Heterogeneous Modeling and Simulation in Ptolemy

Yanwar Asrigo

17 March 2008

Acknowledgement

- This presentation is based on the following paper:

J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs and Y. Xiong,

"Taming Heterogeneity-the Ptolemy Approach," Proceedings of the IEEE, 91(2), January, 2003.

Agenda

- Motivation: Heterogeneous Modeling and Simulation
- Ptolemy Approach
- Ptolemy Design
- Example

Modeling Heterogeneous System

- Example: Embedded System

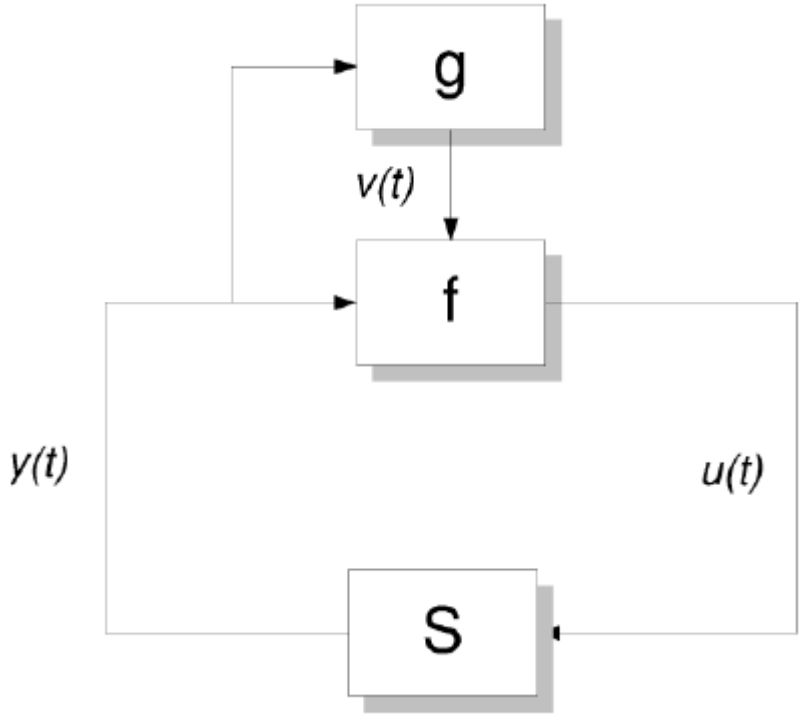
  (include subsytems such as: mechanical, hydraulic, analog, digital hardware, software, control logic, etc)

- Each model typically represent only one aspect of the entire system.

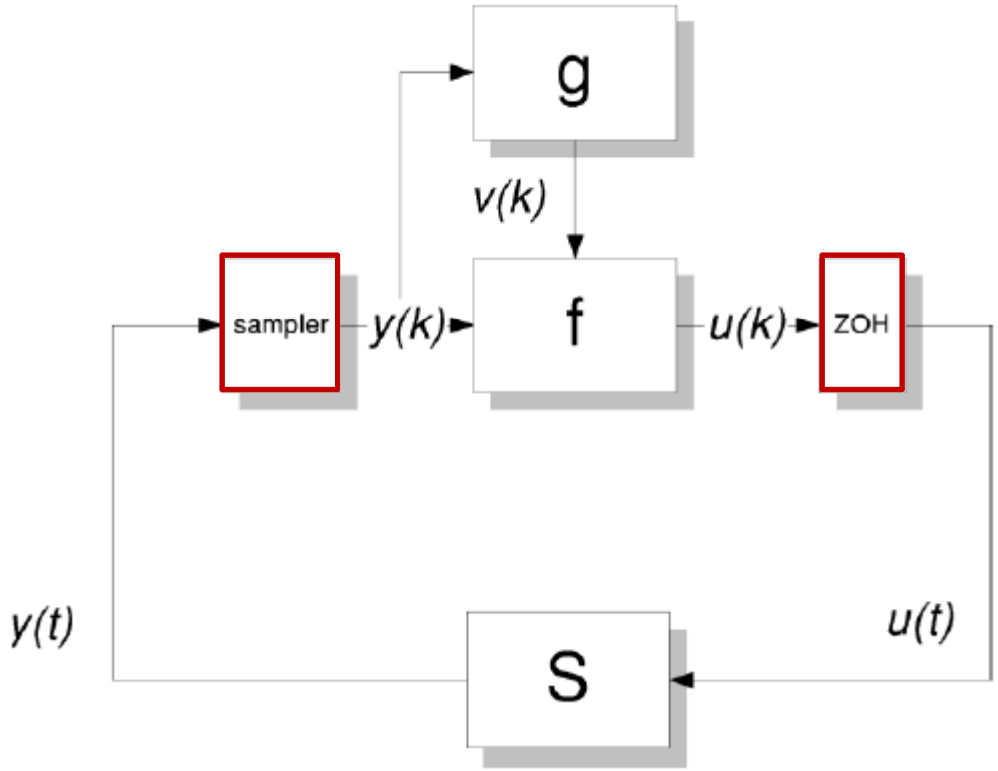
- How to evaluate the behavior of the whole system?

Heterogeneous Composition

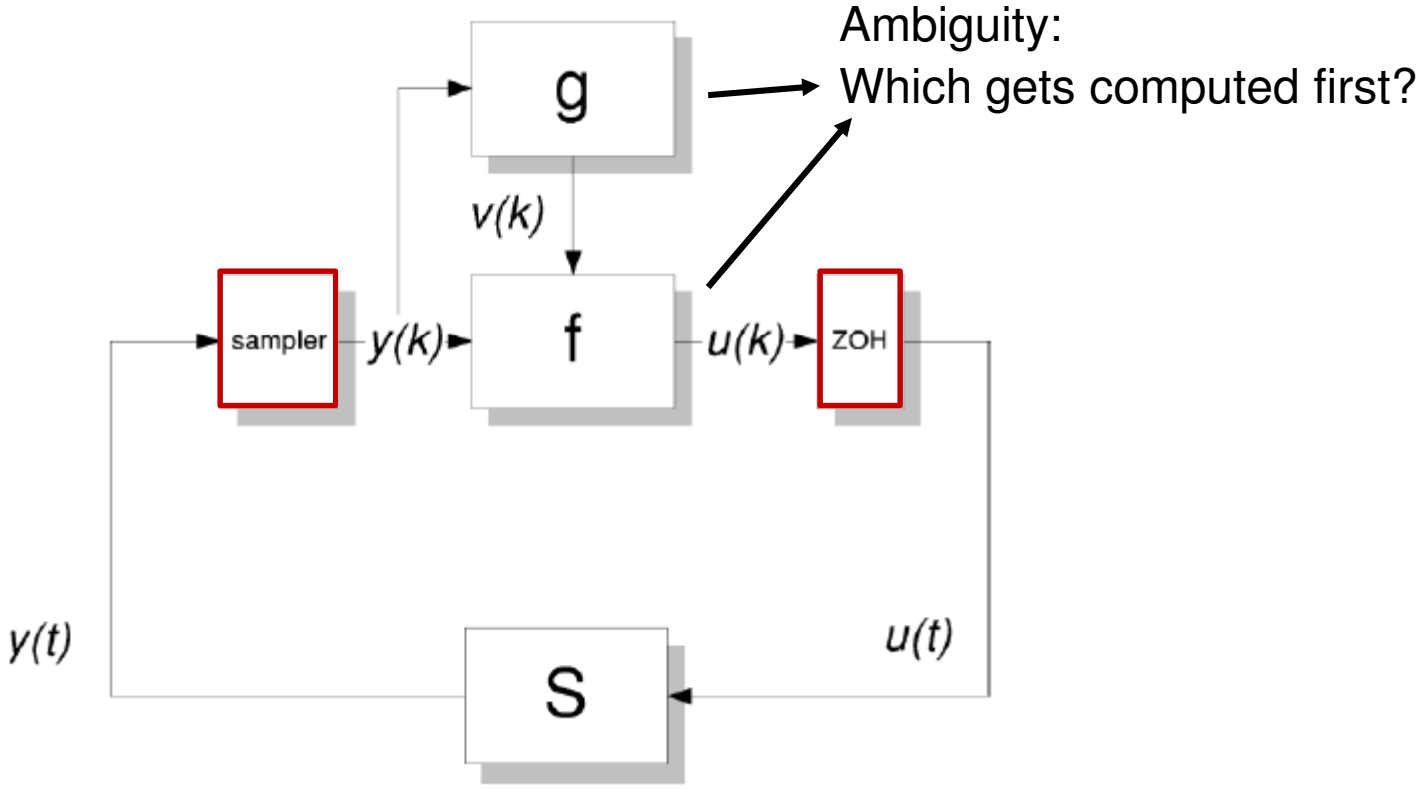
Example:
A model of Continuous Time (CT) System



Example:
An Amorphous Model of a Heterogeneous System



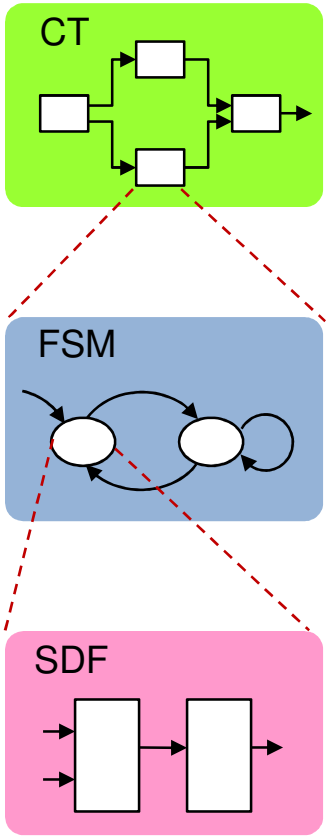
Example:
An Amorphous Model of a Heterogeneous System



Agenda

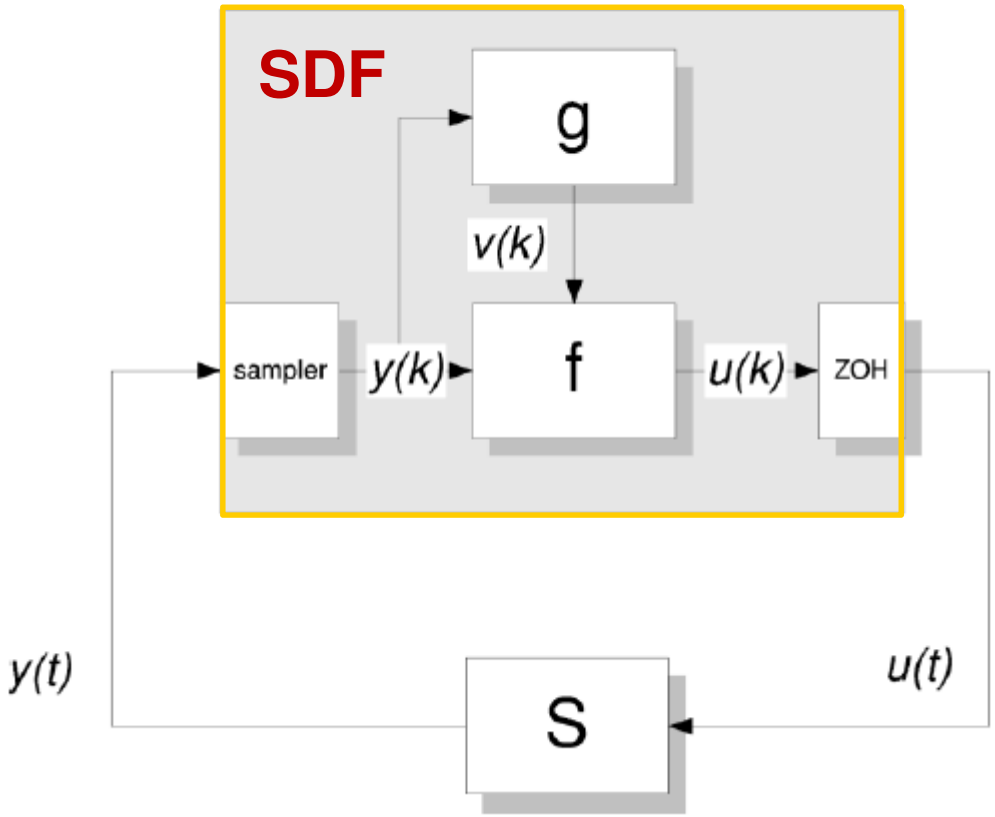
- Motivation: Heterogeneous Modeling and Simulation
- Ptolemy Approach
- Ptolemy Design
- Example

Ptolemy Approach:
Hierarchically Heterogeneous

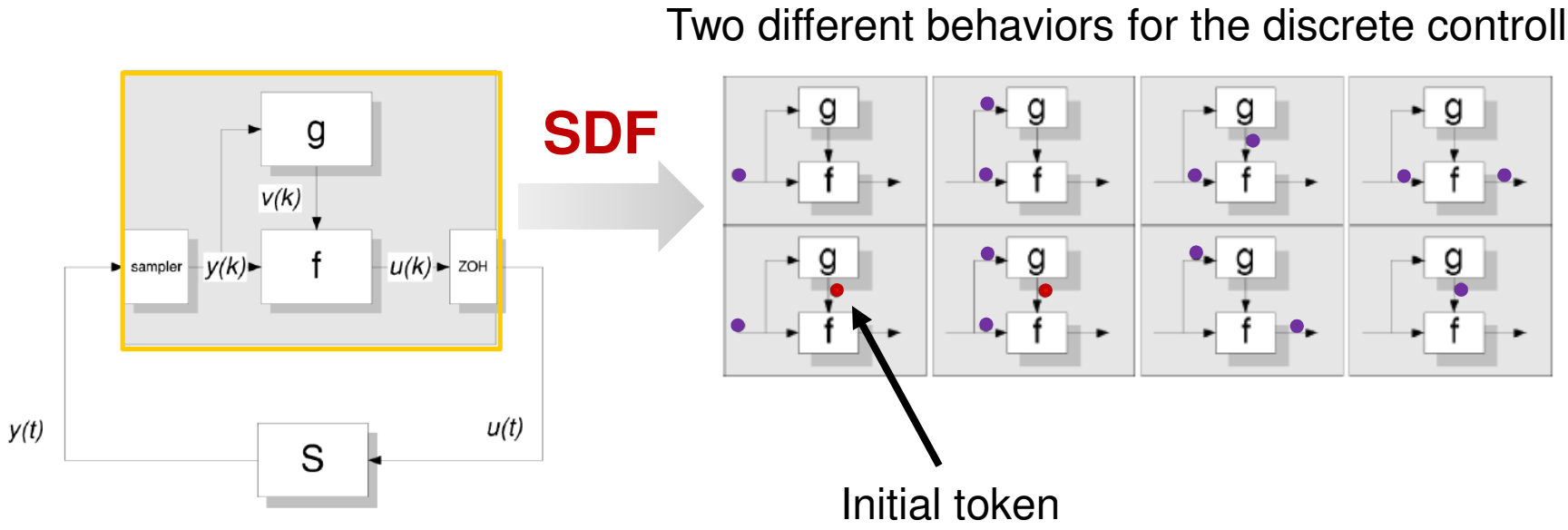


- Divide a complex model into a tree of nested submodels
- Each level is composed of a network of interacting components
- Network at each level is **locally homogenous**, while allowing different interaction mechanisms to be specified at different levels in the hierarchy

Example:
A Hierarchically Heterogeneous Version



Example:
A Hierarchically Heterogeneous Version

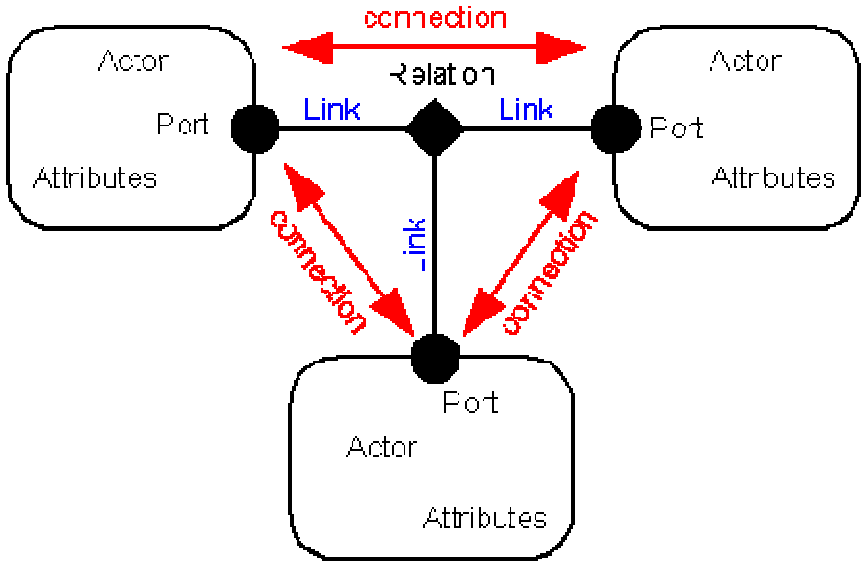


Agenda

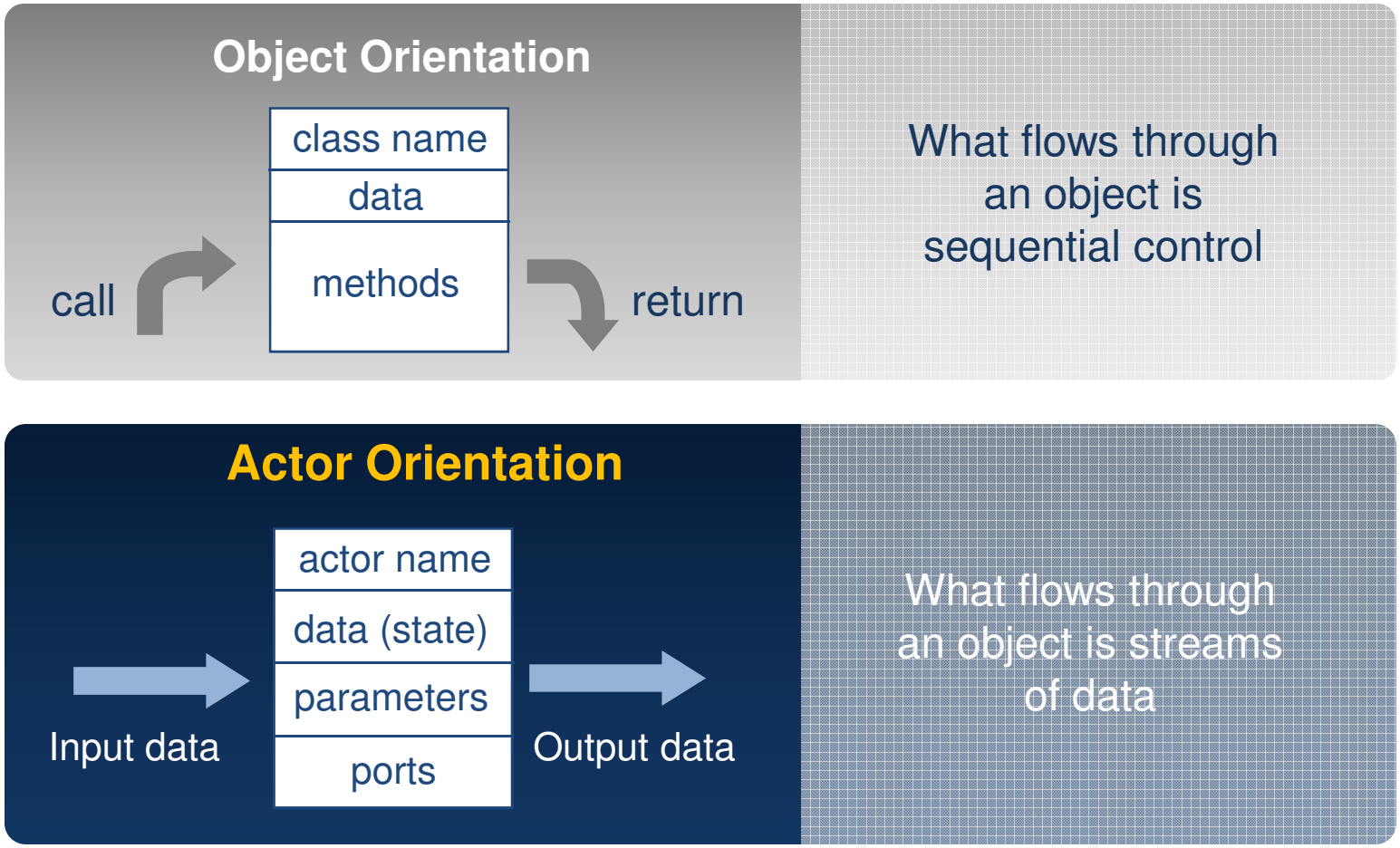
- Motivation: Heterogeneous Modeling and Simulation
- Ptolemy Approach
- Ptolemy Design
  - Actor Oriented Model and Design
  - Actor Execution
  - Domains or Model of Computation
  - Actor polymorphism
- Example

Actor Oriented Modeling and Design

- **Actor** as the basic building block of a system
- Actors are concurrent components that communicate through **ports**.
- **Relations** define the interconnect between these ports



Actor Oriented Design



Actor Oriented Design

- Actor-oriented view of a system decouples the **transmission of data** from the **transfer of control**.

- **Model of Computation (MoC)**, rather than the individual actors themselves, defines the details of **scheduling** and **communication**, and whether and how these are related.

Consequences of Actor Orientation

- Actors (components) are much more reusable

  Actors describe abstract functionality that can be run in a large number of different ways

- A composition of actors governed by some MoC can more easily be analyzed and understood

  No way that different actors might have made different assumptions about the way control flow and communication are handled

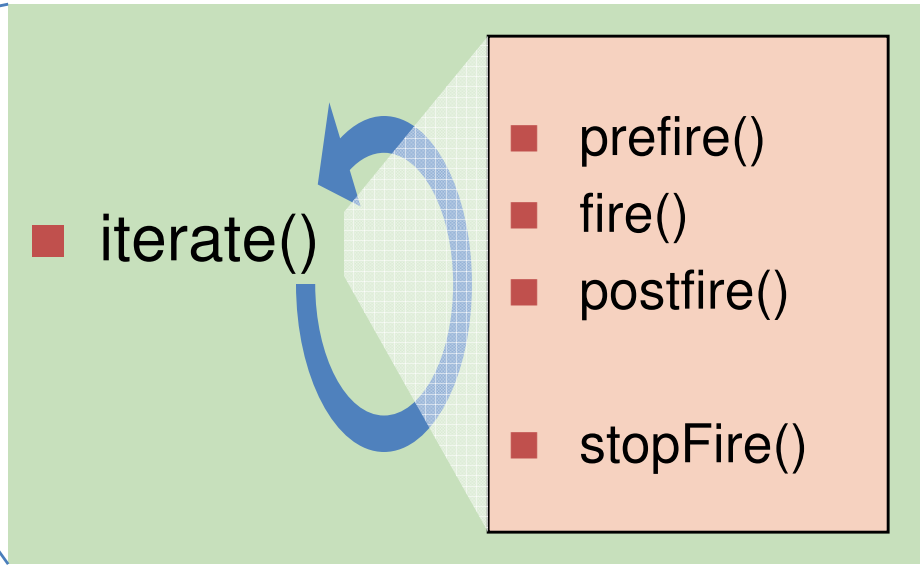
Model Structure and Execution

- Actors (**atomic** and **composite**) are executable.
- An actor is **composite** if it contains other actors.
- The execution of a composite actor is a controlled composition of the execution of the actors it contains.
- This compositionality of execution is the key to managing heterogeneity hierarchically.

Actor Execution

- Execution Phases
  - Initialization
  - Execution
  - Finalization

Despite the computation occurs during the fire phase, the state of the actor is not updated until post-fire



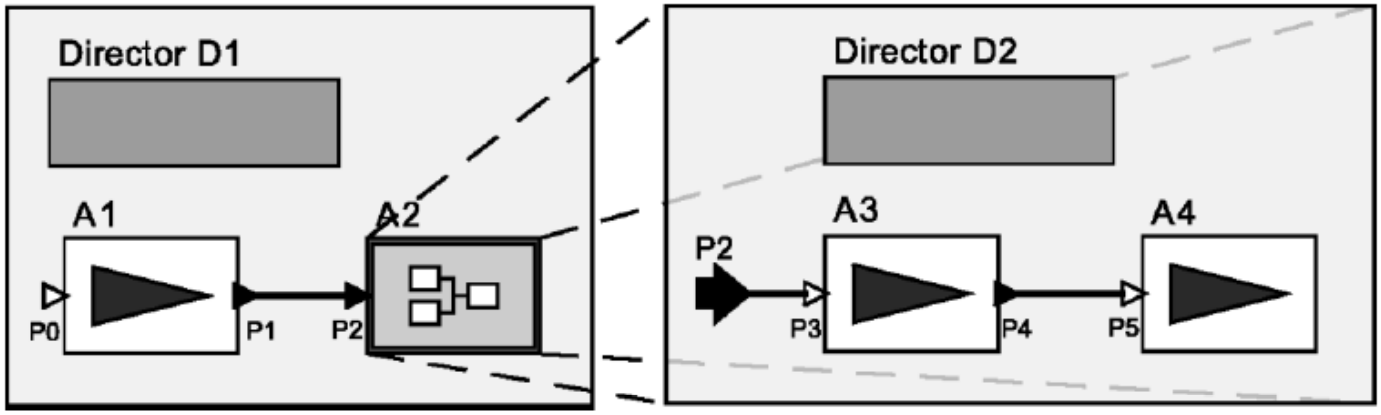
The MoC of a composite actor determines how the iteration of one actor is related to the iterations of other actors in the same composite

Domains or Model of Computation (MoC)

- Domains provide semantic models for component interactions. Domain is realized by:
  - **Director**: governs the execution of a composite entity
  - **Receiver**: implements the communication semantics

- The director is responsible for creating the receivers
- Actors, when resident in a domain, acquire domain-specific receivers.
- This separation of computation and communication allow actors (called **domain-polymorphic actors**) to be reused in different domains.

Domains



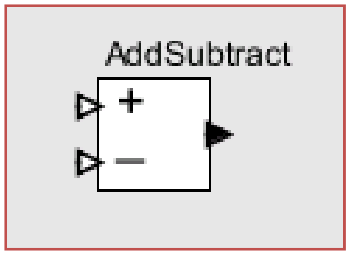
- Director D1 Controls the execution order of actors A1 and A2
- Director D2 controls the execution of A3 and A4 whenever A2 is executed.

Realized Domains / MoC

- Continuous Time (CT)
- Discrete Event (DE)
- Synchronous Data Flow (SDF)
- Finite State Machine (FSM)
- Communicating Sequential Processes (CSP)
- Process Network (PN)
- …

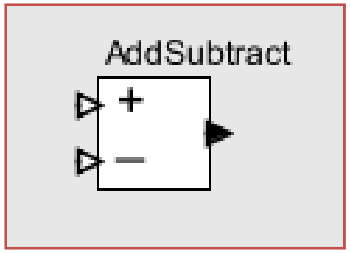
Polymorphic Components Example

- 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
  - …



Polymorphic Components Example

- 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
  - …



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?

Domain Polymorphism

- Ideally:
  - The receiver and director decouple the idiosyncrasies of interaction in a domain from the requirements of the actors embedded in it. GREAT!
  - So, can any actor (including model in any domain) be embedded into any other model? NO!!

Domain Polymorphism

- Issue: Behavior of receivers and directors are not necessarily the same.
- Example:
  - fire() is used to start actor execution
  - In SDF domain: actor is fired if there is a token in its receiver
  - In DE domain: director doesn't make such guarantee.
  - SDF Actor in DE Domain:

    fire() → Exception is raised!

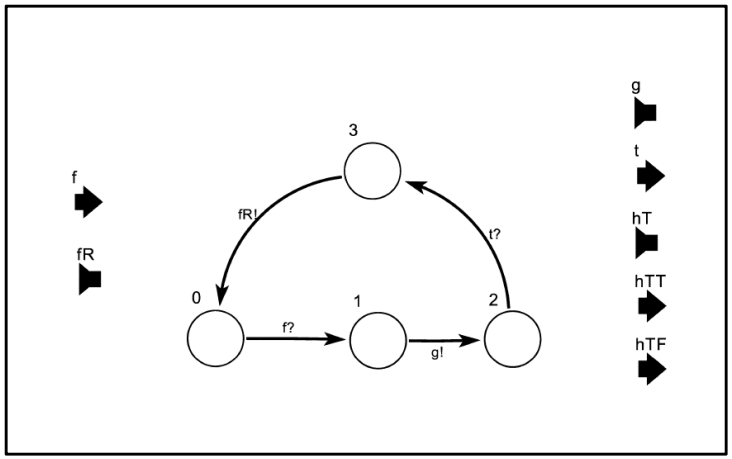
Domain Polymorphism

- Solution: Interface automata
  - **Domain automata**: describe the behavior of receiver + director
  - **Actor automata**: describe the behavior of the actor
- **Compatibility test**:

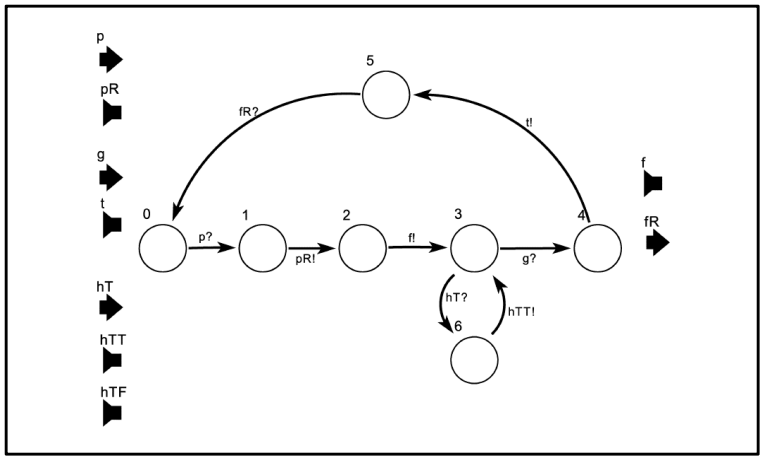
  Composition of domain automata with actor automata is not empty.

Compatibility Test

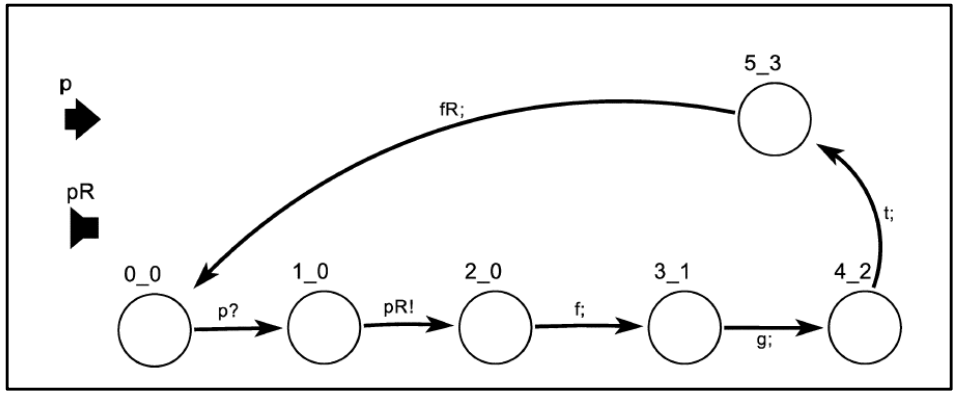
SDF Actor Interface Automaton



SDF Domain Interface Automaton



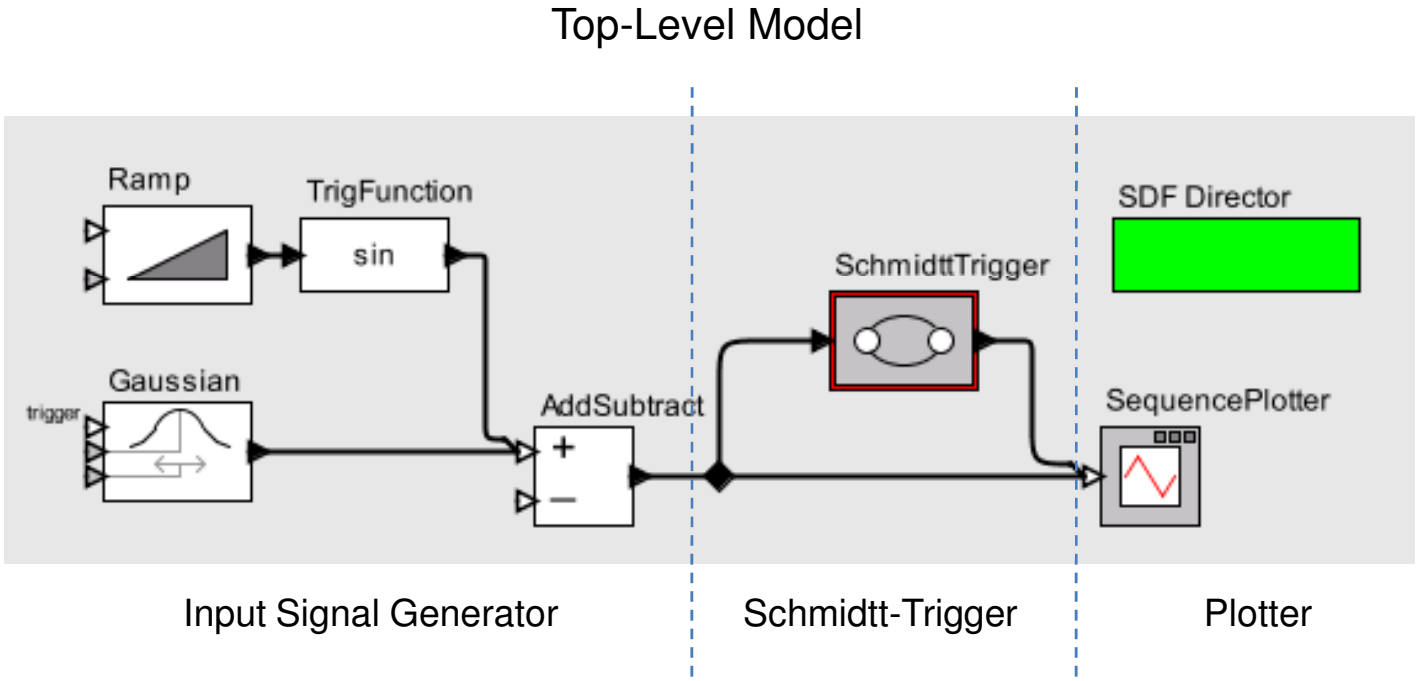
Composed Automaton



Agenda

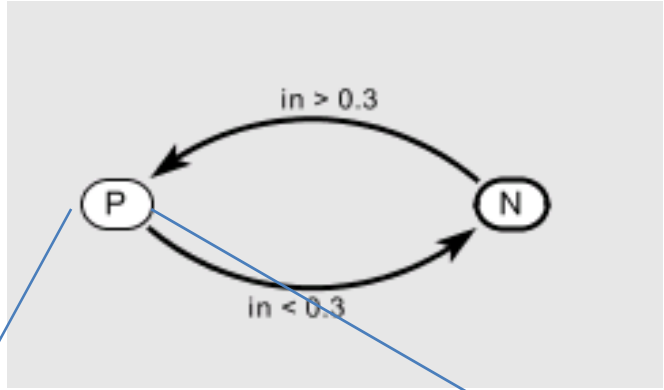
- Motivation: Heterogeneous Modeling and Simulation
- Ptolemy Approach
- Ptolemy Design
- Example

Example – Schmidt Trigger

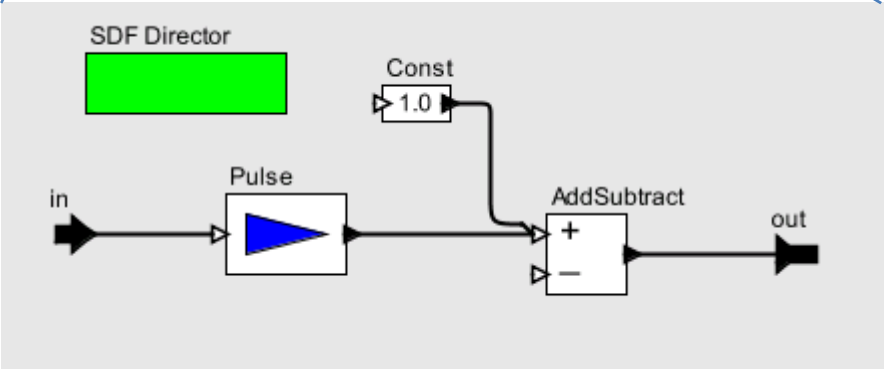


Example – Schmidt Trigger

The mode controller for SchmidttTrigger

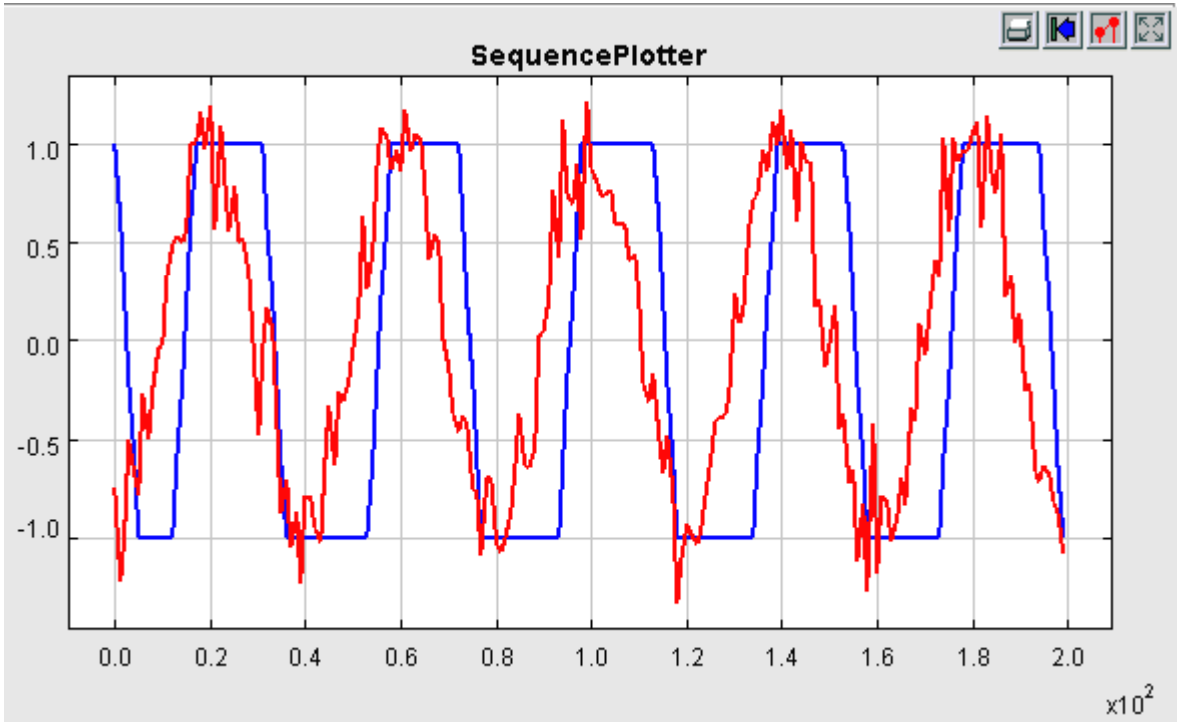


RefinementP has Const set to 1.0
RefinementP has Const set to -1.0



Example – Schmidt Trigger

Sample Result of the Model



Thank You