Happy Birthday ROC!

Heterogeneous Actor Models

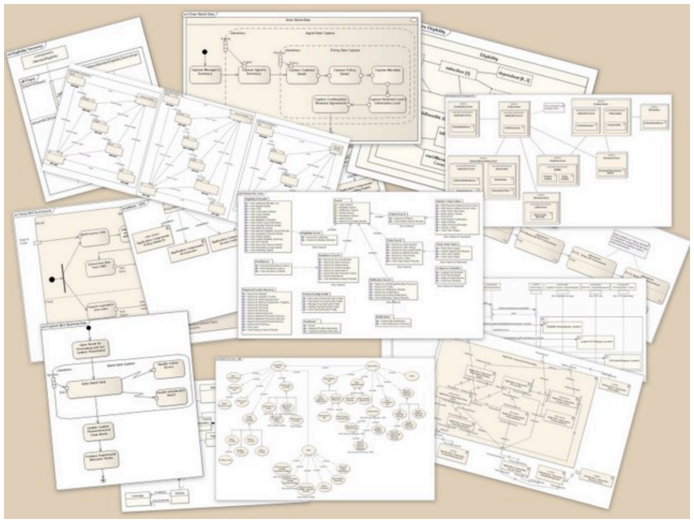
Edward A. Lee

Robert S. Pepper Distinguished Professor
EECS Department
UC Berkeley

Invited Roadmap Talk

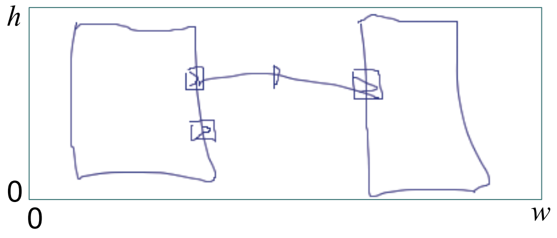
EMSOFT
Taipei, Taiwan, October 10, 2011

One way to build heterogeneous models: UML : Unified?



[Image from Wikipedia Commons. Author: Kishorekumar 62]

The Truly Unified Modeling Language *TUML*



A *model* in TUML is a function of the form

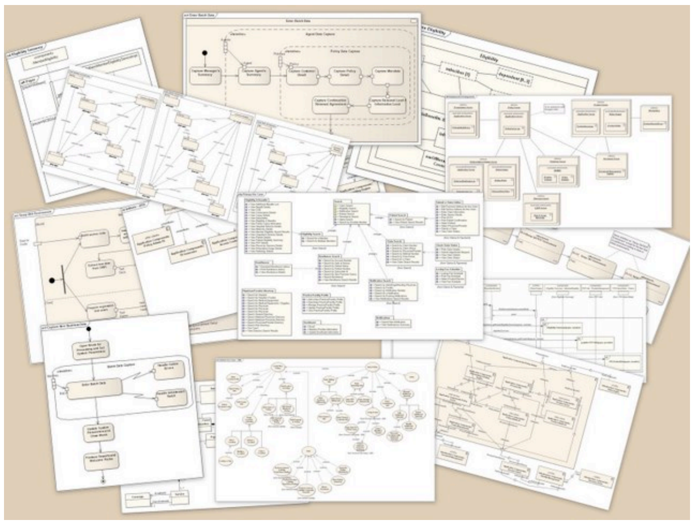
$$f : [0,w] \times [0,h] \to \{0,1\}, \quad w,h \in \mathbb{N}$$

(notice how nicely formal the language is!)

Tools already exist.

With the mere addition of a *TUML profile*, every existing UML notation is a special case!

Examples of TUML Models



[Image from Wikipedia Commons. Author: Kishorekumar 62]

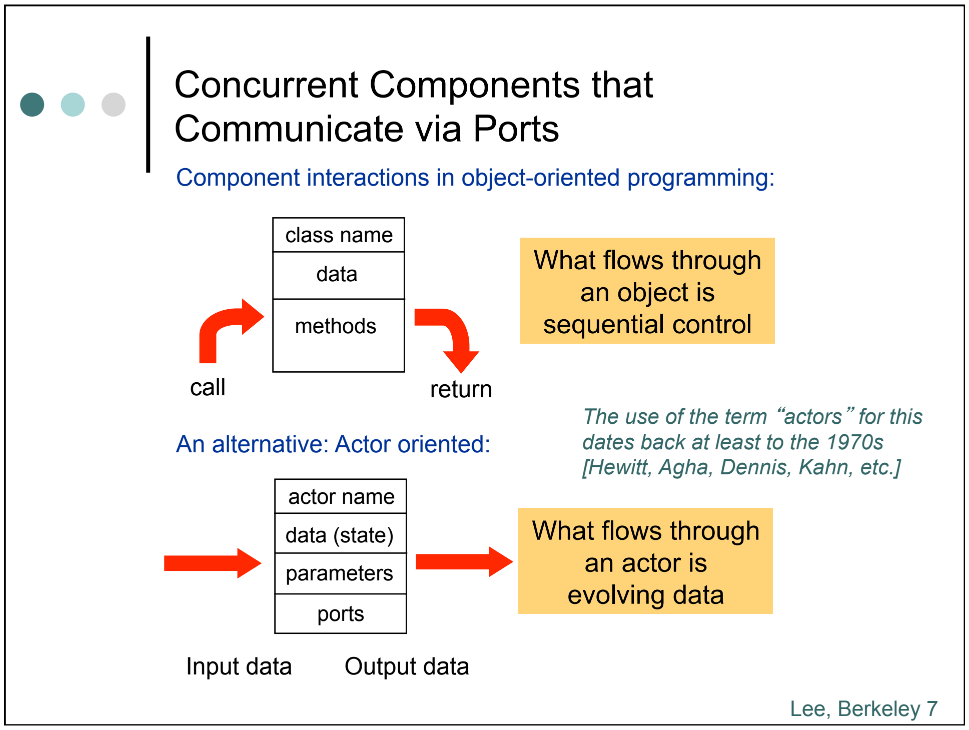
My Claim

Modeling languages that are not executable, or where the execution semantics is vague or undefined are not much better than TUML.

We can do better.

Assumptions of this Talk

- I am interested only in executable models
  (I will not comment about descriptive models)
- I focus on concurrent components that communicate via ports
  (as one might describe in SysML, AADL, or UML Component Diagrams & Communication Diagrams, though my take is more specific than any of these)



Concurrent Components that Communicate via Ports

Component interactions in object-oriented programming:

class name
data
methods
call
return

What flows through an object is sequential control

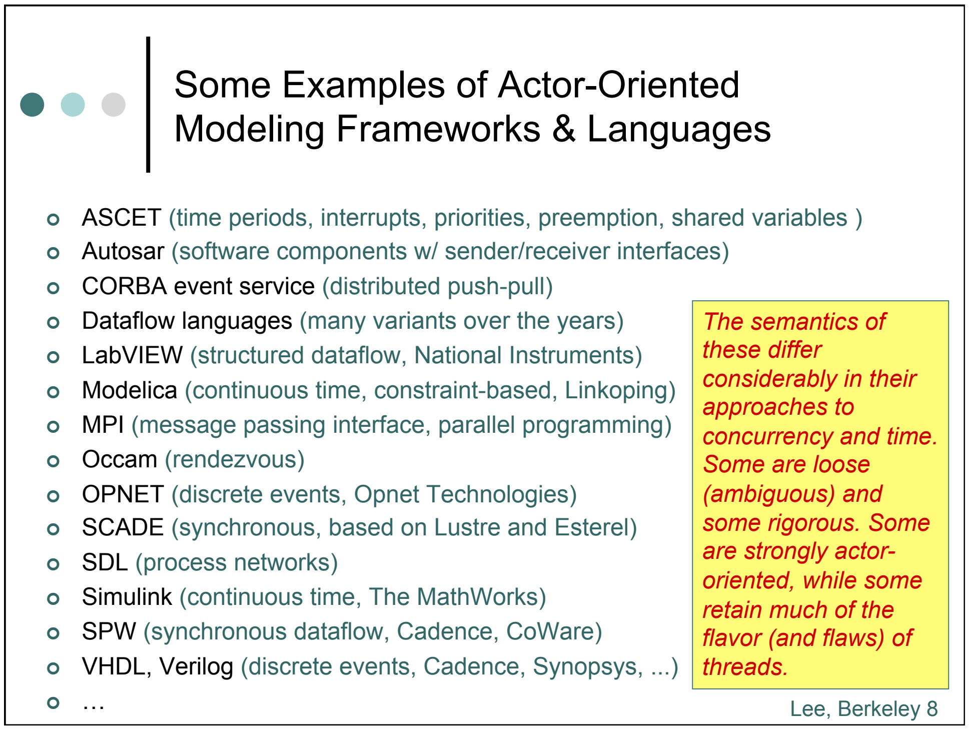
The use of the term "actors" for this dates back at least to the 1970s [Hewitt, Agha, Dennis, Kahn, etc.]

An alternative: Actor oriented:

actor name
data (state)
parameters
ports
Input data
Output data

What flows through an actor is evolving data

Lee, Berkeley 7

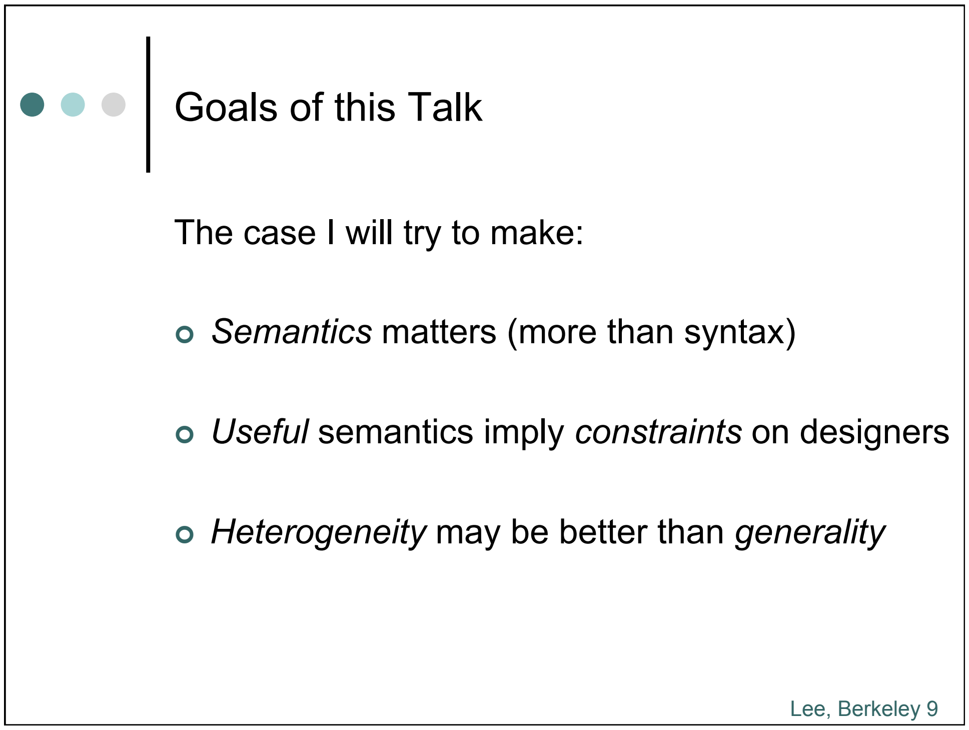


Some Examples of Actor-Oriented Modeling Frameworks & Languages

- ASCET (time periods, interrupts, priorities, preemption, shared variables)
- Autosar (software components w/ sender/receiver interfaces)
- CORBA event service (distributed push-pull)
- Dataflow languages (many variants over the years)
- LabVIEW (structured dataflow, National Instruments)
- Modelica (continuous time, constraint-based, Linkoping)
- MPI (message passing interface, parallel programming)
- Occam (rendezvous)
- OPNET (discrete events, Opnet Technologies)
- SCADE (synchronous, based on Lustre and Esterel)
- SDL (process networks)
- Simulink (continuous time, The MathWorks)
- SPW (synchronous dataflow, Cadence, CoWare)
- VHDL, Verilog (discrete events, Cadence, Synopsys, ...)
- …

The semantics of these differ considerably in their approaches to concurrency and time. Some are loose (ambiguous) and some rigorous. Some are strongly actor-oriented, while some retain much of the flavor (and flaws) of threads.

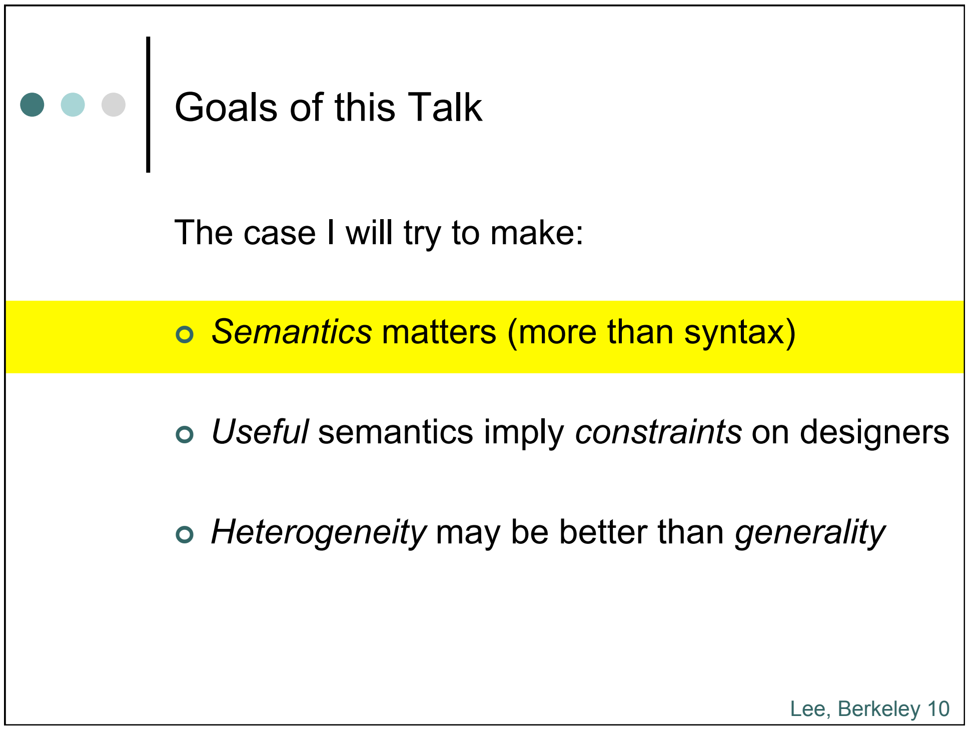
Lee, Berkeley 8



Goals of this Talk

The case I will try to make:

- *Semantics* matters (more than syntax)
- *Useful* semantics imply *constraints* on designers
- *Heterogeneity* may be better than *generality*

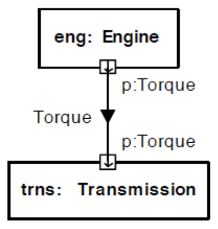


Goals of this Talk

The case I will try to make:

- *Semantics* matters (more than syntax)
- *Useful* semantics imply *constraints* on designers
- *Heterogeneity* may be better than *generality*

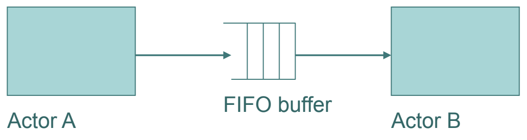
Consider SysML Flow Ports



Item Flow

“In general, flow ports are intended to be used for asynchronous, broadcast, or send-and-forget interactions.”

OMG Systems Modeling Language (OMG SysML™) Version 1.1, Nov. 2008



Seems like message passing.
But what kind of message passing?
To make this *executable*, we need some decisions.

Caution: Even the Pros can mess up semantics!

Consider MPI: A Popular Message Passing Library.

MPI is a collaborative standard developed since the early 1990s with many parallel computer vendors and stakeholders involved. Realized as a C and Fortran APIs. MPI programs are actor-oriented.

Lee, Berkeley 11

Vague MPI Send Semantics

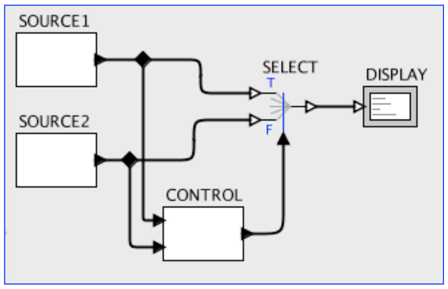
MPI_Send is a “blocking send:”

- It does not return until the memory storing the value to be sent can be safely overwritten.
- The MPI standard allows implementations to either copy the data into a “system buffer” for later delivery to the receiver, or to rendezvous with the receiving process and return only after the receiver has begun receiving the data.

This leads to programs that behave differently under different implementations of MPI!

Lee, Berkeley 12

What does this program do?



CONTROL Process:

```
MPI_Recv(&data1, 1, MPI_INT, SOURCE1, ...);
MPI_Recv(&data2, 1, MPI_INT, SOURCE2, ...);
while (1) {
  if (someCondition(data1, data2)) {
    MPI_Send(&trueValue, 1, MPI_INT, SELECT, ...);
    MPI_Recv(&data1, 1, MPI_INT, SOURCE1, ...);
  } else {
    MPI_Send(&falseValue, 1, MPI_INT, SELECT, ...);
    MPI_Recv(&data2, 1, MPI_INT, SOURCE2, ...);
  }
}
```

With buffered send it sorts the input source data.

With rendezvous it deadlocks!

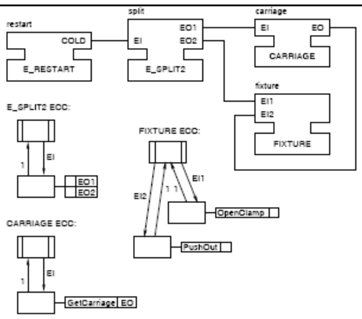
Irony

“The reluctance of MPI to mandate whether standard sends are buffering or not stems from the desire to achieve portable programs.”

Message Passing Interface Forum (2008). MPI: A Message Passing Interface standard -- Version 2.1, University of Tennessee, Knoxville, Tennessee.

“Portability” apparently means that it runs on different platforms, not that it has the same behavior on different platforms.

Another "Failure" IEC 61499



International Electrotechnical Commission (IEC) 61499 is a standard established in 2005 for distributed control systems software engineering for factory automation.

The standard is (apparently) inspired by formal composition of state machines, and is intended to facilitate formal verification.

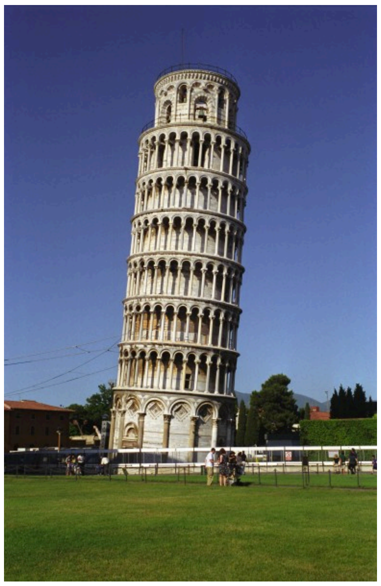
Regrettably, the standard essentially fails to give a concurrency model, resulting in different behaviors of the same source code on runtime environments from different vendors, and (worse) nondeterministic behaviors on runtimes from any given vendor.

See: Ĉengić, G., Ljungkrantz, O. and Åkesson, K., Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime. in *11th IEEE International Conference on Emerging Technologies and Factory Automation*, (Prague, Czech Republic 2006).

Building on Weak Foundations

The semantics of a modeling language is the foundation for the models.

Weak foundations result in less useful models.



Goals of this Talk

The case I will try to make:

- *Semantics* matters (more than syntax)
- *Useful* semantics imply *constraints* on designers
- *Heterogeneity* may be better than *generality*

Does "More General" Always mean "Better"?

Obsessive flexibility can lead to languages like Jisp++:

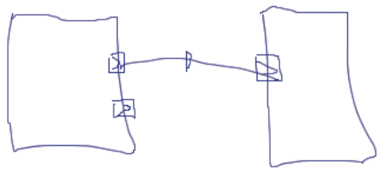
```
List myList
   = new List((cons (cdr foo->bar)).elements());
```

Usable design practice implies:

"freedom *from* choice"

[Alberto Sangiovanni-Vincentelli, on *platform-based design*].

So what does this SysML model mean?



One possibility: Asynchronous, stream-based message passing.

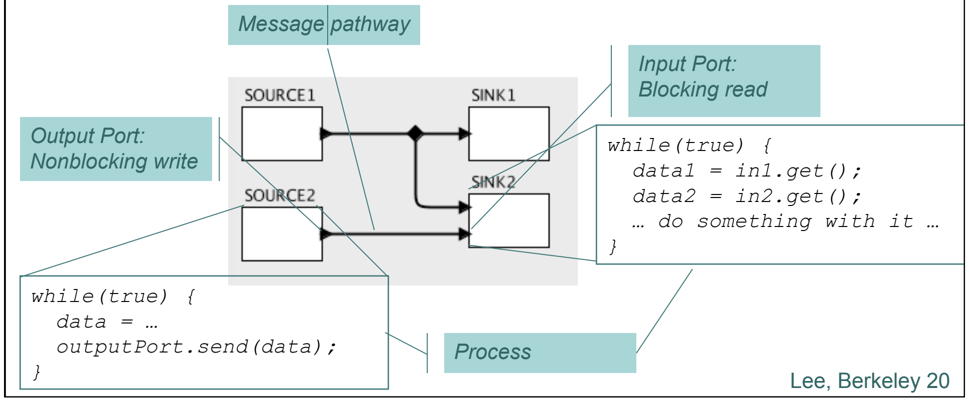
We should use the most general notion of streams, right?

But the most general notion of streams in nondeterminate.

- Kahn [1974] gave a theory for *determinate* streams.
- The MoC is known as Kahn Process Networks (KPN).
- But determinism is constraining.
  Some useful models cannot be built with KPN.

Kahn Process Networks

- Kahn & MacQueen [1977] showed that *blocking reads* and *nonblocking writes* in concurrent processes are sufficient to ensure determinate models.
- But blocking reads are quite constraining!



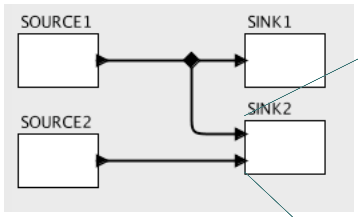
```
while(true) {
  data1 = in1.get();
  data2 = in2.get();
  … do something with it …
}
```

```
while(true) {
  data = …
  outputPort.send(data);
}
```

Generalizing KPN with Non-Blocking Reads

One option is to loosen the constraint and allow processes to test for availability of input data.

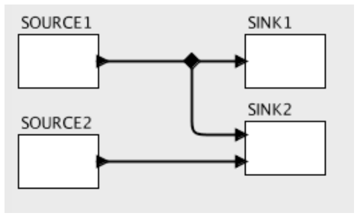
This leads to nondeterminate models.



```
while(true) {
  if (in1.hasData()) {
    data1 = in1.get();
  }
  if (in2.hasData()) {
    data2 = in2.get();
  }
  … do something with it …
}
```

Is the distinction between blocking and nonblocking reads too much detail?

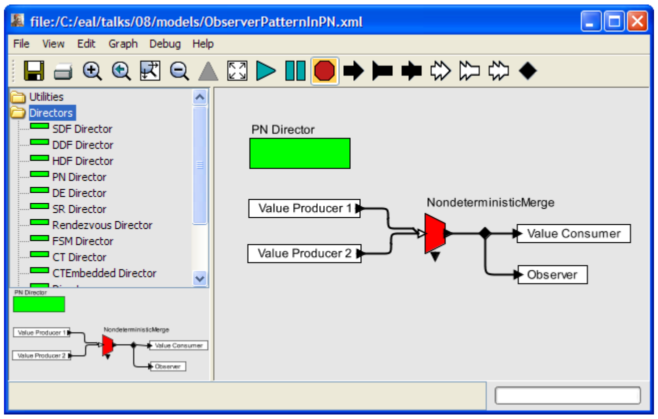
What does this model mean?



Unless we commit to a semantics, the model means different things to different observers!

The Tension in this Case: Generality vs. Determinacy?

The Ptolemy II PN director gives one possible resolution to this tension.



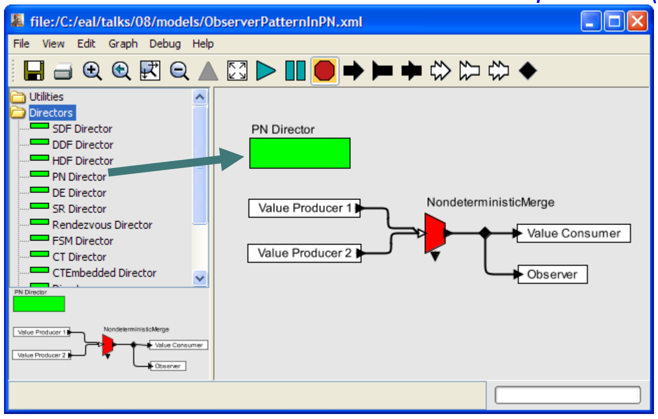
This modeling language implements KPN (preserving determinacy), but provides explicit notations for nondeterminate extensions.

We achieve generality without weakening the core semantics!

The Ptolemy Approach:
A single framework / many modeling languages

Directors define semantics

Directors annotate models, endowing them with an executable semantics, typically a concurrent model of computation (MoC)



Directors are defined and designed by experts (typically).

Asynchronous stream-based message passing has many additional subtleties.

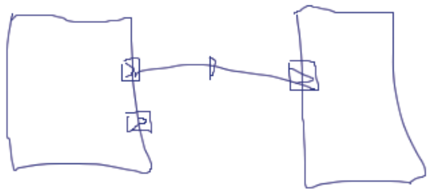
- Bounding the buffers.
- Termination, deadlock, and livelock (halting)
- Fairness
- Parallelism
- Data structures and shared data

Dataflow models of computation provide ways to formalize asynchronous stream-based message passing.

A few variants of dataflow MoCs

- *Computation graphs [Karp and Miller, 1966]*
- *Static dataflow [Dennis, 1974]*
- *Dynamic dataflow [Arvind, 1981]*
- *Structured dataflow [Matwin & Pietrzykowski 1985]*
- *K-bounded loops [Culler, 1986]*
- *Synchronous dataflow [Lee & Messerschmitt, 1986]*
- *Structured dataflow and LabVIEW [Kodosky, 1986]*
- *PGM: Processing Graph Method [Kaplan, 1987]*
- *Synchronous languages [Lustre, Signal, 1980's]*
- *Well-behaved dataflow [Gao, 1992]*
- *Boolean dataflow [Buck and Lee, 1993]*
- *Multidimensional SDF [Lee, 1993]*
- *Cyclo-static dataflow [Lauwereins, 1994]*
- *Integer dataflow [Buck, 1994]*
- *Bounded dynamic dataflow [Lee and Parks, 1995]*
- *Heterochronous dataflow [Girault, Lee, & Lee, 1997]*
- *…*

A TUML Dataflow Diagram



Dataflow models can be described with vague, underdefined modeling languages.

But should the model builder be asked to handle the considerable subtleties?

Few of them will get it right…

Useful Modeling Languages with Strong Semantics

Useful executable modeling languages impose *constraints* on the designer.

The constraints may come with benefits.

We have to stop thinking of constraints as a universal negative!!!

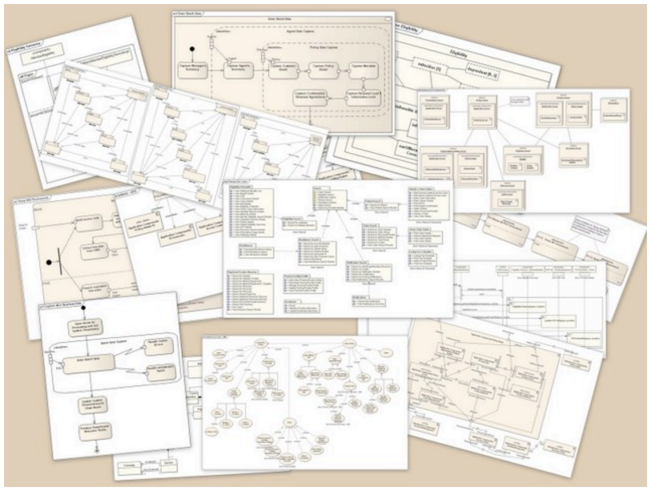
Freedom from choice!!!

Goals of this Talk

The case I will try to make:

- *Semantics* matters (more than syntax)
- *Useful* semantics imply *constraints* on designers
- *Heterogeneity* may be better than *generality*

Heterogeneity may be Better than Generality



The UML community does not need my help to embrace heterogeneity.

With the mere addition of profile, most of these diagrams can describe anything that another diagram can describe.

Is this better than TUML?

[Image from Wikipedia Commons. Author: Kishorekumar 62]

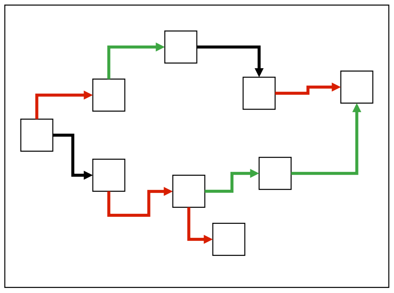
Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling:** hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.
- **multi-view modeling:** distinct and separate models of the same system are constructed to model different aspects of the system.
- **meta modeling:** use of models together with models of the modeling language.

Amorphous vs. Hierarchical Heterogeneity

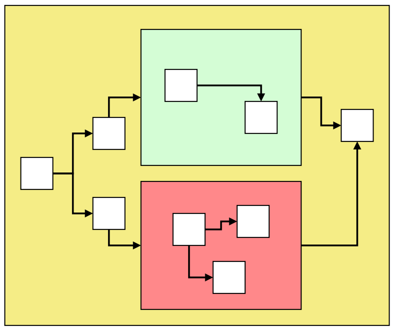
Amorphous



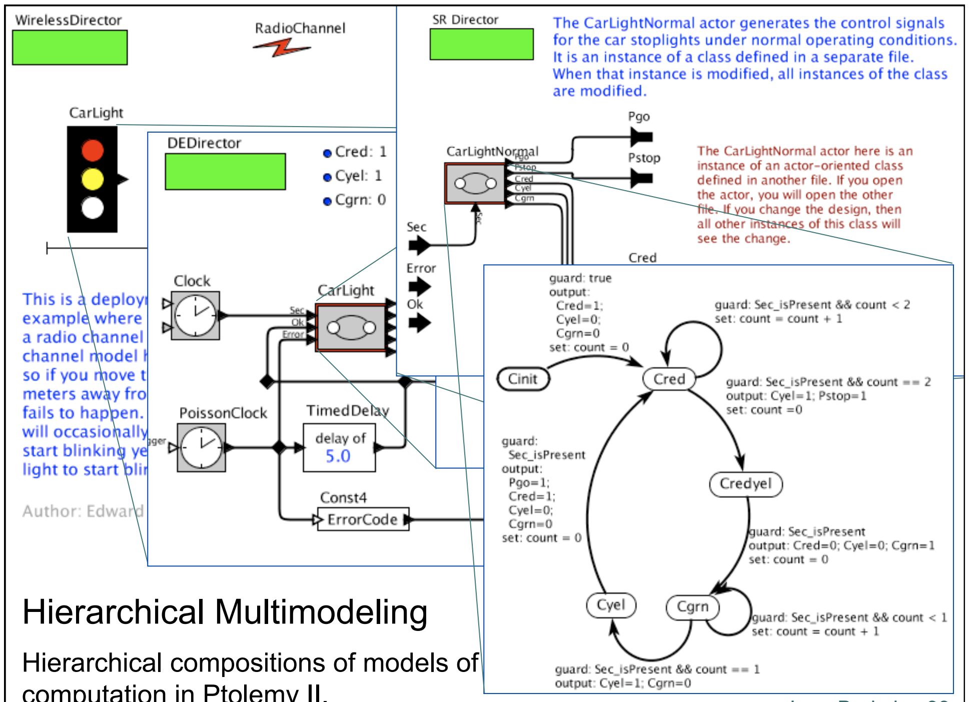
Color is an interaction style (KPN, pub/sub, procedure call, …).

How to understand the model?

Hierarchical



Color is an MoC, with constraints. The meaning is clear at each level of the hierarchy.



The CarLightNormal actor generates the control signals for the car stoplights under normal operating conditions. It is an instance of a class defined in a separate file. When that instance is modified, all instances of the class are modified.

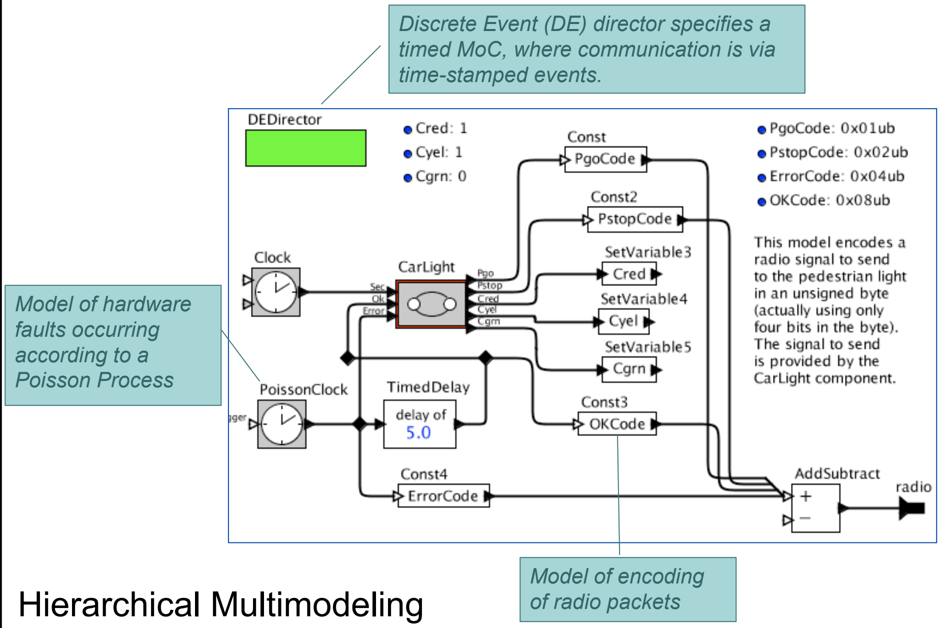
The CarLightNormal actor here is an instance of an actor-oriented class defined in another file. If you open the actor, you will open the other file. If you change the design, then all other instances of this class will see the change.

This is a deploy
example where
a radio channel
channel model
so if you move t
meters away fro
fails to happen.
will occasionally
start blinking ye
light to start blir

Author: Edward

Hierarchical Multimodeling

Hierarchical compositions of models of computation in Ptolemy II.



Discrete Event (DE) director specifies a timed MoC, where communication is via time-stamped events.

Model of hardware faults occurring according to a Poisson Process

This model encodes a radio signal to send to the pedestrian light in an unsigned byte (actually using only four bits in the byte). The signal to send is provided by the CarLight component.

Model of encoding of radio packets

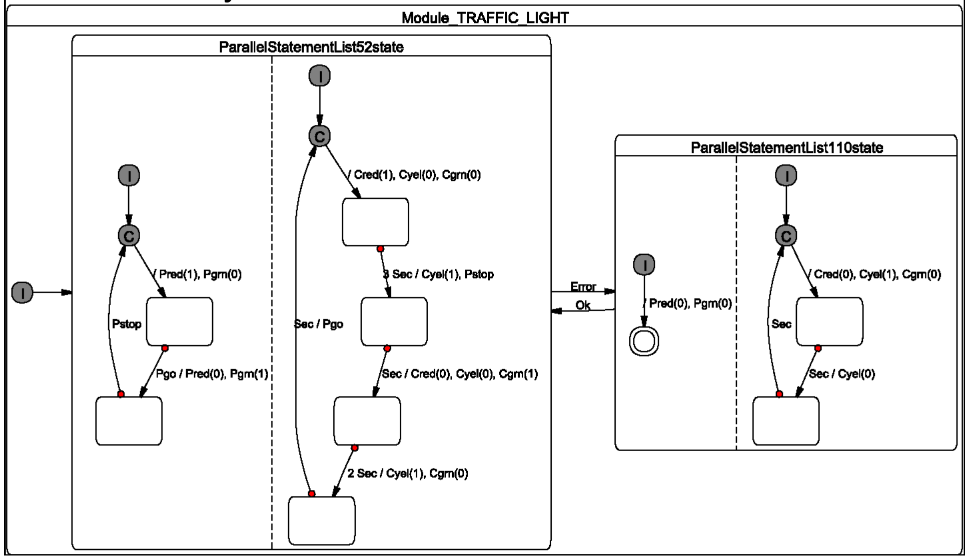
Hierarchical Multimodeling

At each level of the hierarchy, the model is understandable.

Background on Hierarchical Multimodeling

- Statecharts [Harel 87]
- Ptolemy Classic [Buck, Ha, Lee, Messerschmitt 94]
- SyncCharts [André 96]
- *Charts [Girault, Lee, Lee 99]
- Colif [Cesario, Nicolescu, Guathier, Lyonnard, Jerraya 01]
- Metropolis [Goessler, Sangiovanni-Vincentelli 02]
- Ptolemy II [Eker, et. al. 03]
- Safe State Machine (SSM) [André 03]
- SCADE [Berry 03]
- ForSyDe [Jantsch, Sander 05]
- ModHelX [Hardebolle, Boulanger07]

Statecharts are hierarchical combinations of state machines with a (sometimes poorly defined) concurrency model.



How Does This Work in Ptolemy II? Actor Abstract Semantics

Actions invoked on an actor by a director:

- Preinitialization
- Initialization
- Execution
- Finalization

Compare to Profiles

Note that while you can, in principle, use profiles with SysML to get Kahn process networks, discrete-event models, finite state machines, synchronous/reactive models, and dataflow models, there is nothing about SysML that will ensure that they interoperate…

In Ptolemy, they interoperate.

Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling:** hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.
- **multi-view modeling:** distinct and separate models of the same system are constructed to model different aspects of the system.
- **meta modeling:** use of models together with models of the modeling language.

Multimodeling

Simultaneous use of multiple modeling techniques.

- **hierarchical multimodeling:** hierarchical compositions of distinct modeling styles, combined to take advantage of the unique capabilities and expressiveness of each style.

No Time for These
(invite me back!)

Modeling in an Artificial Universe

Components of a model interact in an artificial universe.
An MoC provides the "laws of physics" of the artificial universe.

Nonnegotiable requirement:
The laws of physics must be the same to every observer.

Desirable property:
The laws of physics should result in understandable models.

Not necessary:
The laws need not easily express every possible model. Use heterogeneity.