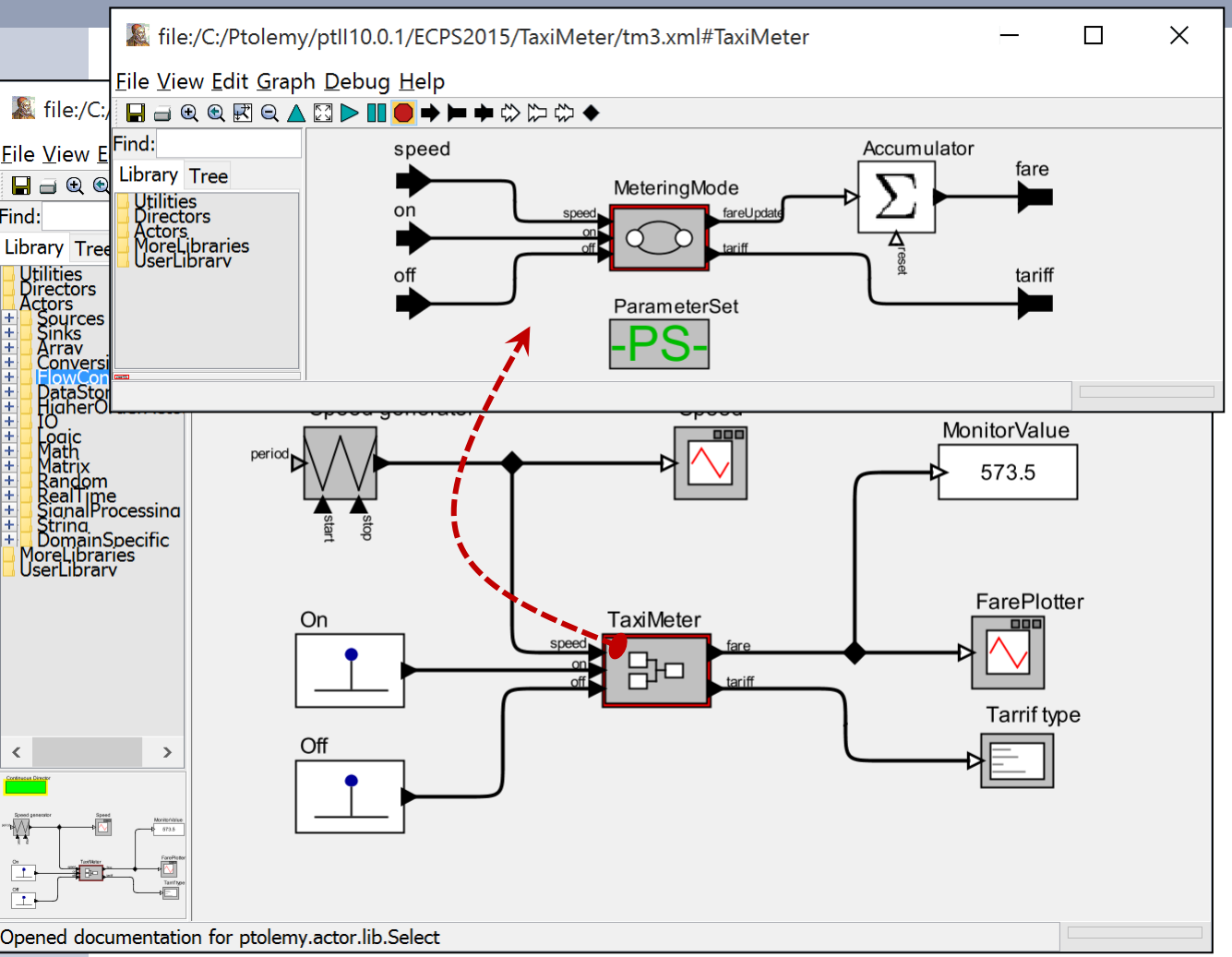
- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- OPNET (Opnet Technologies)
- Polis & Metropolis (UC Berkeley)
- Gabriel, Ptolemy, and Ptolemy II (UC Berkeley)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing 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

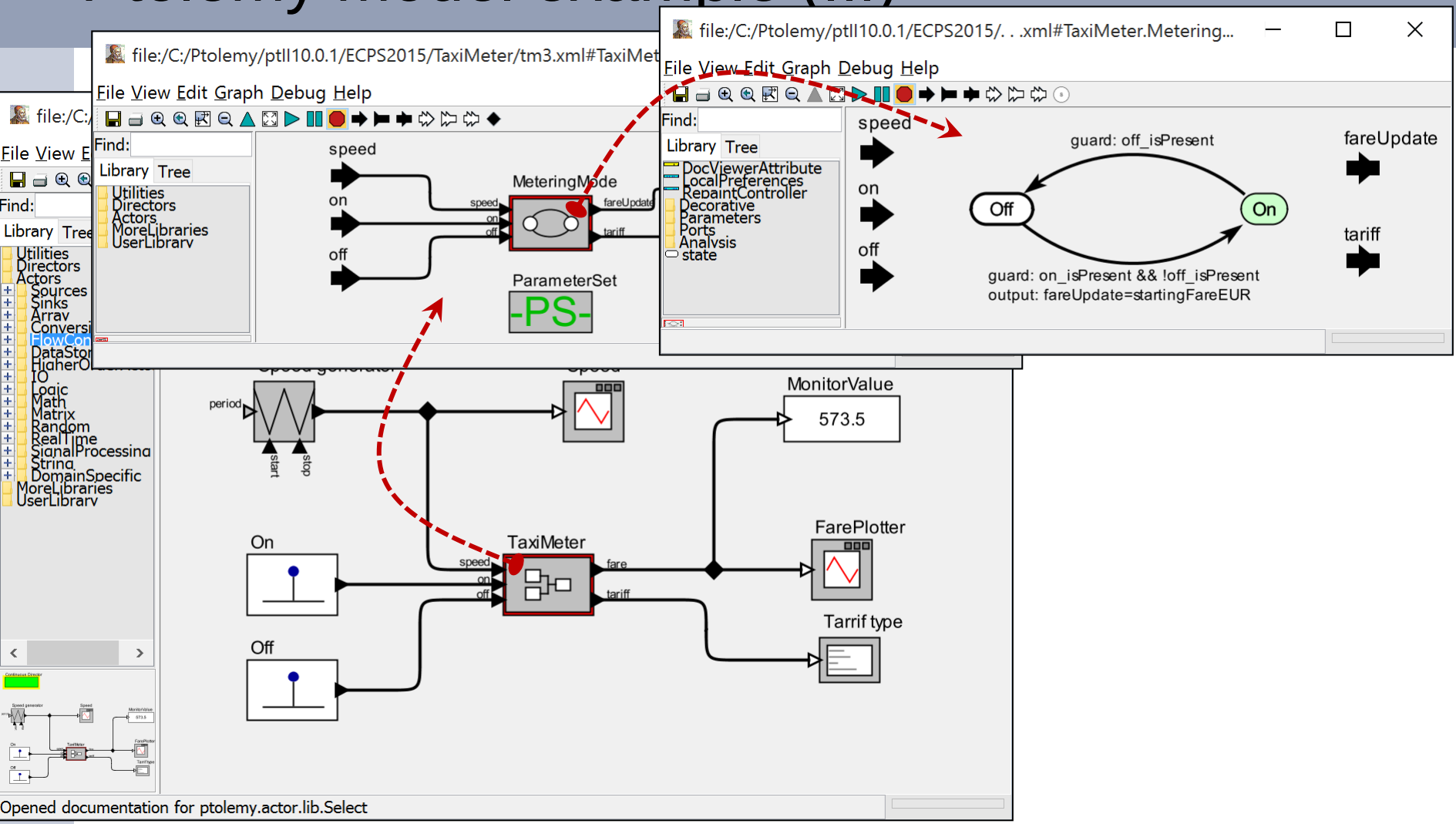
Ptolemy model example (I)



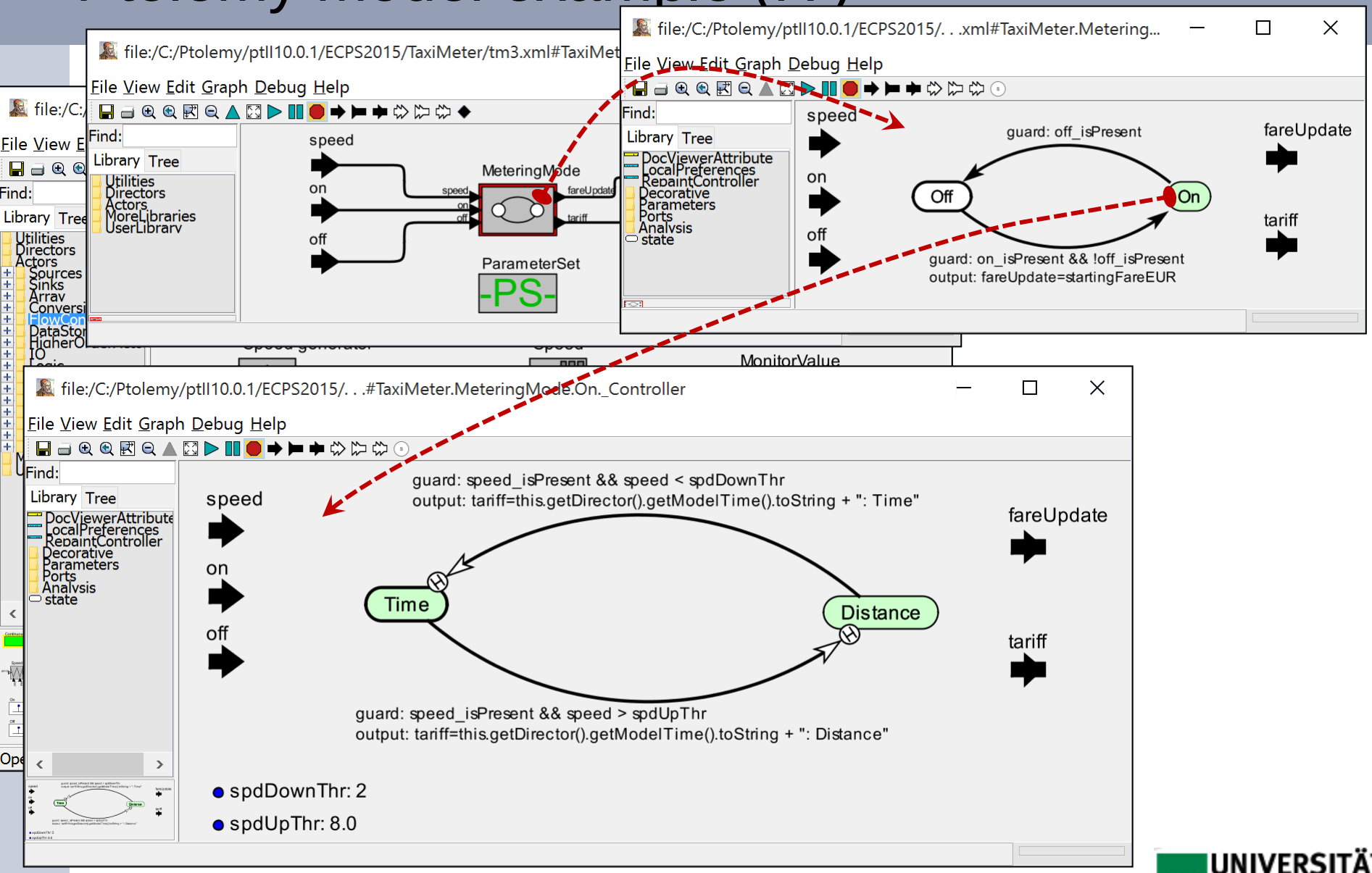
Ptolemy model example (II)



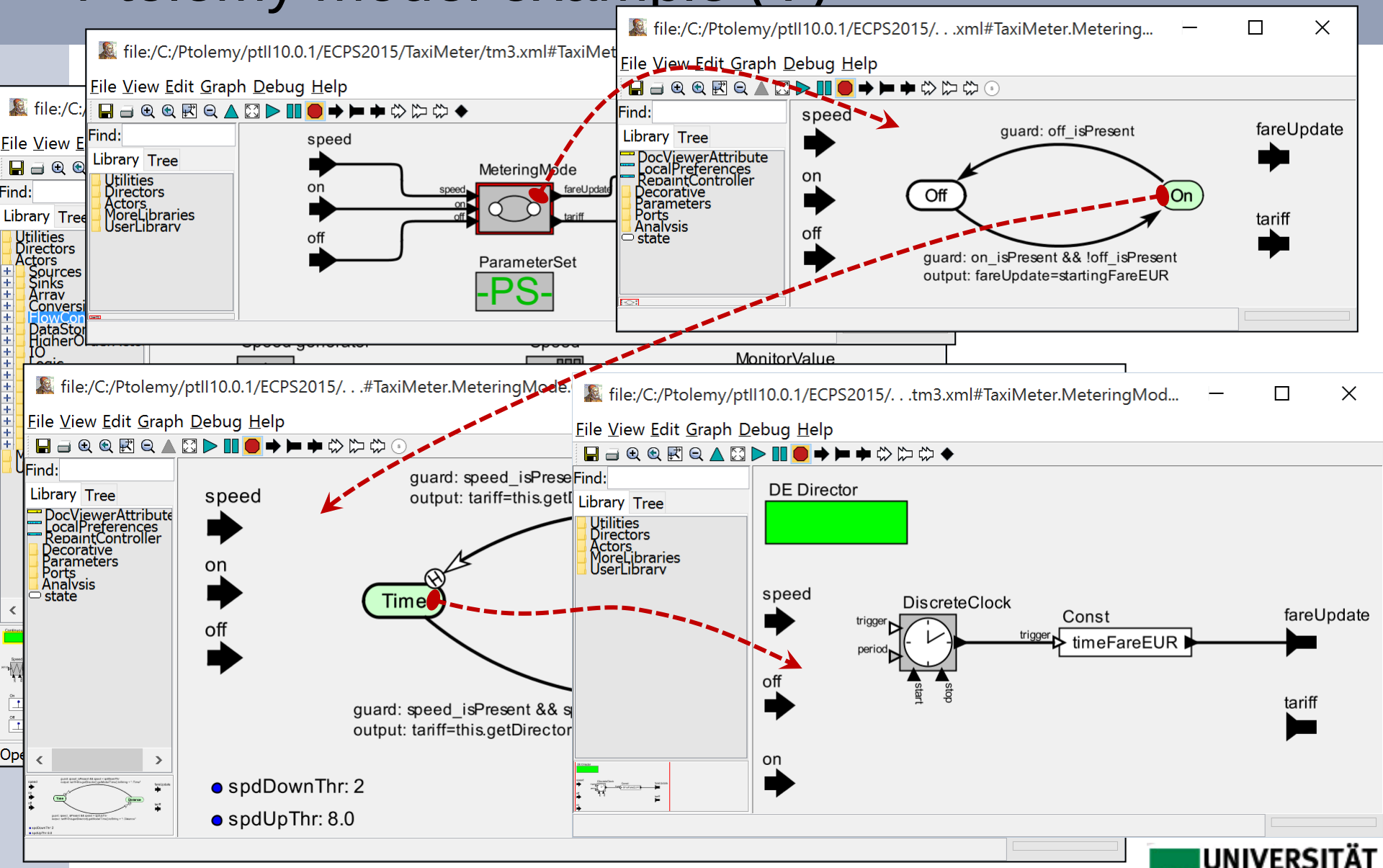
Ptolemy model example (III)



Ptolemy model example (IV)



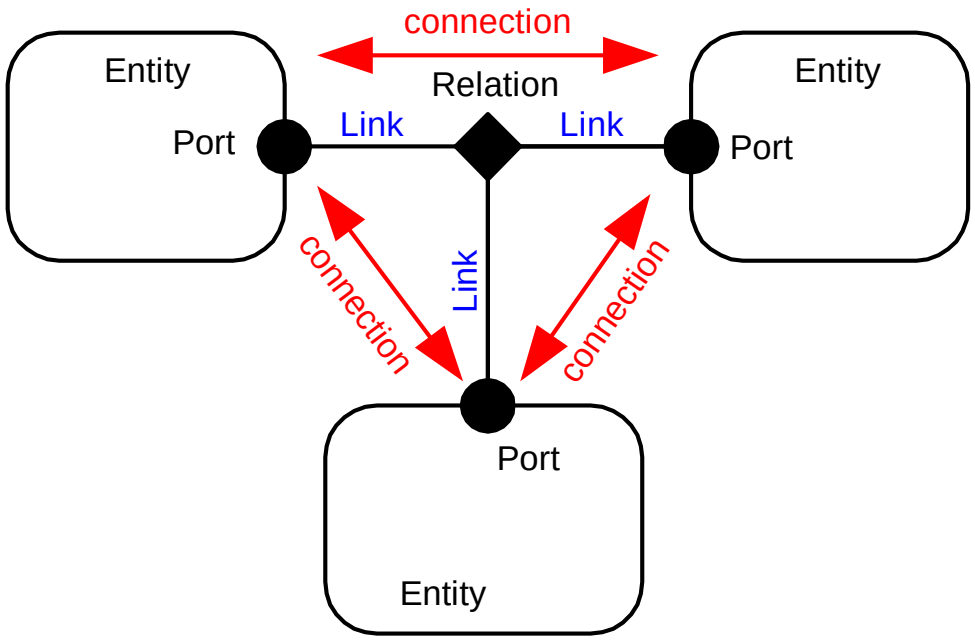
Ptolemy model example (V)



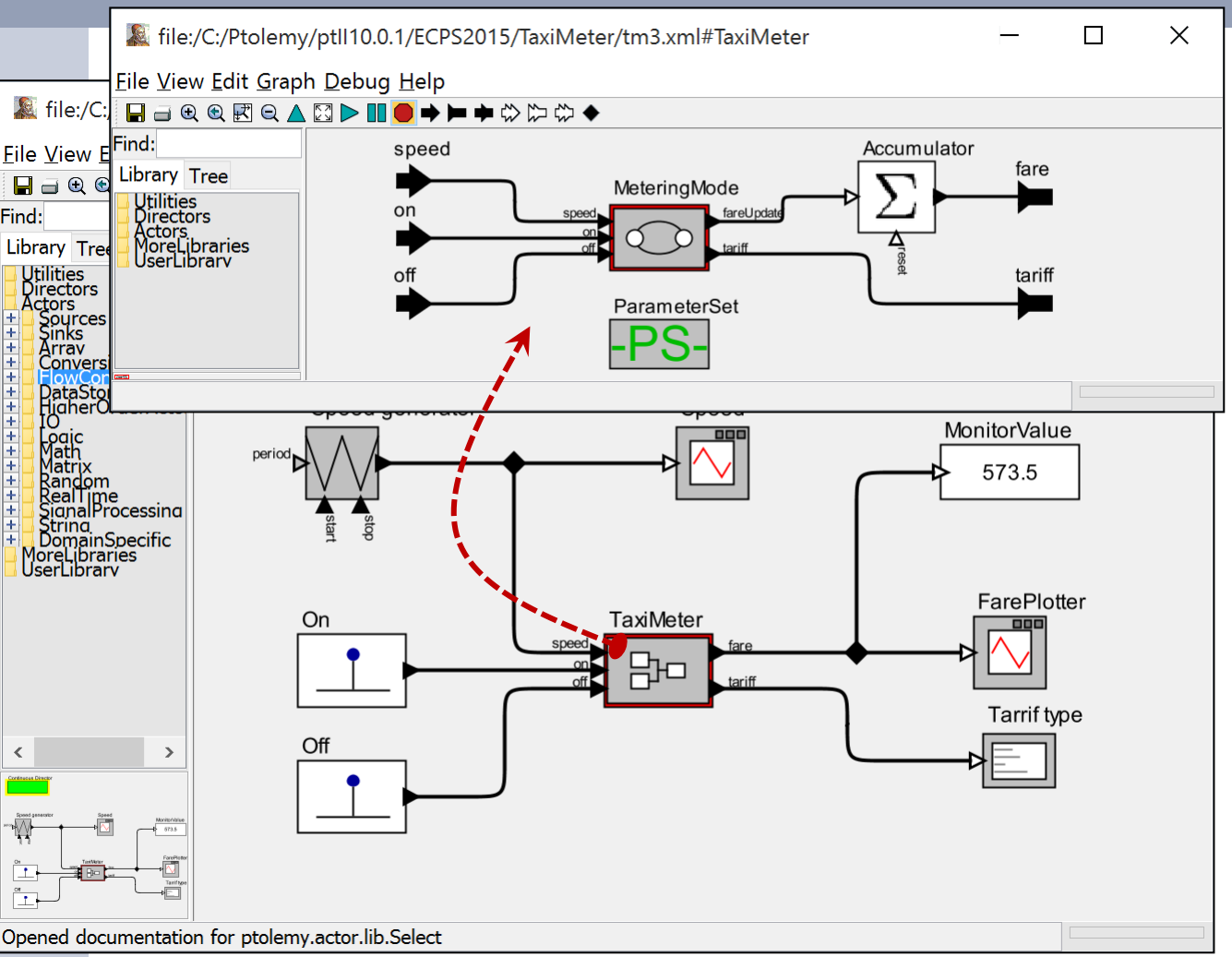
Ptolemy Kernel: abstract syntax

- Clustered graph

- Entities
- Ports
- Relations
- Attributes



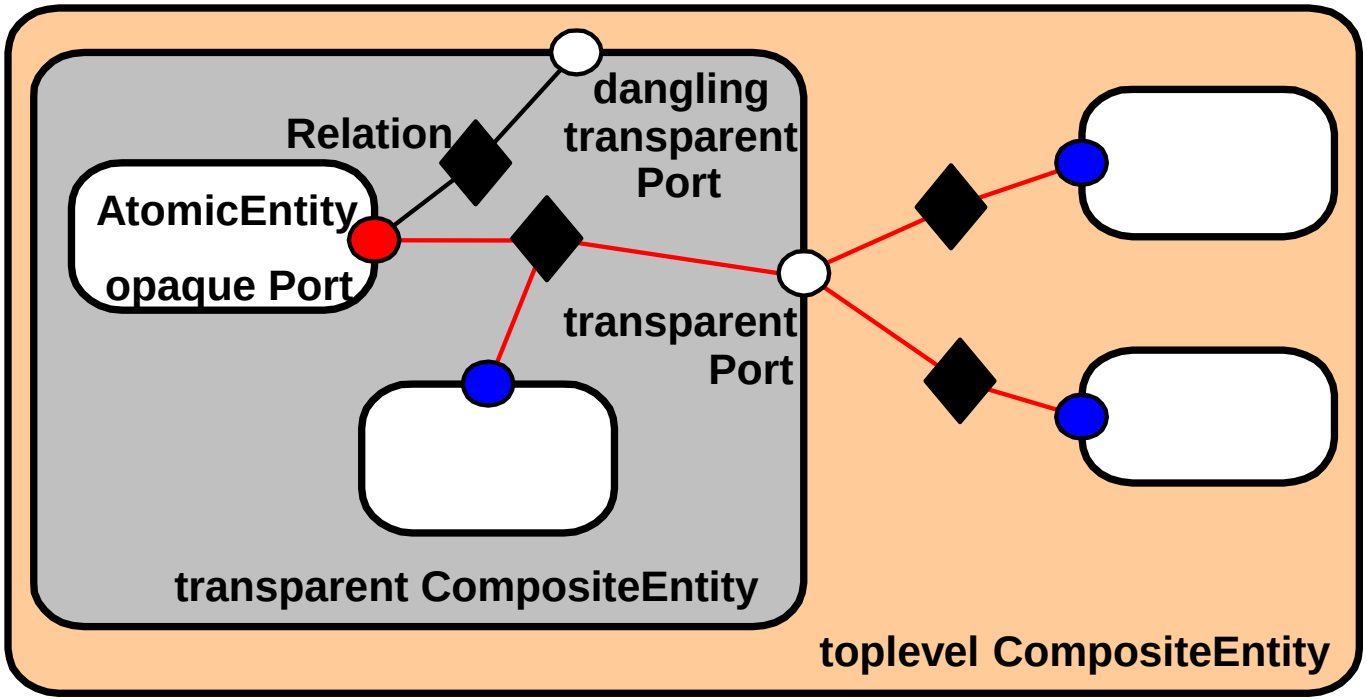
Ptolemy model example



MoML: XML schema for Ptolemy models

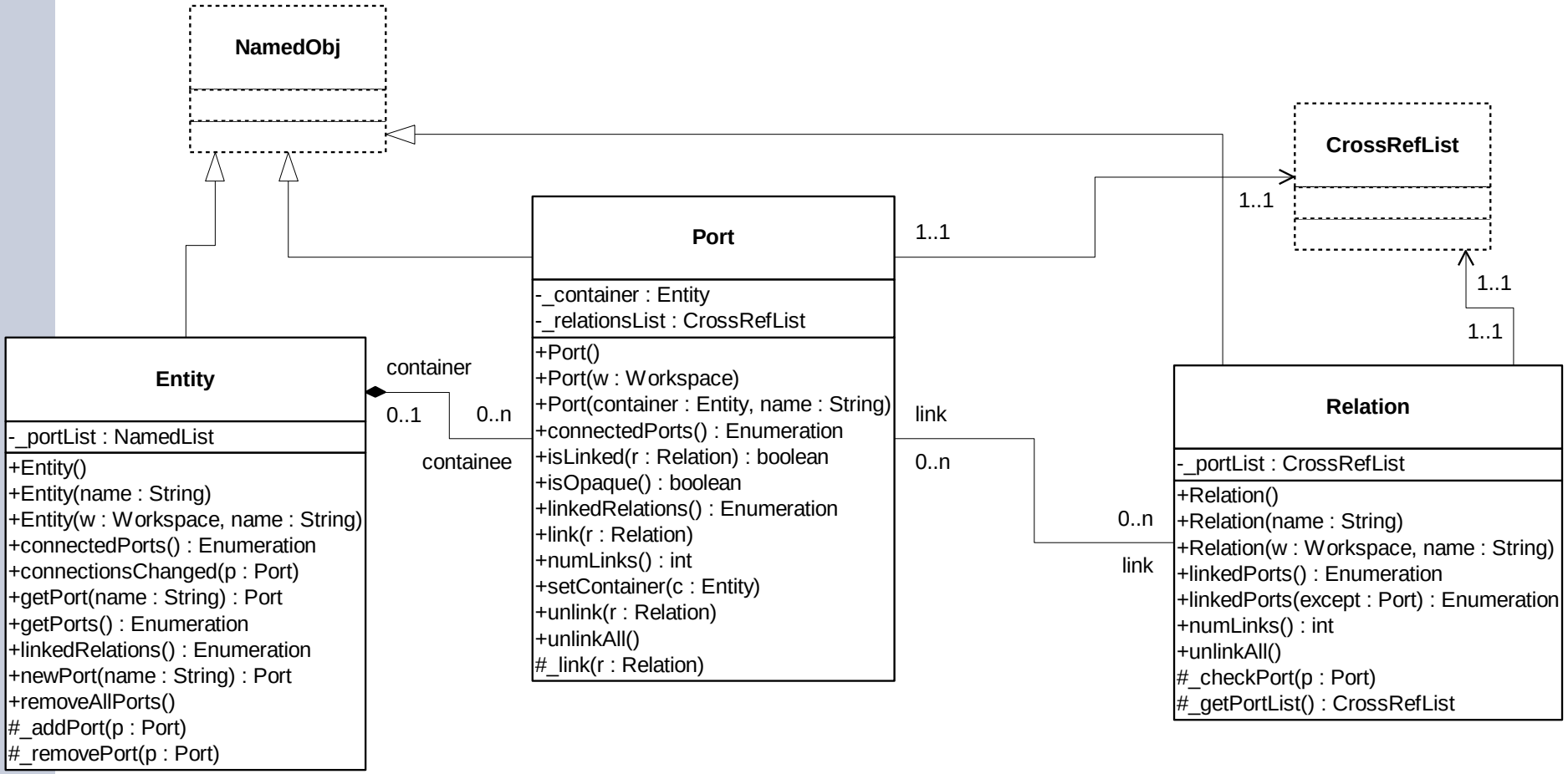
```
<entity name="tm3" class="ptolemy.actor.TypedCompositeActor">
  <property name="Continuous Director" class="ptolemy.domains.continuous.kernel.ContinuousDirector">
  <entity name="TaxiMeter" class="ptolemy.actor.TypedCompositeActor">
    <property name="ParameterSet" class="ptolemy.actor.parameters.ParameterSet">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

Hierarchical composition



* Edward Lee, UC Berkeley

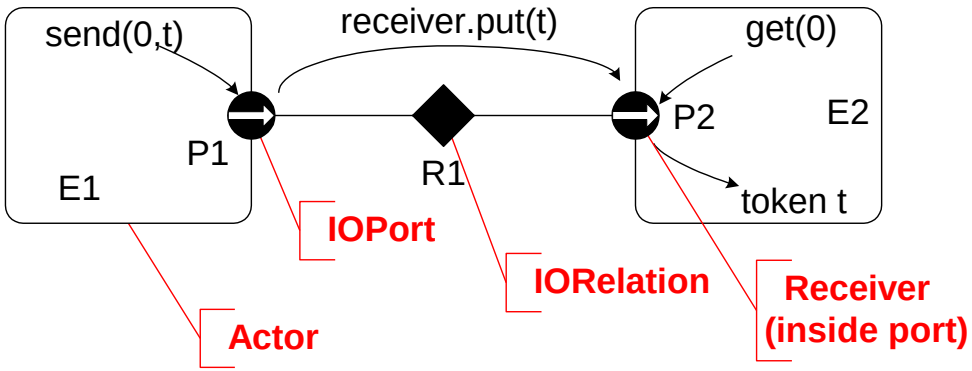
Kernel classes



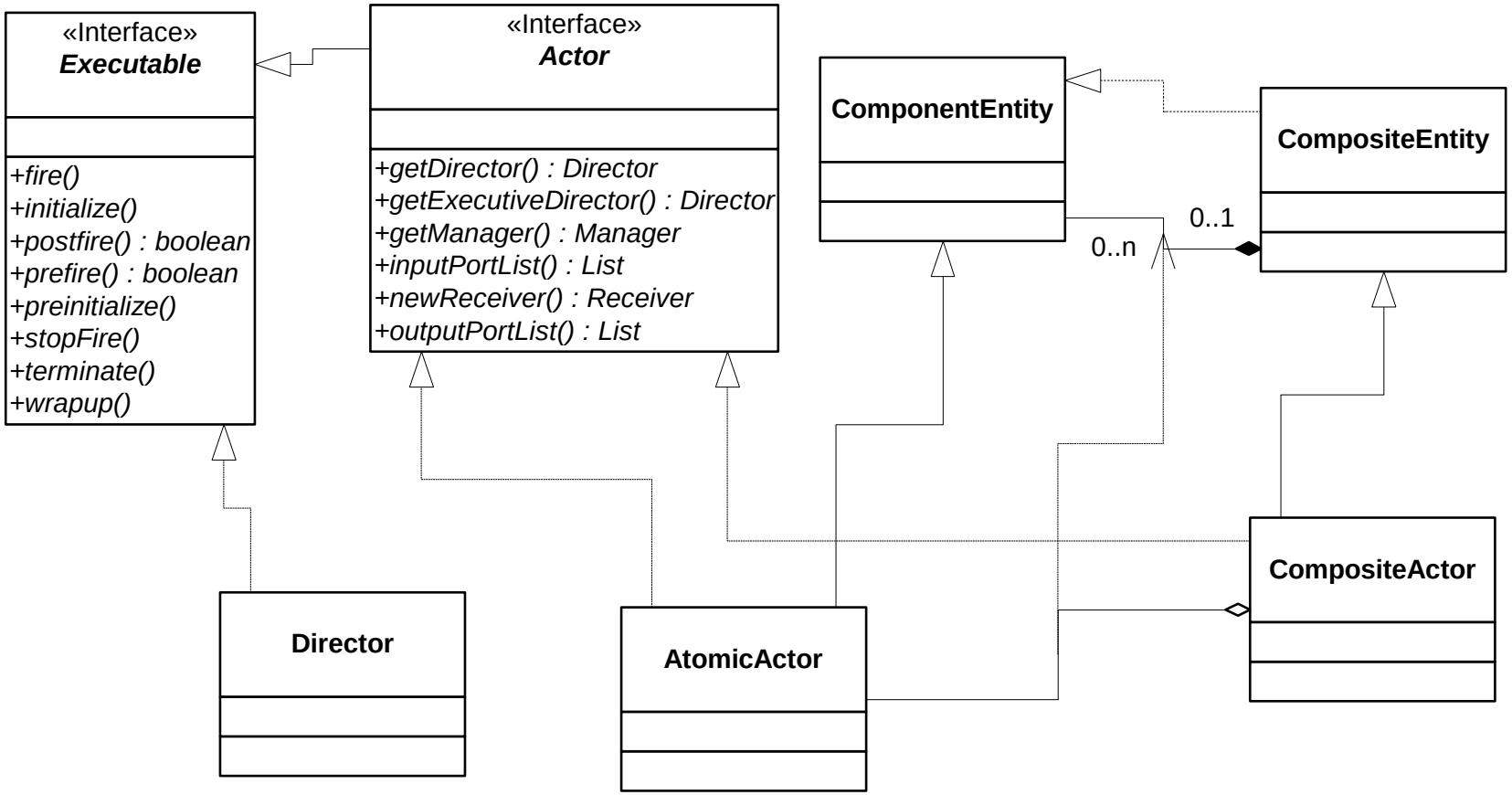
Actor package: producer/consumer components

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

Basic Transport:



Object Model for Executable Components

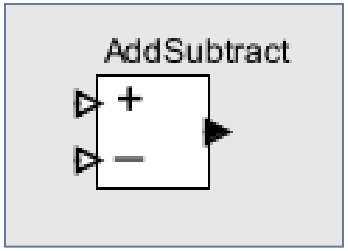


Actor execution interface

- 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



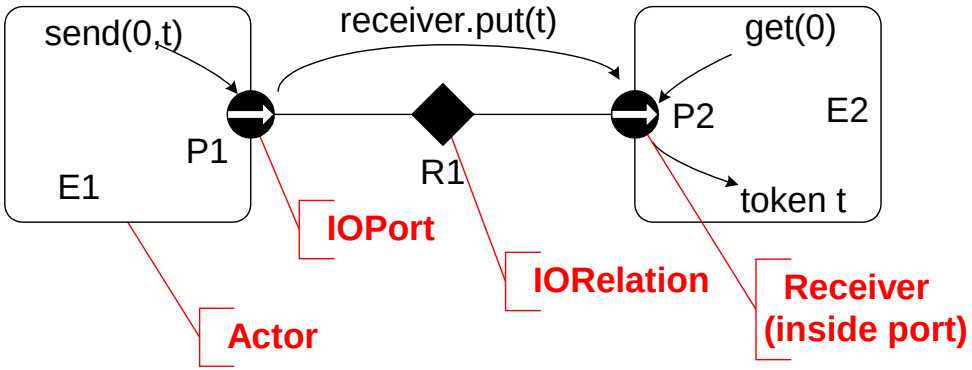
- Behavioral polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a time-triggered model, add when the clock ticks
  - In discrete-event, add when any connected input has data, and add in zero time
  - In process networks, execute an infinite loop in a thread that blocks when 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

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?

* Edward Lee, UC Berkeley

Actor View of Producer/Consumer Components

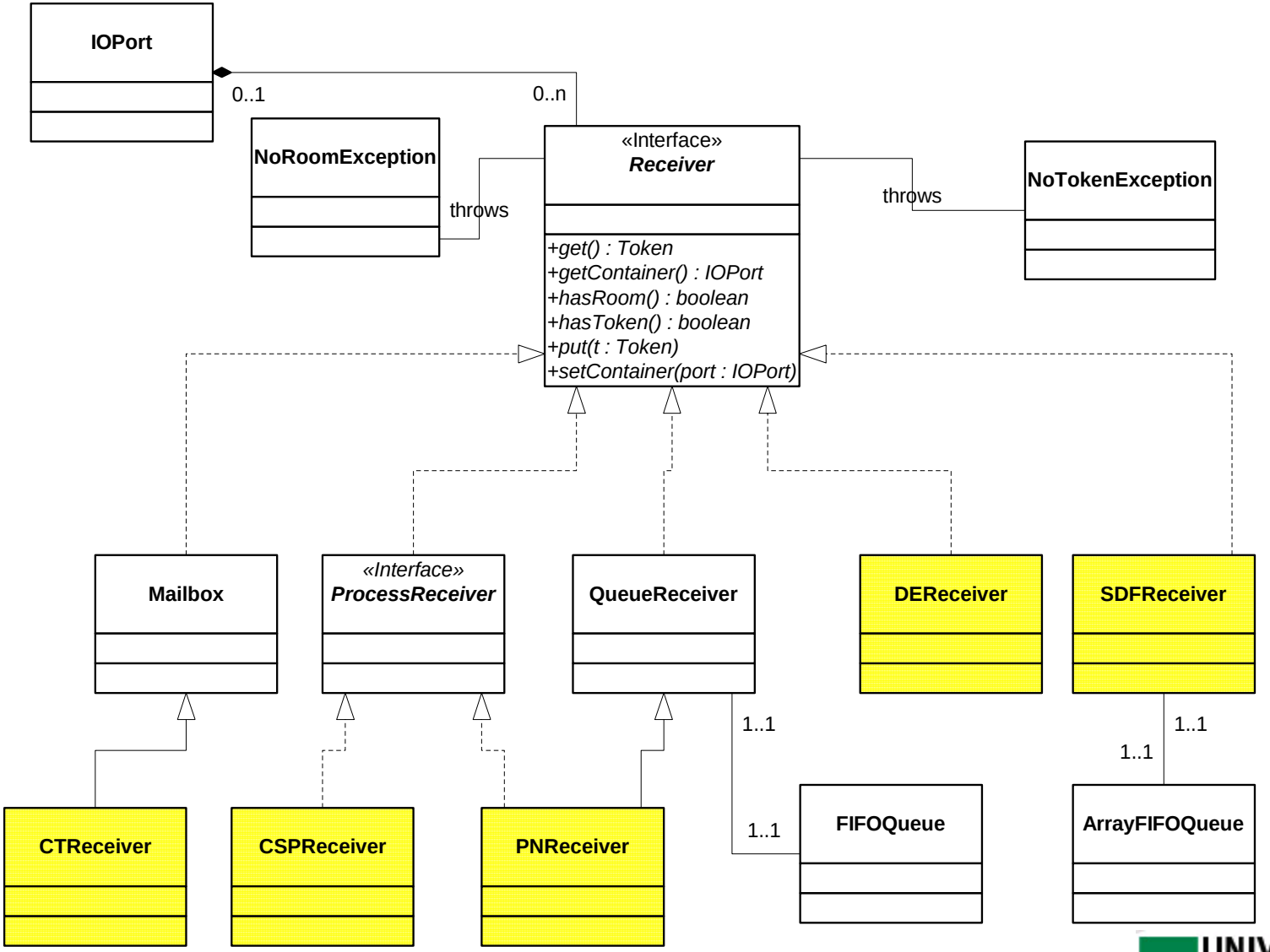
Basic Transport:



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.

* Edward Lee, UC Berkeley

Object Model for Communication Infrastructure

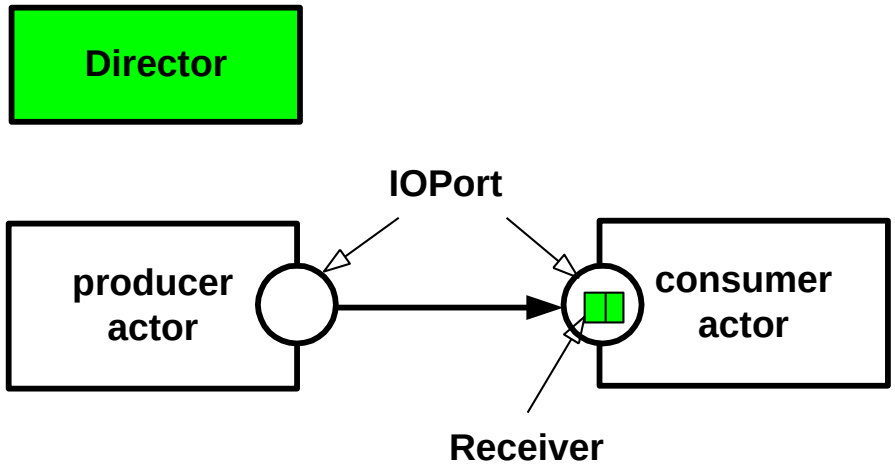


Object-Oriented Approach to Achieving Behavioral Polymorphism

«Interface» ***Receiver***
+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

Model of Computation

- Rules for actor execution
  - Static scheduling
  - Determined at runtime
  - Number of iterations
- Semantics of data transfer
- Rules for actor behavior
  - How tokens are consumed and produced in every iteration
- Implemented by the *Director* attribute

Discrete Event Director (DE)

- Has the notion of *model time*
- Maintains an event queue, where events are stored in increasing timestamp order
- At each iteration processes the events with the smallest timestamp in the queue (by iterating their target actors)
- Event are introduced in the queue by
  - An actor calling the *fireAt* method of the director:

    The calling actor will be the target and the event timestamp may be in the future
  - A token received in an input port: the *put* method of the DE receiver inserts an event having as target the container of the input port and as timestamp the current model time

Dataflow

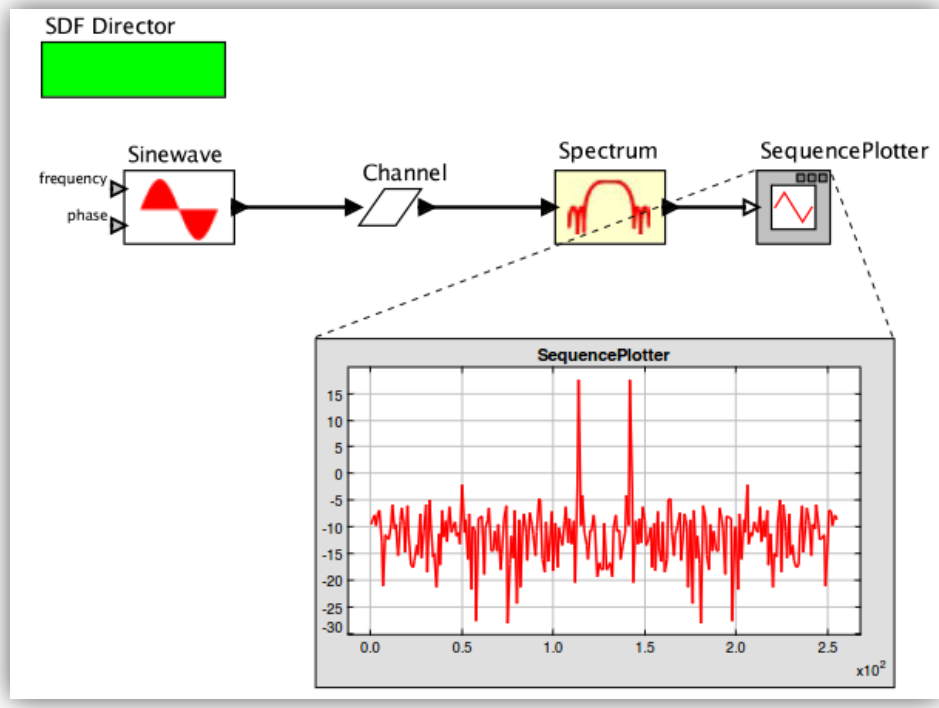
- Used for **pipe-and-filter** models
- Actors are fired when their required inputs become available
- Synchronous dataflow (SDF)
  - Data rates statically specified
  - Static schedule of actor execution

- Dynamic dataflow (DDF)
  - Constraints on consumption and production of tokens
  - No static schedule

SDF Example

- The Spectrum actor requires 256 tokens to fire

All other actors require one token each.



SDF issues

- Deadlock
- Consistency of data rates
- Scheduling