Event Relation Graphs and Extensions in Ptolemy II

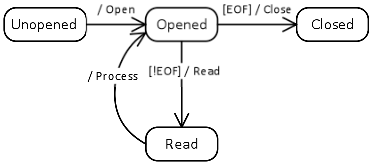
Thomas Huining Feng

EECS, UC Berkeley

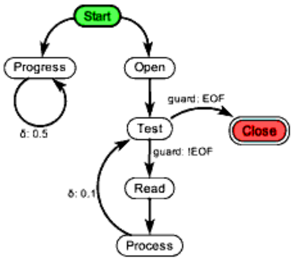
EE290N Class Presentation *May 8, 2009*

Background

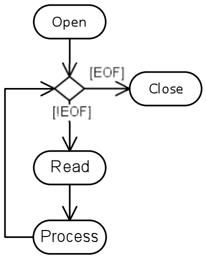
Finite State Machine



Event Graph



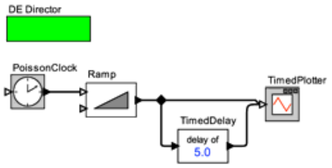
Activity Diagram



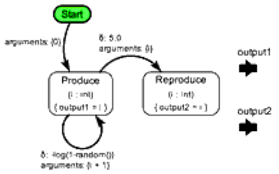
- Like activity diagram, nodes are in fact state transitions
- More expressive (equivalent to Petri net with inhibitor arcs and Turing machine)
- Model time and event queue (similar to DE)

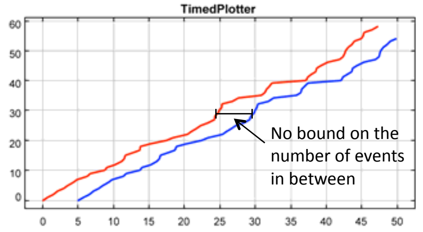
Background

Actor-Oriented Modeling



Event-Oriented Modeling



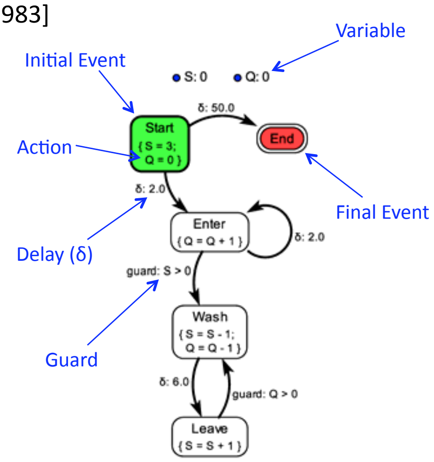


Different views of the system.
Both require unbounded event queue in general.

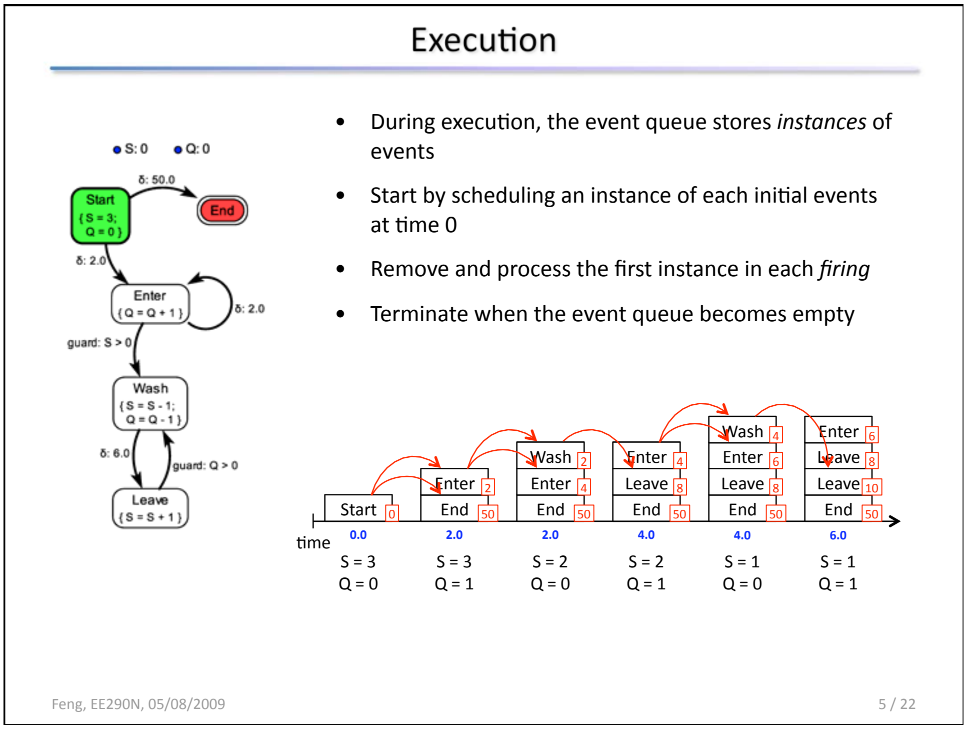
Syntax

- Based on event graphs [Schruben 1983]
- Visual representation
  - Nodes are events
  - Edges are scheduling relations
- Compare to FSM
  - Actions on events
  - Can schedule multiple events
  - Timed

CarWash: single queue multiple servers



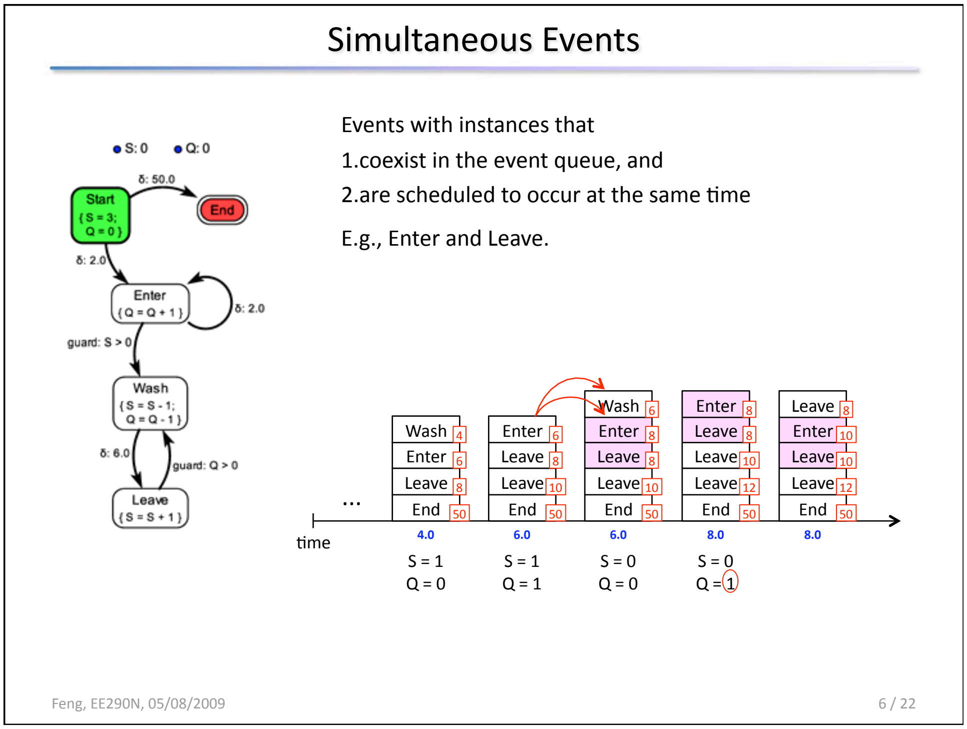
The CarWash model



Execution

- During execution, the event queue stores *instances* of events
- Start by scheduling an instance of each initial events at time 0
- Remove and process the first instance in each *firing*
- Terminate when the event queue becomes empty

0.0	2.0	2.0	4.0	4.0	6.0
				Wash 4	Enter 6
		Wash 2	Enter 4	Enter 6	Leave 8
	Enter 2	Enter 4	Leave 8	Leave 8	Leave 10
Start 0	End 50	End 50	End 50	End 50	End 50
S = 3 Q = 0	S = 3 Q = 1	S = 2 Q = 0	S = 2 Q = 1	S = 1 Q = 0	S = 1 Q = 1



Simultaneous Events

Events with instances that

1.coexist in the event queue, and

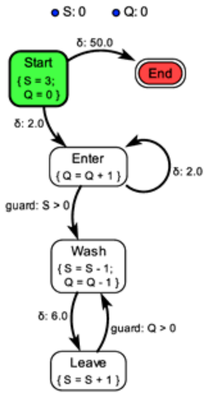
2.are scheduled to occur at the same time

E.g., Enter and Leave.

4.0	6.0	6.0	8.0	8.0
		Wash 6	Enter 8	Leave 8
Wash 4	Enter 6	Enter 8	Leave 8	Enter 10
Enter 6	Leave 8	Leave 8	Leave 10	Leave 10
Leave 8	Leave 10	Leave 10	Leave 12	Leave 12
End 50	End 50	End 50	End 50	End 50

S = 1
Q = 0
	S = 1
	Q = 1
		S = 0
		Q = 0
			S = 0
			Q = 1	

Simultaneous Events

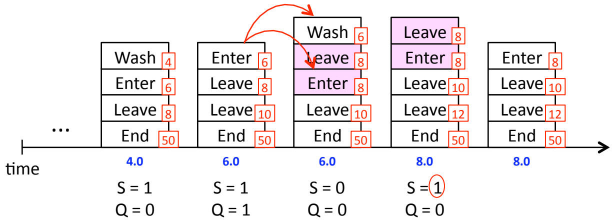


Events with instances that

1.coexist in the event queue, and

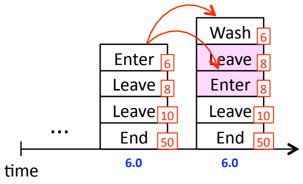
2.are scheduled to occur at the same time

E.g., Enter and Leave.

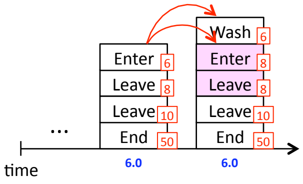


FIFO and LIFO Policies

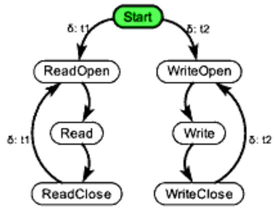
- With FIFO (First In First Out) policy



- With LIFO (Last In First Out) policy



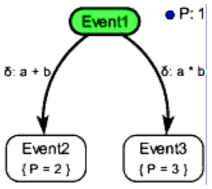
Rationale for FIFO and LIFO



1. With FIFO, when $x*t1 = y*t2 \wedge t1 > t2$
   ReadOpen ⟶ WriteOpen ⟶ Read ⟶ Write ⟶ ReadClose ⟶ WriteClose

2. With LIFO, always
   (ReadOpen ⟶ Read ⟶ ReadClose), (WriteOpen ⟶ Write ⟶ WriteClose)

Suggestions for This Case?

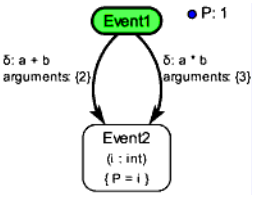


Suppose a and b are defined elsewhere.

In case a = b = 2, we have simultaneous instances of Event2 and Event3.

- Do not allow such design
  How to identify them?
- Leave undefined
  Ambiguous semantics
- Throw exception
  Maybe surprise the user
- Randomly pick one
  Unexpected nondeterministic behavior
- Use location of the events
  Forbid rerendering the graph
- Use names of the events
  Partially solves the problem

Suggestions for This Case?



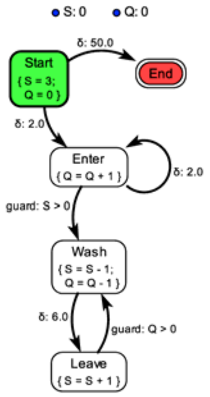
Suppose a and b are defined elsewhere.

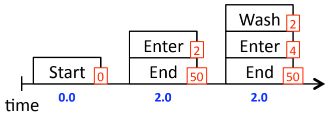
In case a = b = 2, we have simultaneous instances of Event2 itself.

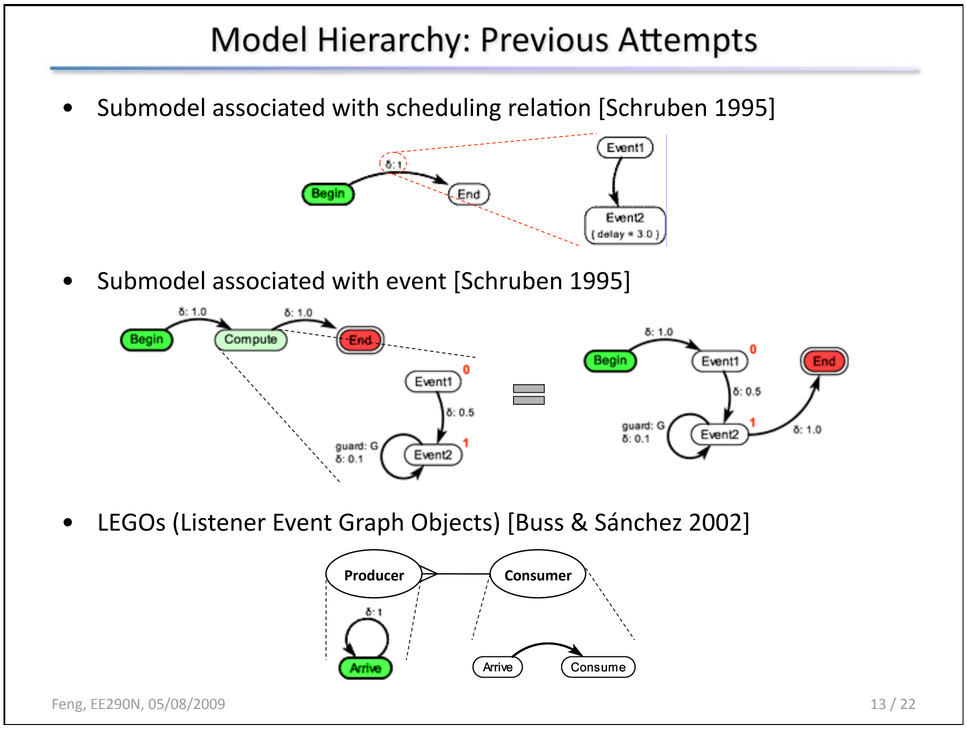
- Do not allow such design
  How to identify them?
- Leave undefined
  Ambiguous semantics
- Throw exception
  Maybe surprise the user
- Randomly pick one
  Unexpected nondeterministic behavior
- Use location of the events
  Forbid rerendering the graph
- Use names of the events
  Partially solves the problem
- Now we really need some hidden info
  E.g., names of the scheduling relations

Execution Algorithm

1. Initialize *E* to contain all initial events
2. While *E* is not empty
   a. Remove the top instance *t* from *E*
   b. Execute *t*’s actions
   c. Terminate if *t* is a final event
   d. Schedule events in *E* in the order of
      1. Time stamp
      2. FIFO or LIFO policy
      3. Event name
      4. Scheduling relation name





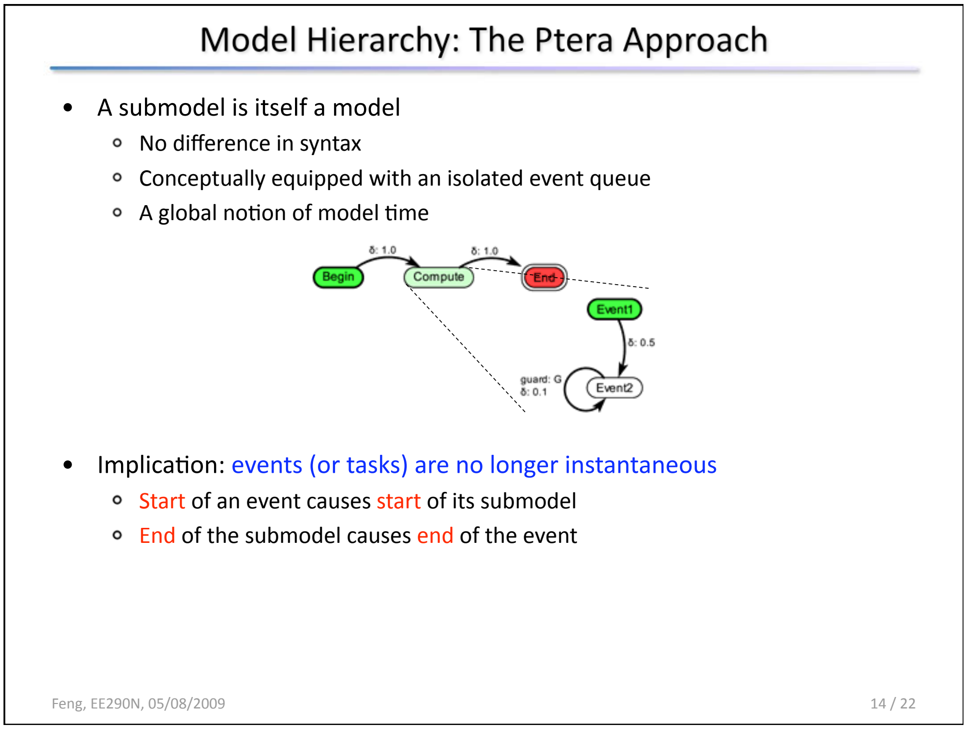


Model Hierarchy: Previous Attempts

- Submodel associated with scheduling relation [Schruben 1995]

- Submodel associated with event [Schruben 1995]

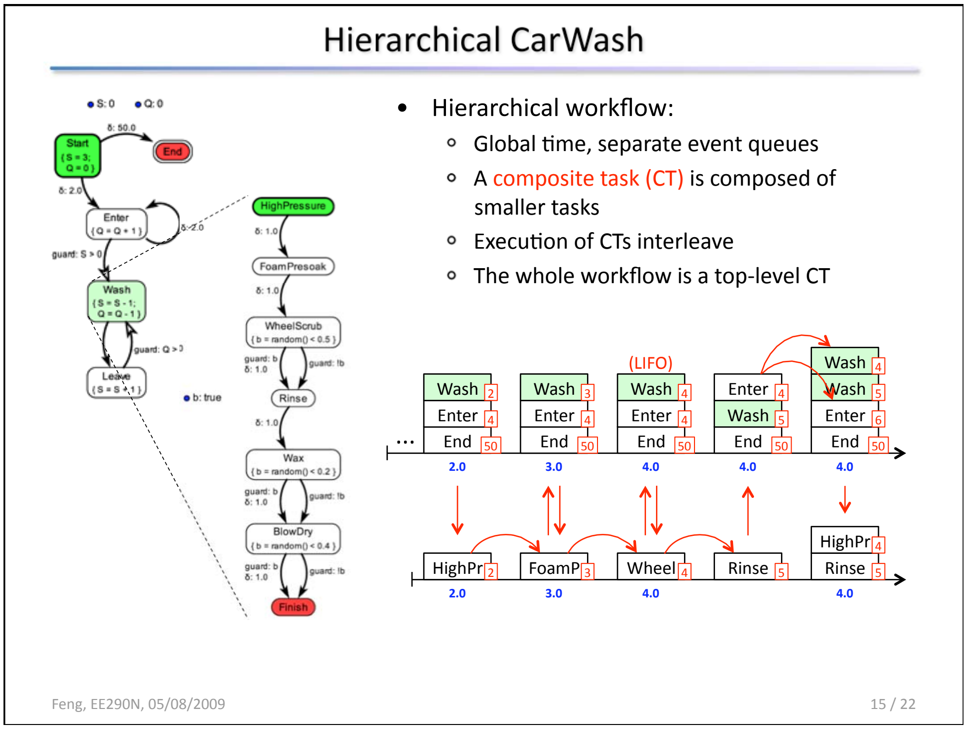
- LEGOs (Listener Event Graph Objects) [Buss & Sánchez 2002]



Model Hierarchy: The Ptera Approach

- A submodel is itself a model
  - No difference in syntax
  - Conceptually equipped with an isolated event queue
  - A global notion of model time

- Implication: events (or tasks) are no longer instantaneous
  - Start of an event causes start of its submodel
  - End of the submodel causes end of the event

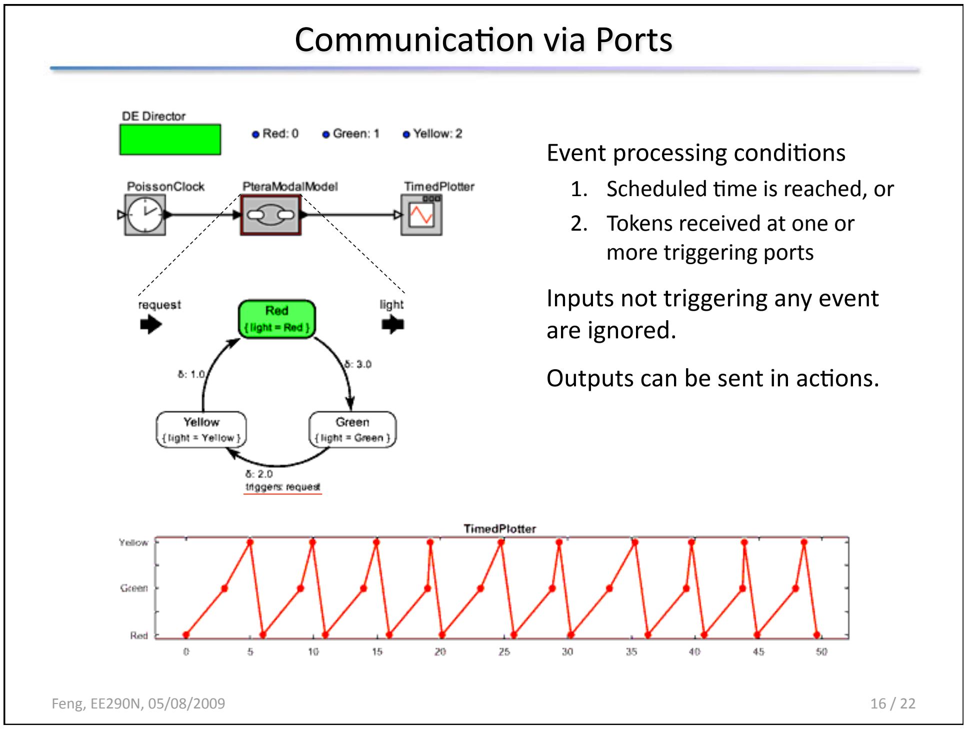


Hierarchical CarWash

- Hierarchical workflow:
  - Global time, separate event queues
  - A composite task (CT) is composed of smaller tasks
  - Execution of CTs interleave
  - The whole workflow is a top-level CT

Feng, EE290N, 05/08/2009

15 / 22



Communication via Ports

Event processing conditions

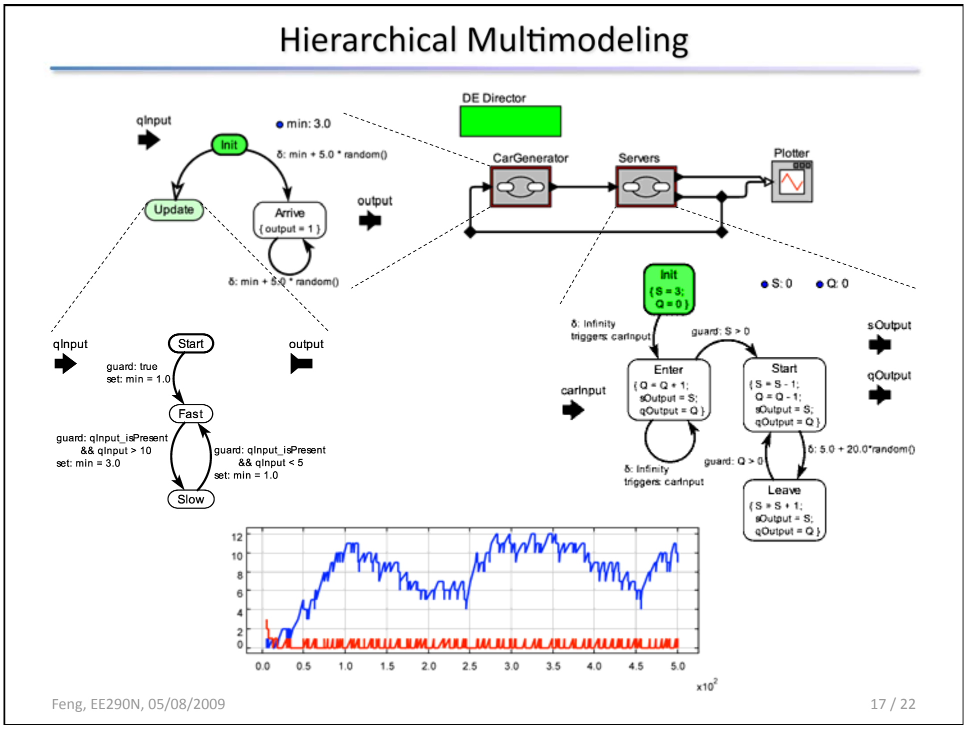
1. Scheduled time is reached, or
2. Tokens received at one or more triggering ports

Inputs not triggering any event are ignored.

Outputs can be sent in actions.

Feng, EE290N, 05/08/2009

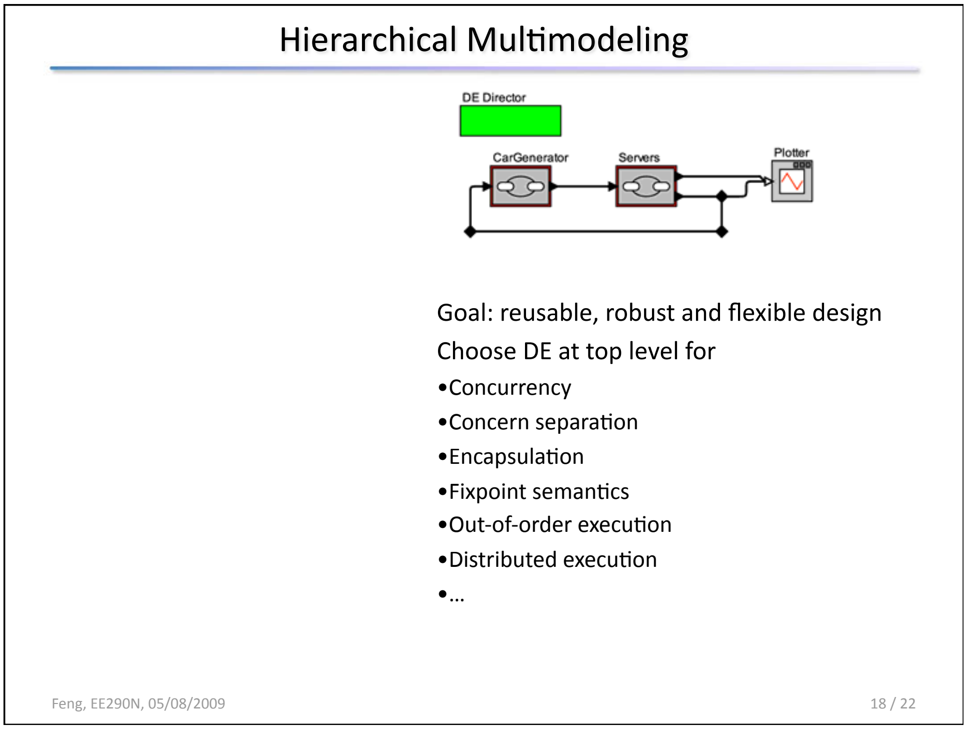
16 / 22



Hierarchical Multimodeling

Feng, EE290N, 05/08/2009

17 / 22



Hierarchical Multimodeling

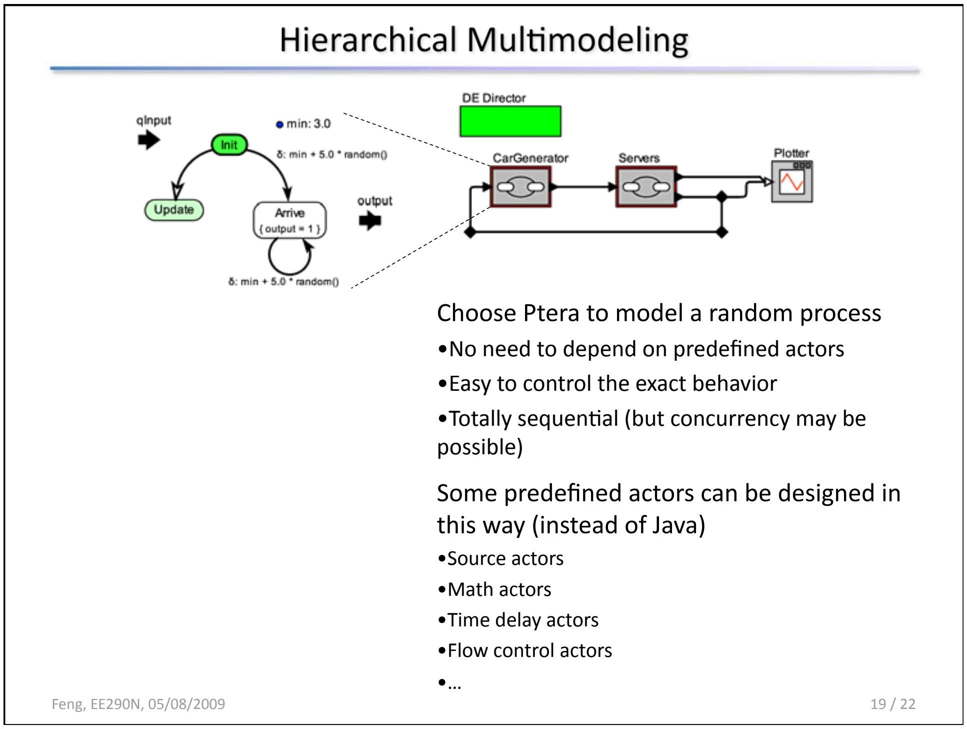
Goal: reusable, robust and flexible design

Choose DE at top level for

- Concurrency
- Concern separation
- Encapsulation
- Fixpoint semantics
- Out-of-order execution
- Distributed execution
- …

Feng, EE290N, 05/08/2009

18 / 22



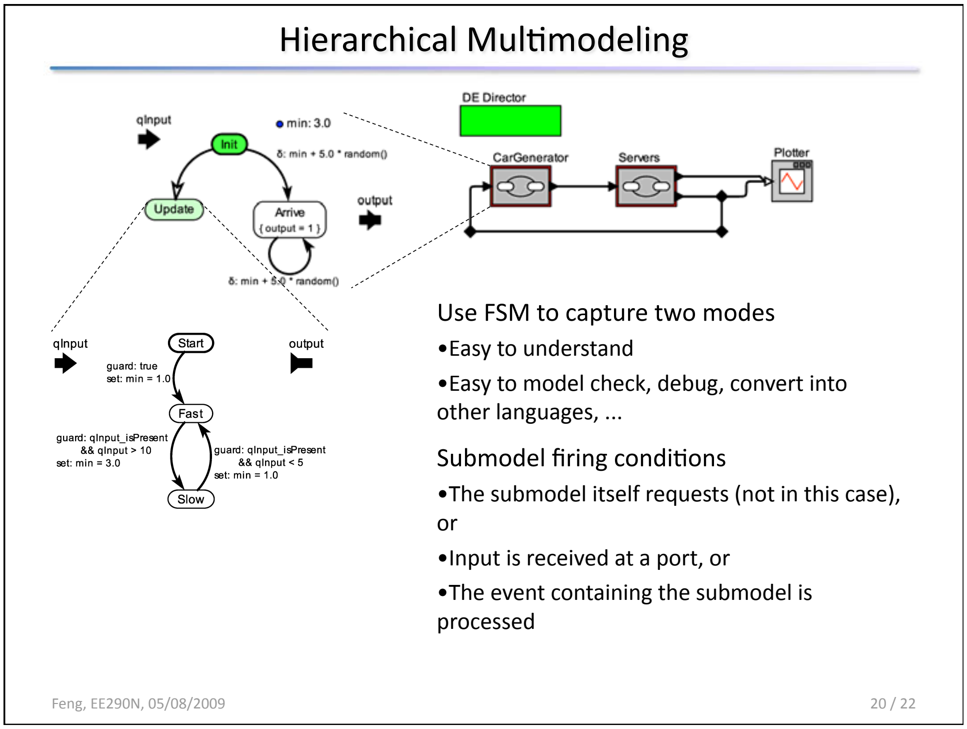
Hierarchical Multimodeling

Choose Ptera to model a random process

- No need to depend on predefined actors
- Easy to control the exact behavior
- Totally sequential (but concurrency may be possible)

Some predefined actors can be designed in this way (instead of Java)

- Source actors
- Math actors
- Time delay actors
- Flow control actors
- …



Hierarchical Multimodeling

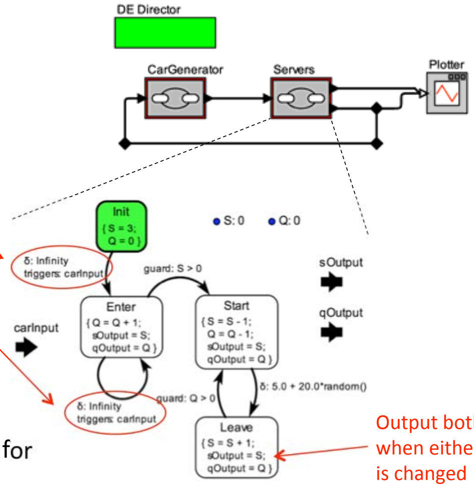
Use FSM to capture two modes

- Easy to understand
- Easy to model check, debug, convert into other languages, ...

Submodel firing conditions

- The submodel itself requests (not in this case), or
- Input is received at a port, or
- The event containing the submodel is processed

Hierarchical Multimodeling



Indefinitely wait for change of car number.

Output both S and Q when either of them is changed

A Ptera model similar to the standalone version.

Listen to the input port for car arrivals.

Composition with other components is richer than the LEGOs approach.

Future Work

- Composition with other MoCs
  (Especially, Ptides and continuous time)
- Formal analysis
  (Bound of event queue, simultaneous events, termination condition, model categorization, …)
- Behavior-preserving concurrent and distributed execution
- Other application domains
  (Currently studied: statistical analysis, model transformation)
- Tool support
  (Debugging and testing, code generation)
- Design patterns
  (Currently studied: Input, Output, LoopForCount, ParallelTasks, SingleQueueMultipleServers)