Model Engineering using Multimodeling

Christopher Brooks (UC Berkeley)
Chih-Hong Cheng (UC Berkeley & TU Munich)
Thomas Huining Feng (UC Berkeley)
Edward A. Lee (UC Berkeley)
Reinhard von Hanxleden (Christian-Albrechts-Univ. Kiel)

1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 2008)

September 30, 2008

Toulouse, France

Summary: *Model Engineering*

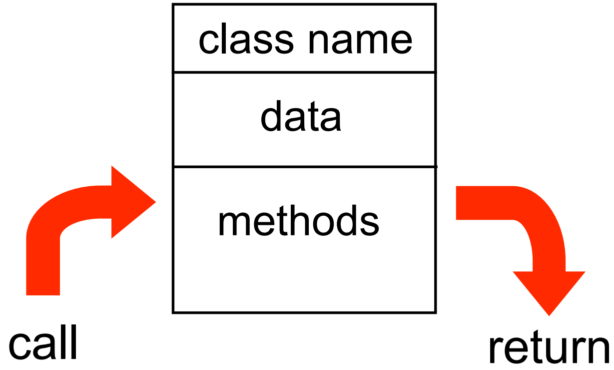
This project is about "model engineering" for model-based design of scalable systems of systems. Analogous to "software engineering," which enables scaling up software development efforts, "model engineering" enables scaling up of model-based design.

Our approach focuses on technologies rather than design process. Specifically, we are concerned with models of system dynamics (such as actor models) more than with static structure (such as UML class diagrams), with data ontologies (which associate data structures with their meaning) more than data types (which associate data structures with their layout in memory), and with heterogeneous systems (such as hybrid systems and multimodeling) more than homogenized systems.

Acknowledgement: This work is heavily influenced by our colloboration with Lockheed Martin, particularly Trip Denton and Edward Jones.

Our Premise:
Components are Actors rather than Objects

The established: Object-oriented:

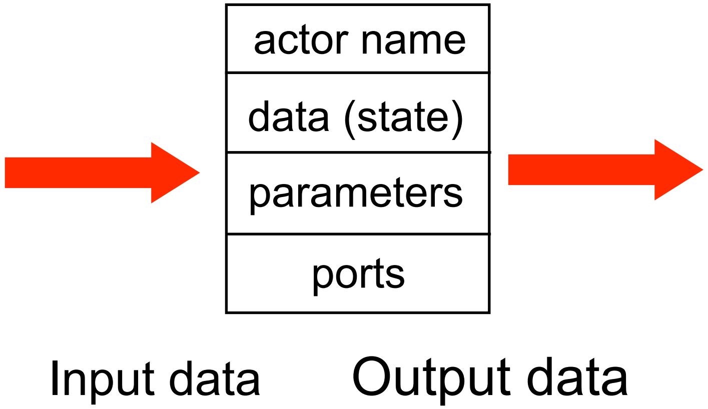


What flows through an object is sequential control

Things happen to objects

The alternative: Actor oriented:

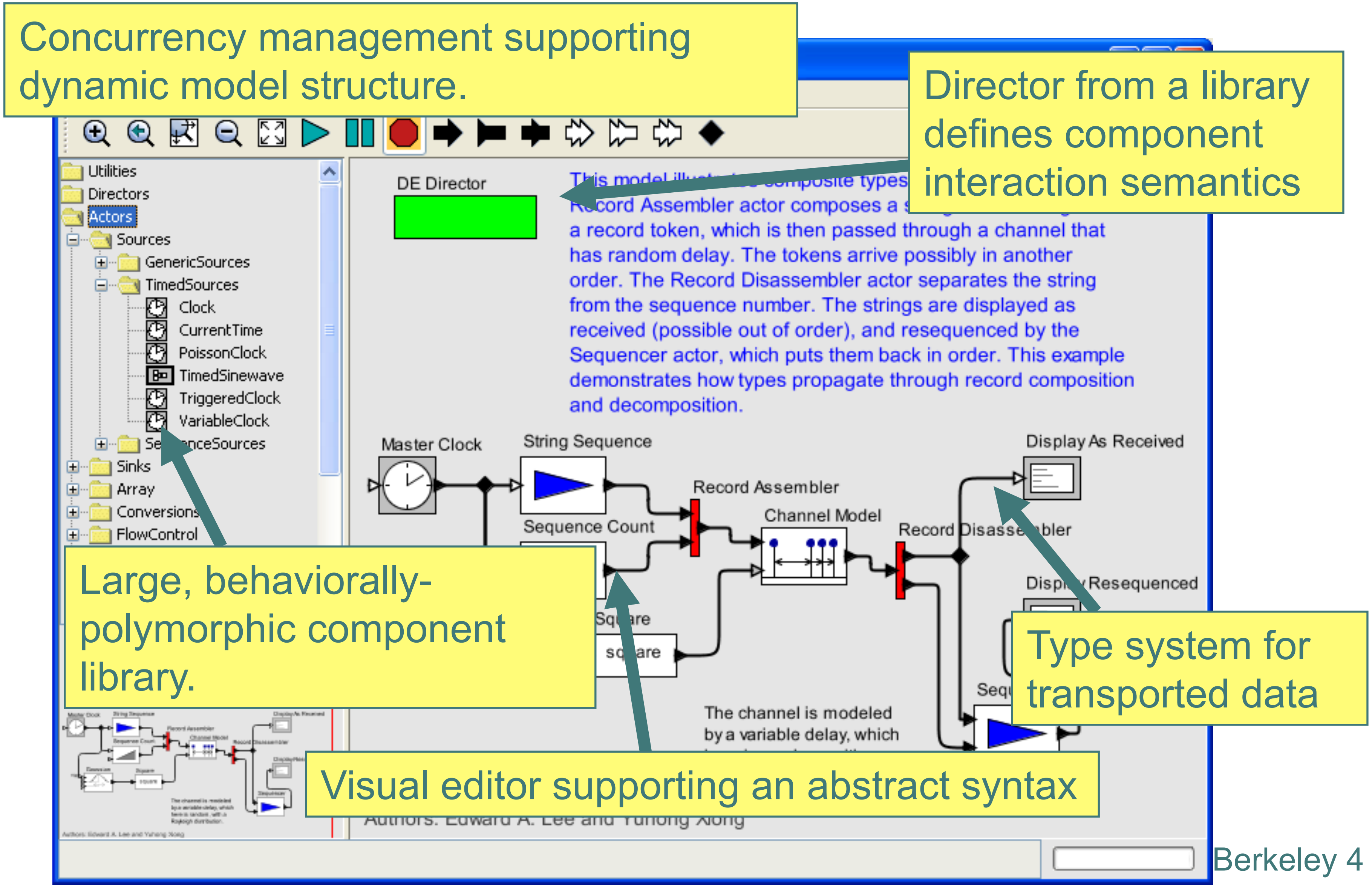
Actors make things happen



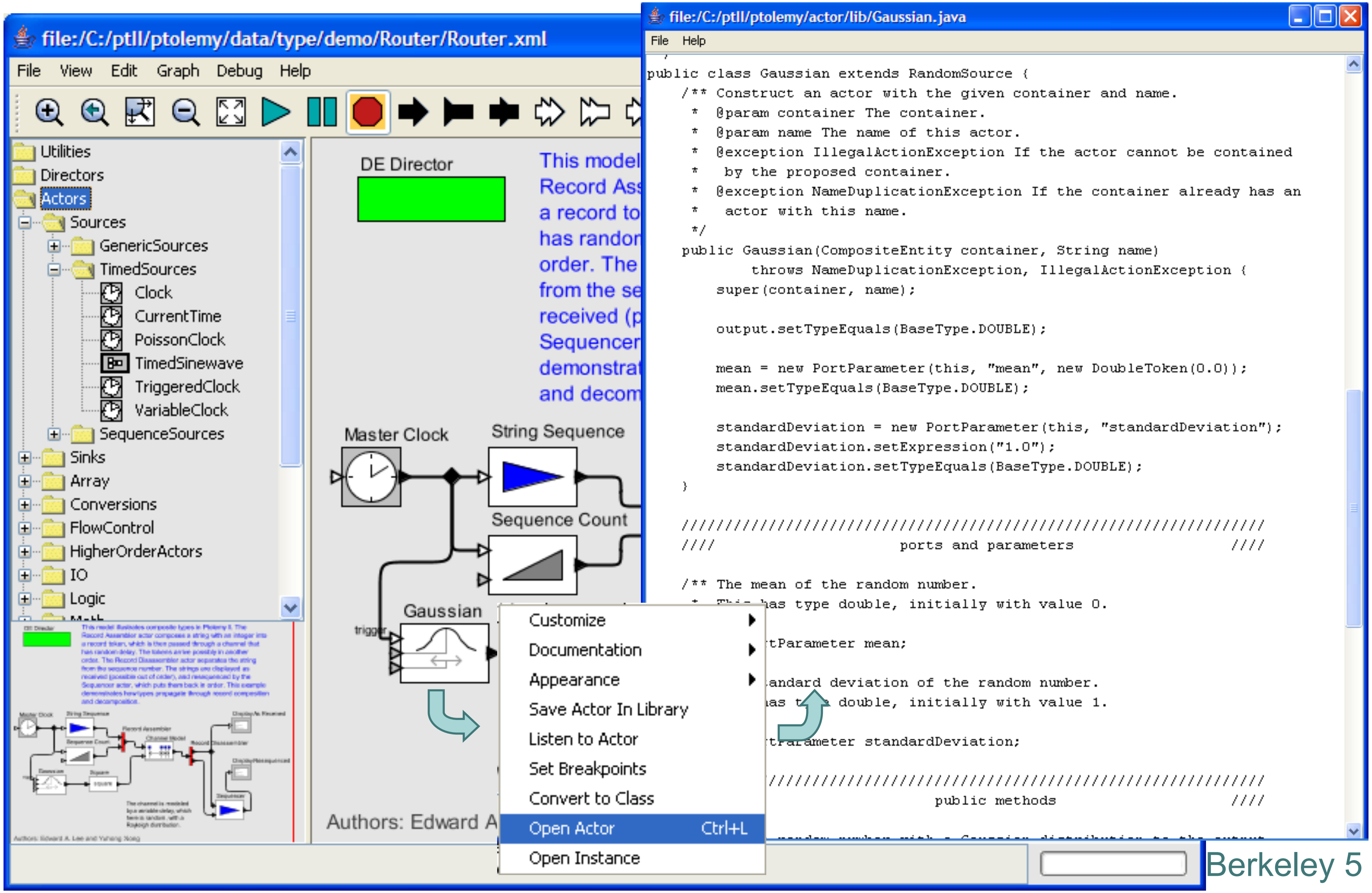
What flows through an object is evolving data

Ptolemy II: Our Open-Source Laboratory for Experiments with Actor-Oriented Design

http://ptolemy.org



Approach: Concurrent Composition of Software Components, which are themselves designed with Conventional Languages (Java, C, C++ MATLAB, Python)

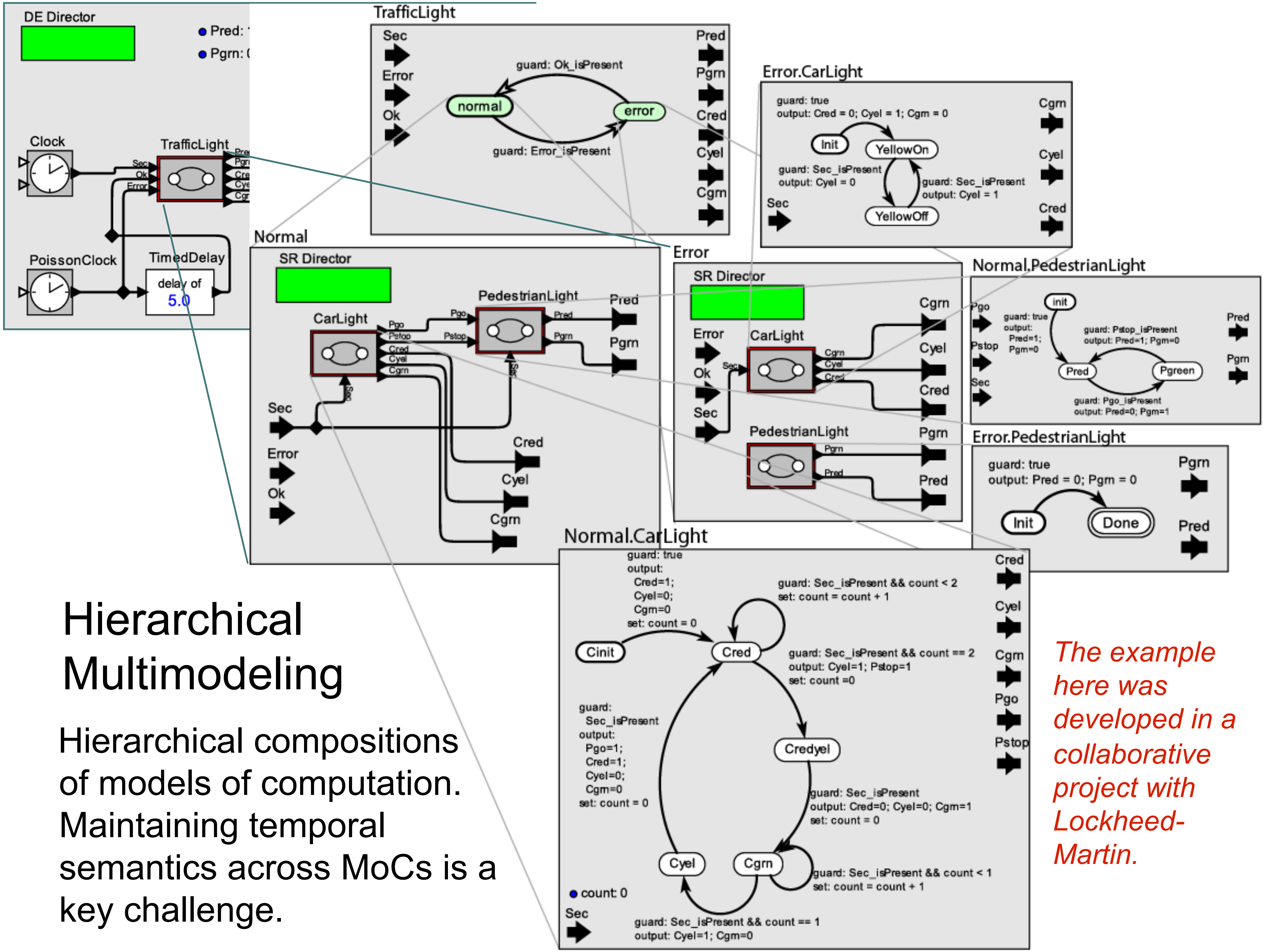


Multimodeling

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.



Hierarchical Multimodeling

Hierarchical compositions of models of computation. Maintaining temporal semantics across MoCs is a key challenge.

The example here was developed in a collaborative project with Lockheed-Martin.

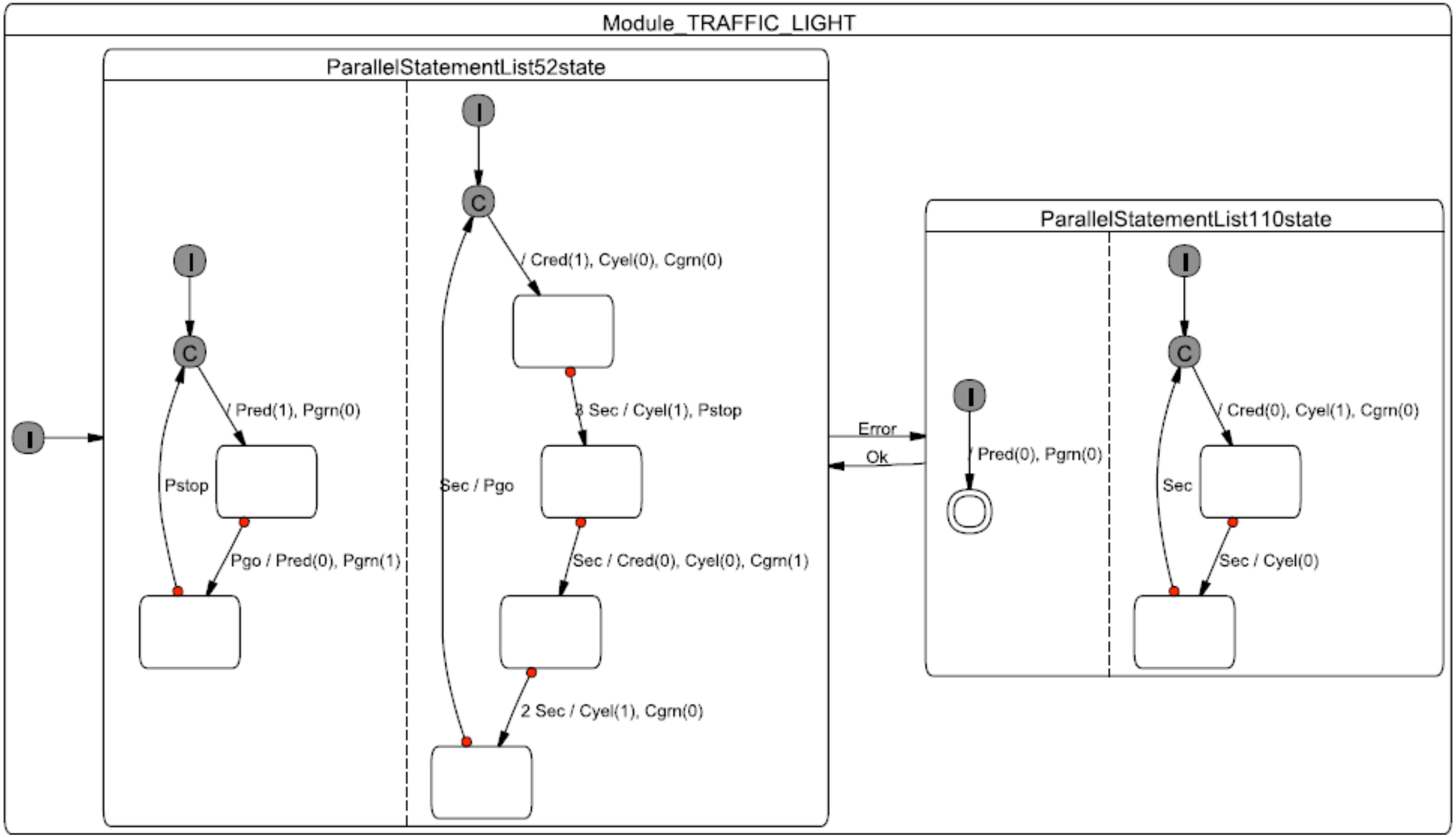
Background on Hierarchical Multimodeling

- 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 [Jantsch, Sander 07]

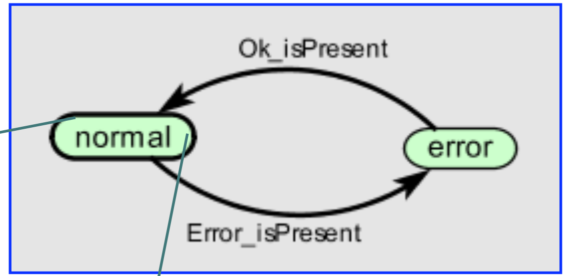
Simple Traffic Light Example in Statecharts

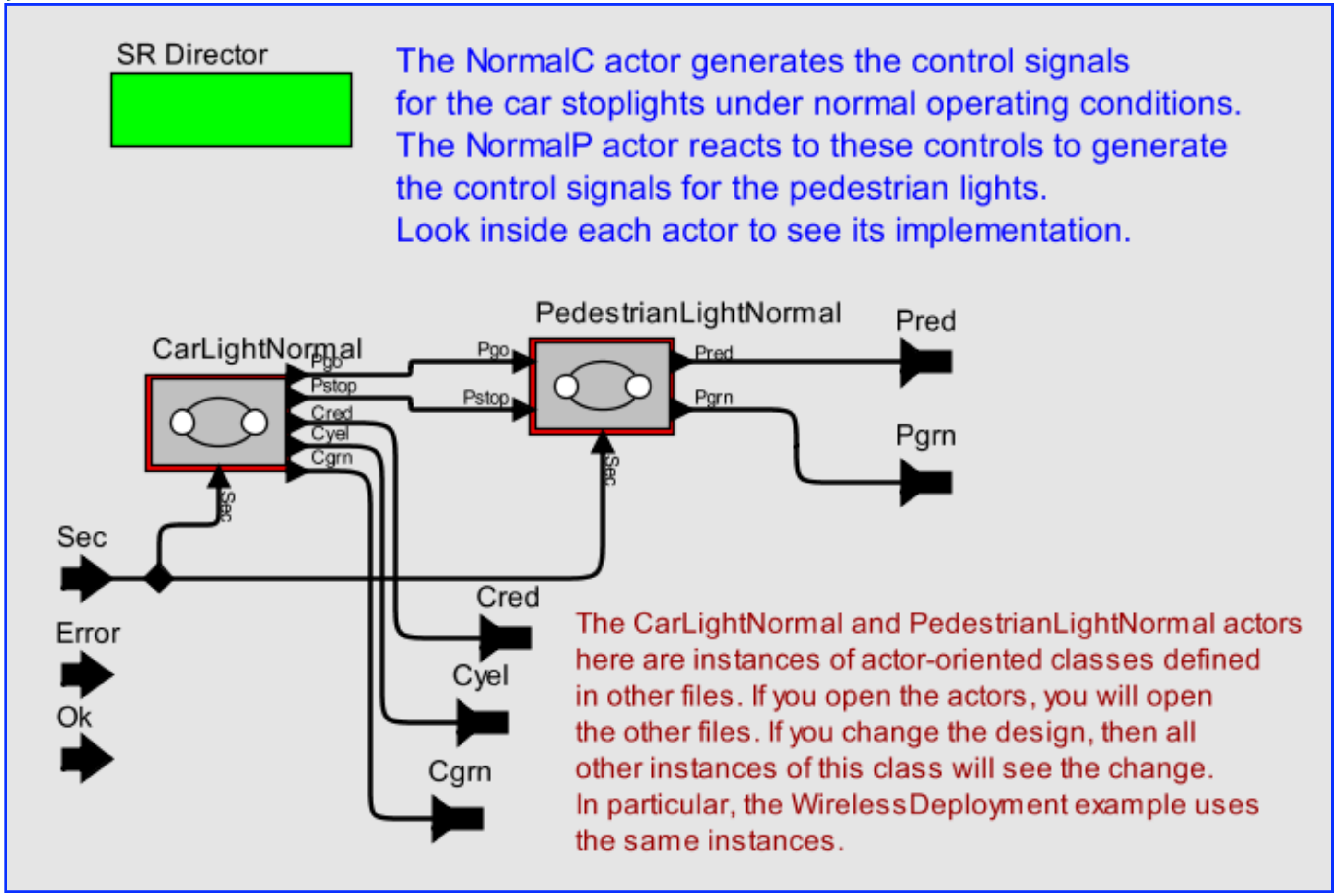
Case study

- *Pred: pedestrian red signal*
- *Pgrn(0): turn pedestrian green off*
- *Cgrn: car green*
- *Sec: one second time*
- *2 Sec: two seconds time*
- *Pgo/Pstop: pedestrian go/stop*



Traffic Light Example in Ptolemy II



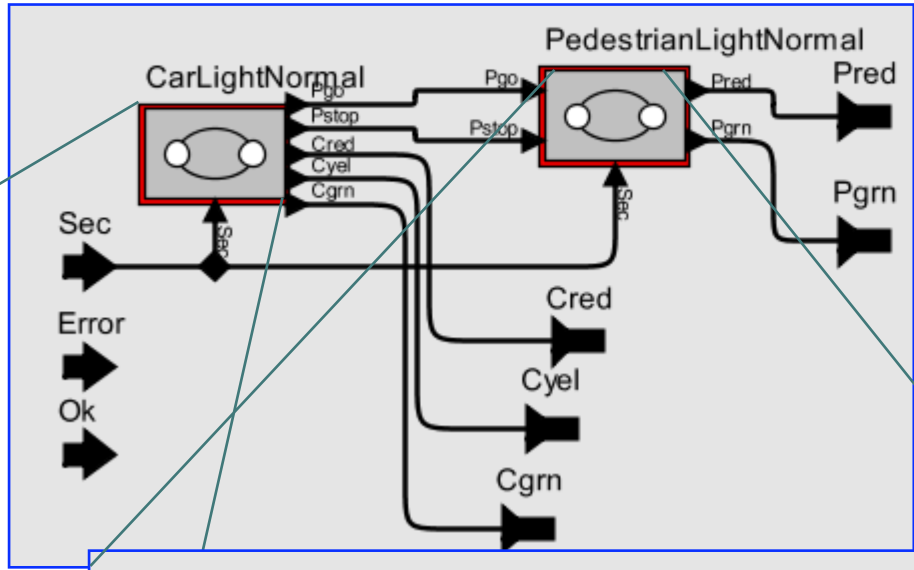


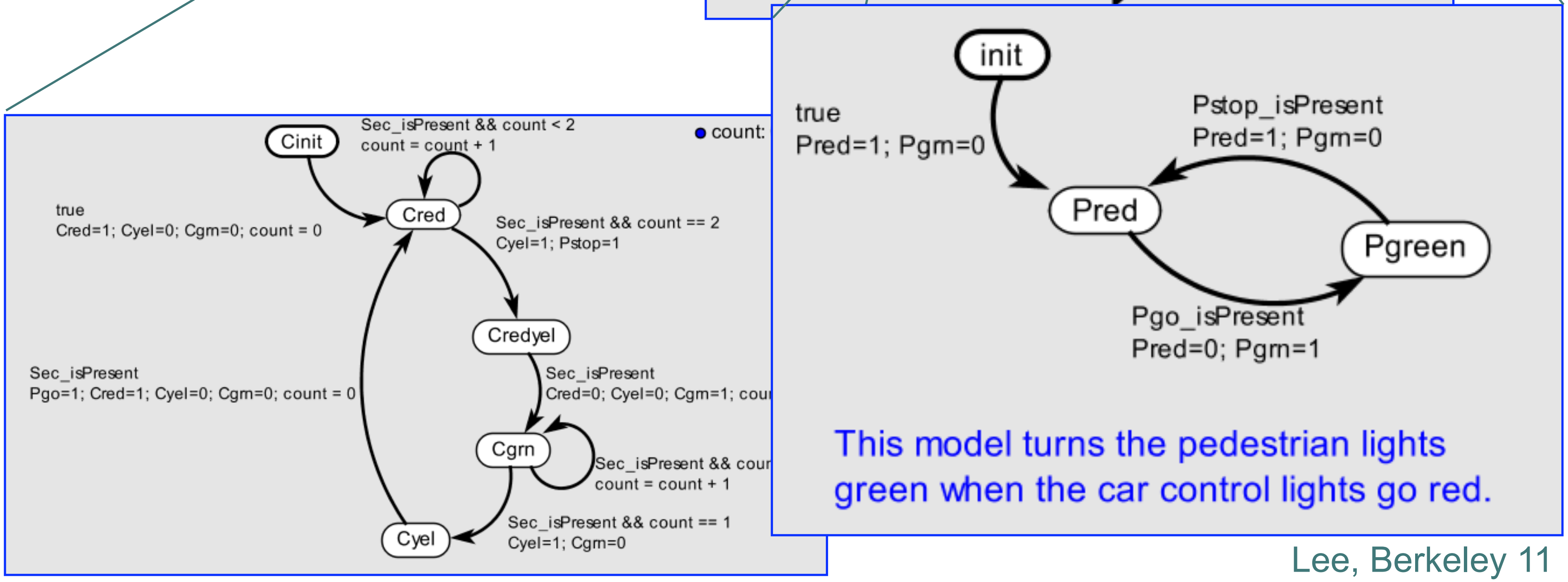
Whereas Statecharts lumps together the state machine semantics and the concurrency model, Ptolemy II separates these.

Here we have chosen the SR Director, which realizes a true synchronous fixed point semantics.

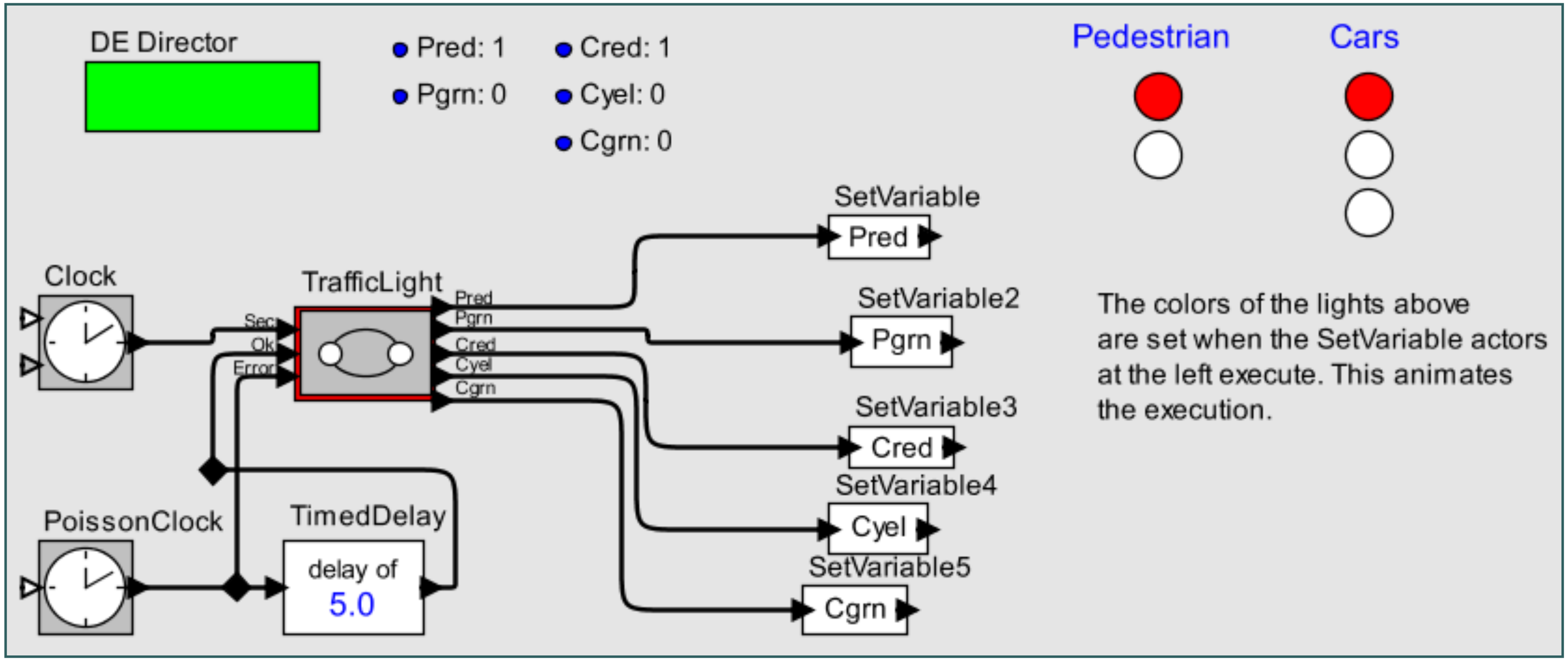
Concurrent State Machines in Ptolemy II

In Ptolemy II, we have implemented an SR Director (for synchronous concurrent models) and an FSM Director (for sequential decision logic). Rather than combining them into one language (like Statecharts), Ptolemy II supports hierarchical combinations of MoCs.





Stepping Outside Statecharts: Modeling the Environment



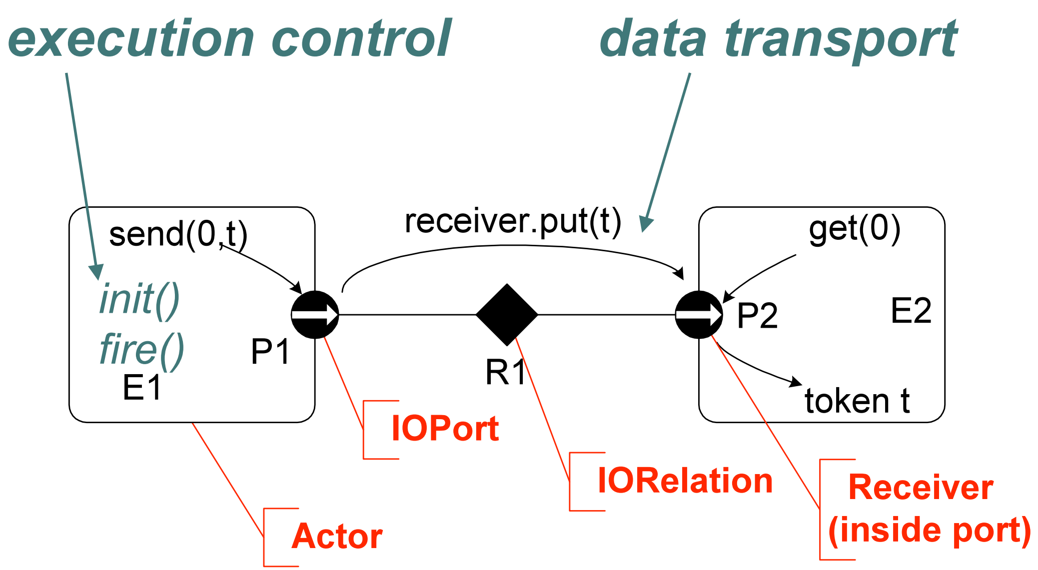
The above model places the TrafficLight model in a discrete-event testbench that clocks the light an injects faults according to a stochastic model.

What Makes This Possible: The Ptolemy II Actor Abstract Semantics

- Abstract Syntax
- Concrete Syntax
- Type System
- Abstract Semantics
- Concrete Semantics

Abstract Semantics (Informally)

of *Actor-Oriented* Models of Computation



Actor-Oriented Models of Computation that follow this:

- *dataflow (several variants)*
- *process networks*
- *distributed process networks*
- *Click (push/pull)*
- *continuous-time*
- *CSP (rendezvous)*
- *discrete events*
- *distributed discrete events*
- *synchronous/reactive*
- *time-driven (several variants)*
- *…*

How Does This Work?
Execution of Ptolemy II Actors

Flow of control:

- Preinitialization
- Initialization
- Execution
- Finalization

How Does This Work?
Execution of Ptolemy II Actors

Flow of control:

- Preinitialization
- Initialization
- Execution
- Finalization

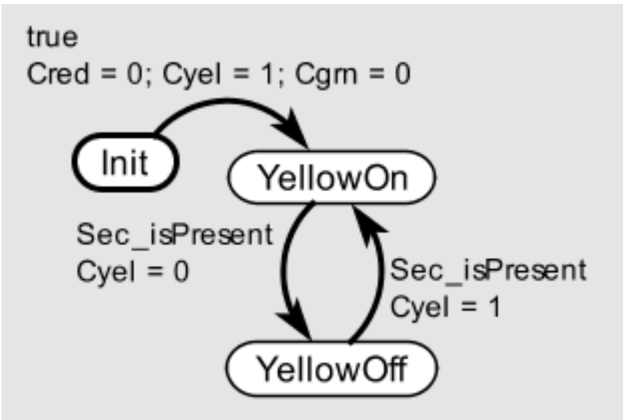
E.g., Partial evaluation (esp. higher-order components), set up type constraints, etc. Anything that needs to be done prior to static analysis (type inference, scheduling, …)

How Does This Work?
Execution of Ptolemy II Actors

Flow of control:

- Preinitialization
- Initialization
- Execution
- Finalization

E.g., Initialize actors, produce initial outputs, etc.

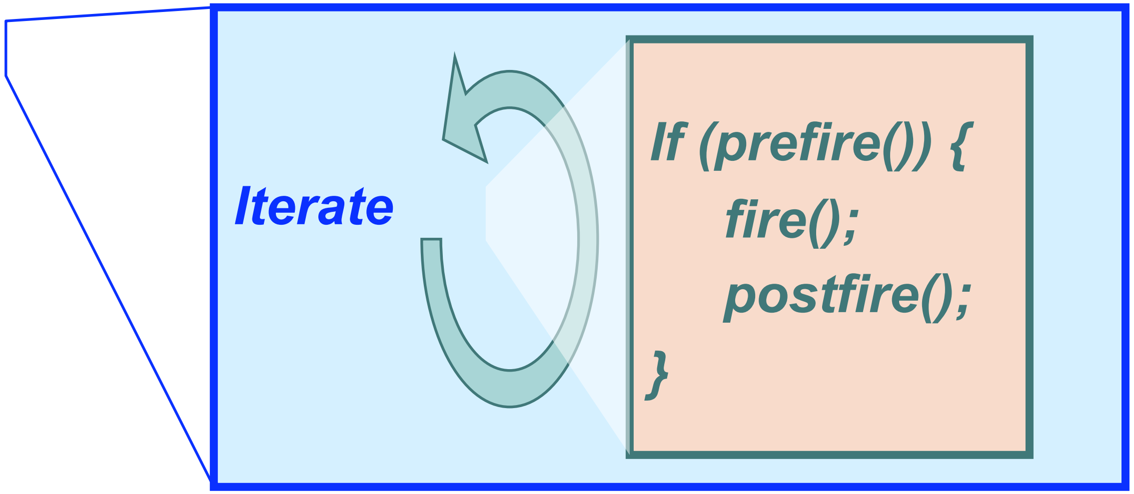


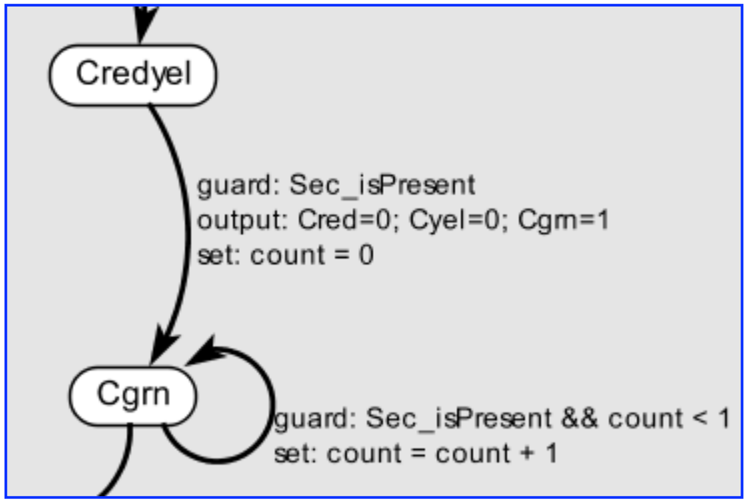
E.g., set the initial state of a state machine. Initialization may be repeated during the run (e.g. if the reset parameter of a transition is set and the destination state has a refinement).

How Does This Work?
Execution of Ptolemy II Actors

Flow of control:

- Preinitialization
- Initialization
- Execution
- Finalization





In fire(), an FSM first fires the refinement of the current state (if any), then evaluates guards, then produces outputs specified on an enabled transition. In postfire(), it postfires the current refinement (if any), executes set actions on an enabled transition, and takes the transition.

How Does This Work? Execution of Ptolemy II Actors

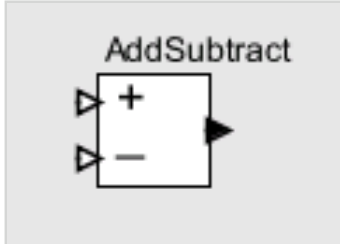
Flow of control:

- Preinitialization
- Initialization
- Execution
- Finalization

A Consequence of Our Abstract Semantics: Behavioral Polymorphism

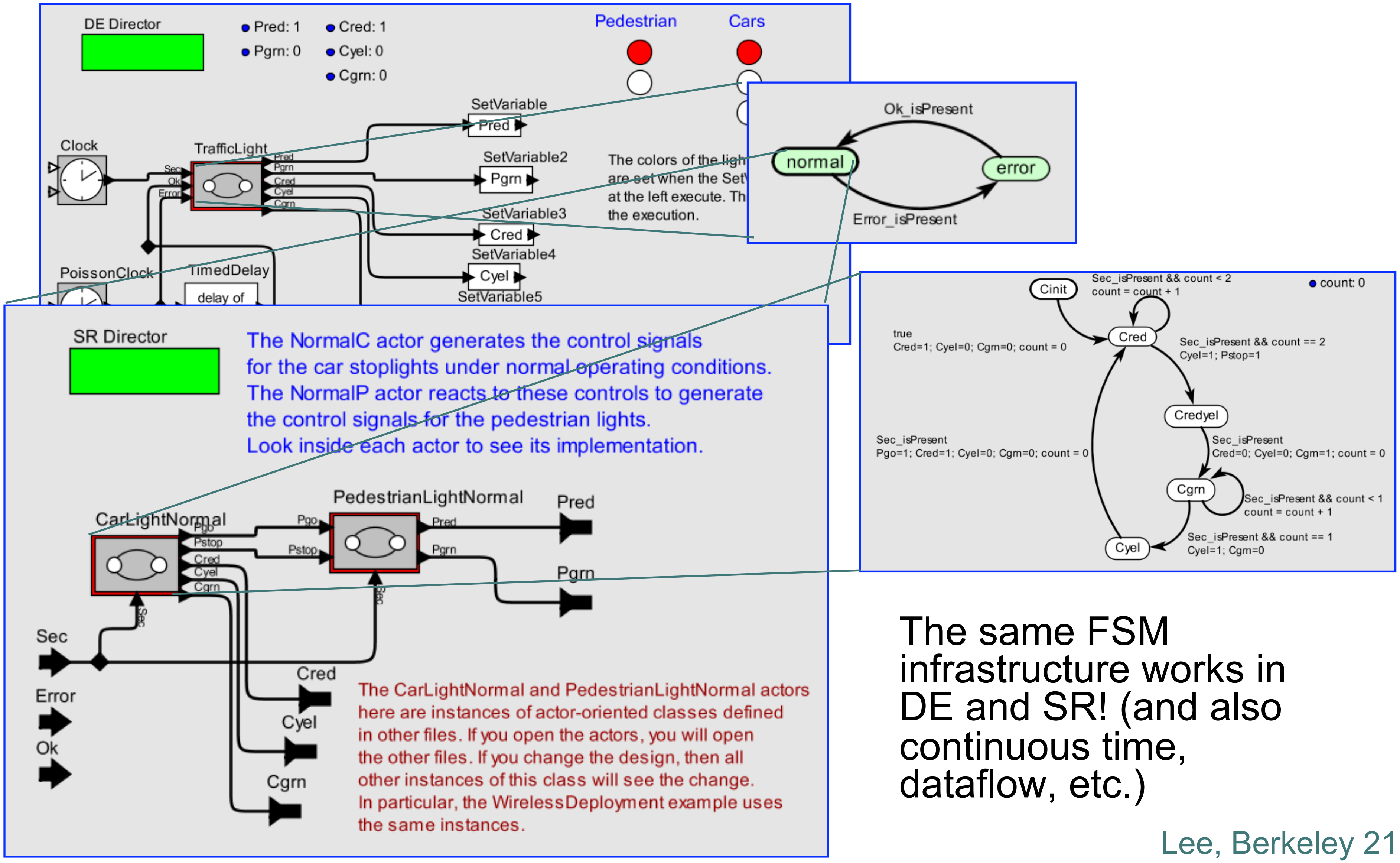
- Data polymorphism:
  - Add numbers (int, float, double, Complex)
  - Add strings (concatenation)
  - Add composite types (arrays, records, matrices)
  - Add user-defined types

- Behavioral polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a synchronous/reactive 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 rendezvous, 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



By not choosing among these when defining the component, we get a huge increment in component re-usability. Abstract semantics ensures that the component will work in all these circumstances.

More Interestingly, Hierarchical Models are Also Behaviorally Polymorphic



The same FSM infrastructure works in DE and SR! (and also continuous time, dataflow, etc.)

Separable Tool Archictecture

- Abstract Syntax
- Concrete Syntax
- Type System
- Abstract Semantics
- Concrete Semantics

Concrete Models of Computation Implemented in Ptolemy II

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- Continuous – continuous-time modeling with fixed-point semantics
- CT – continuous-time modeling
- DDF – Dynamic dataflow
- DE – discrete-event systems
- DDE – distributed discrete events
- DPN – distributed process networks
- FSM – finite state machines
- DT – discrete time (cycle driven)
- Giotto – synchronous periodic
- GR – 3-D graphics
- PN – process networks
- Rendezvous – extension of CSP
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

FSMs can be embedded in all of these (including FSMs). Many of these (but not all) can be embedded within state refinements of FSMs and/or within composite actors. See [Goderis, Brooks, Altintas, Lee, Goble, 2007]

Multimodeling

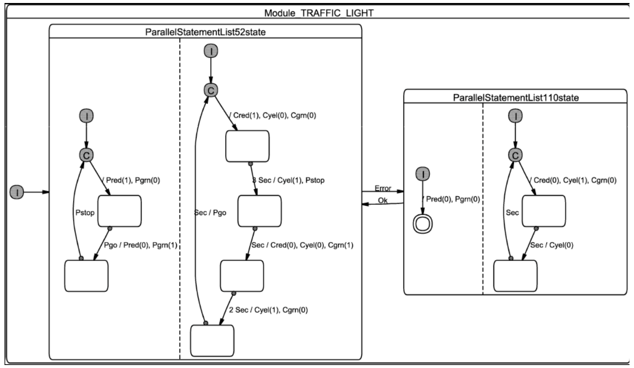
Simultaneous use of multiple modeling techniques.

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

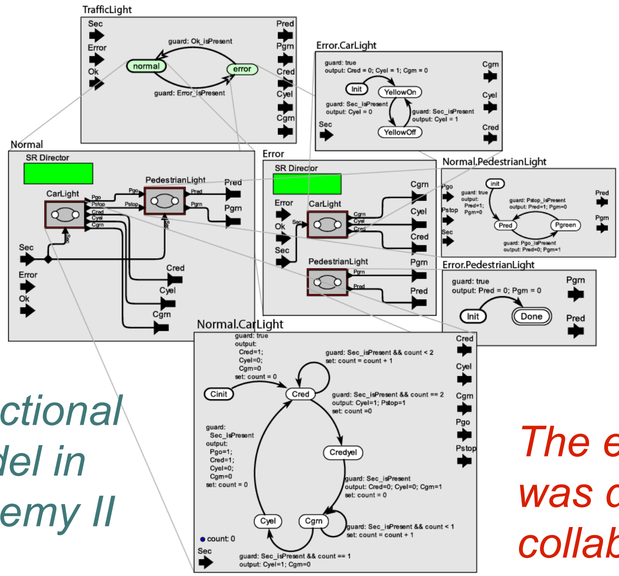
- **multi-view modeling:** distinct and separate models of the same system are constructed to model different aspects of the system.

Multi-View Modeling:

Distinct and separate models of the same system are constructed to model different aspects of the system.

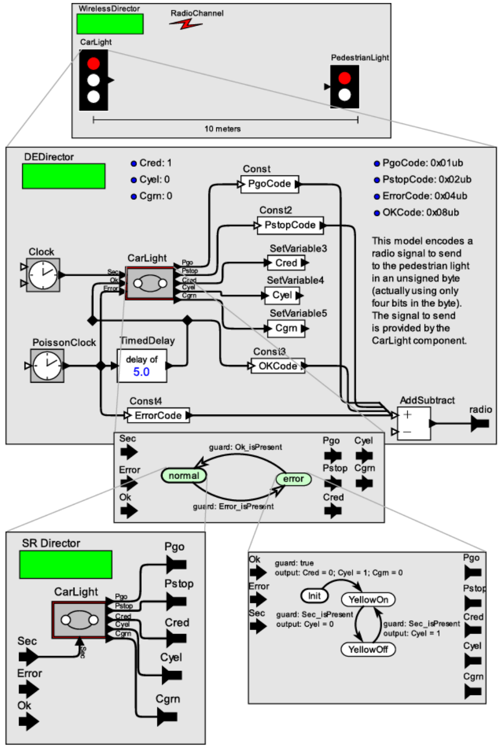


Functional model in Statecharts



Functional model in Ptolemy II

The example here was developed in a collaborative project with Lockheed-Martin.



Deployment model in Ptolemy II

```
MODULE CarLightNormal( Sec_isPresent )
    VAR
        state : {Cyel,Credyel,Cred,Cinit,Cgrn};
        count : { ls,0,1,2,gt };
    ASSIGN
        init(state) := Cinit;
        next(state) :=
            case
                state=Cinit & count=ls :{ Cred };
                ...
                Sec_isPresent & state=Cred
                              & count=ls :{ Cred };
                ...
                1             : state;
            esac;
        init(count) := 0;
        next(count) :=
            case
                state=Cinit & count=ls :{ 0 };
                ...
                Sec_isPresent & state=Cred & count=ls :{ ls };
                ...
                1             : count;
            esac;
    DEFINE
        Pstop_isPresent :=  ( Sec_isPresent
                              & state=Cred & count=2 ) ;
        Cred_isPresent :=  ...
        Cgrn_isPresent :=  ...
        Pgo_isPresent :=  ...
        Cyel_isPresent :=  ...
 MODULE PedestrianLightNormal( Pstop_isPresent,Pgo_isPresent )
    VAR
        state : {Pinit,Pgreen,Pred};
    ASSIGN
        init(state) := Pinit;
        next(state) :=
            case
                state=Pinit :{ Pred };
                Pgo_isPresent & state=Pred :{ Pgreen };
                Pstop_isPresent & state=Pgreen :{ Pgreen };
                1             : state;
            esac;
    DEFINE
        Pred_isPresent :=  ...
        Pgrn_isPresent :=  ...
 MODULE main
    VAR
        CarLightNormal: CarLightNormal( 1);
        PedestrianLightNormal: PedestrianLightNormal(
          CarLightNormal.Pstop_isPresent,
          CarLightNormal.Pgo_isPresent );
    SPEC
        ! EF (CarLightNormal.state = Cgrn
             & PedestrianLightNormal.state = Pgreen)
```

Verification model in SMV

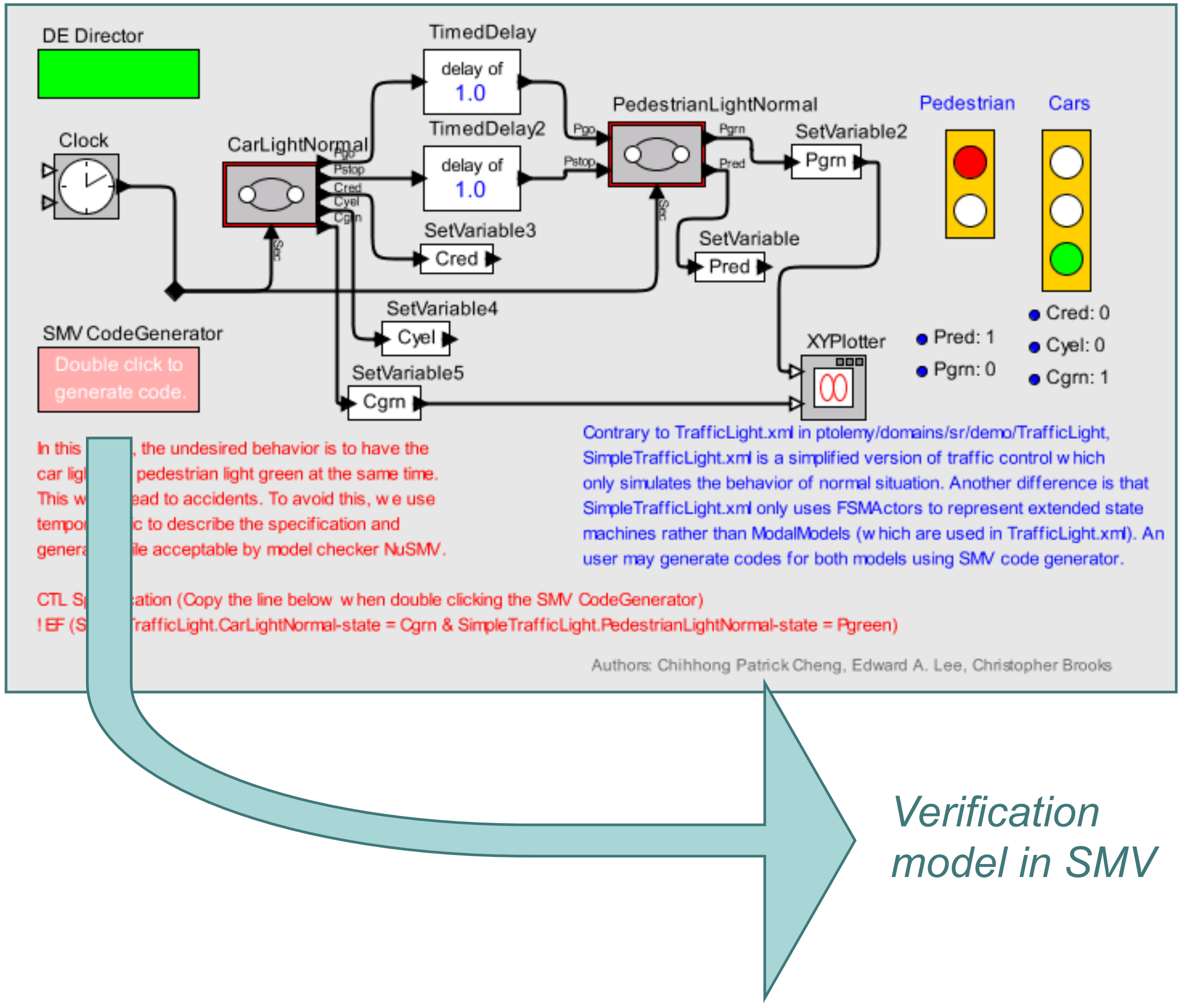
2	Project Name: Process-Based Software Components for Embedded Systems	Current Official Milestone Completinon Date	Does your project support this milestone? (Y or N)	Identify your s support(e
7	4. Demonstrate ability of propagating constraints among views	2QFY02	Y	Ptolemy II - hierar
8	5. Demonstrate ability to integrate different models of concurrency	2QFY02	Y	Ptolemy II - multi-
9	6. Demonstrate ability to integrate domain specific modeling tools	2QFY02	Y	Ptolemy II - multi-
10	7. Demonstrate ability to compose multiple view models	4QFY02	Y	Ptolemy II - multi-
11	8. Demonstrate ability to verify multiple-view models	4QFY03	Y	Ptolemy II - simul
12	Task 2: Model-Based Generation Technology			
13	1. Demonstrate ability to mathematically model generators	4QFY01	N	

Reliability model in Excel

Background on Multi-View Modeling

- Ptolemy Classic [Buck, Ha, Lee, Messerschmitt 94]
- UML [Various, 90s]
- Model-integrated computing [Sztipanovits, Karsai, Franke 96]
- SyncCharts [André 96]
- *Charts [Girault, Lee, Lee 99]
- Colif [Cesario, Nicolescu, Guathier, Lyonnard, Jerraya 01]
- Metropolis [Goessler, Sangiovanni-Vincentelli 02]
- KIEL [Prochnow, von Hanxleden 07]

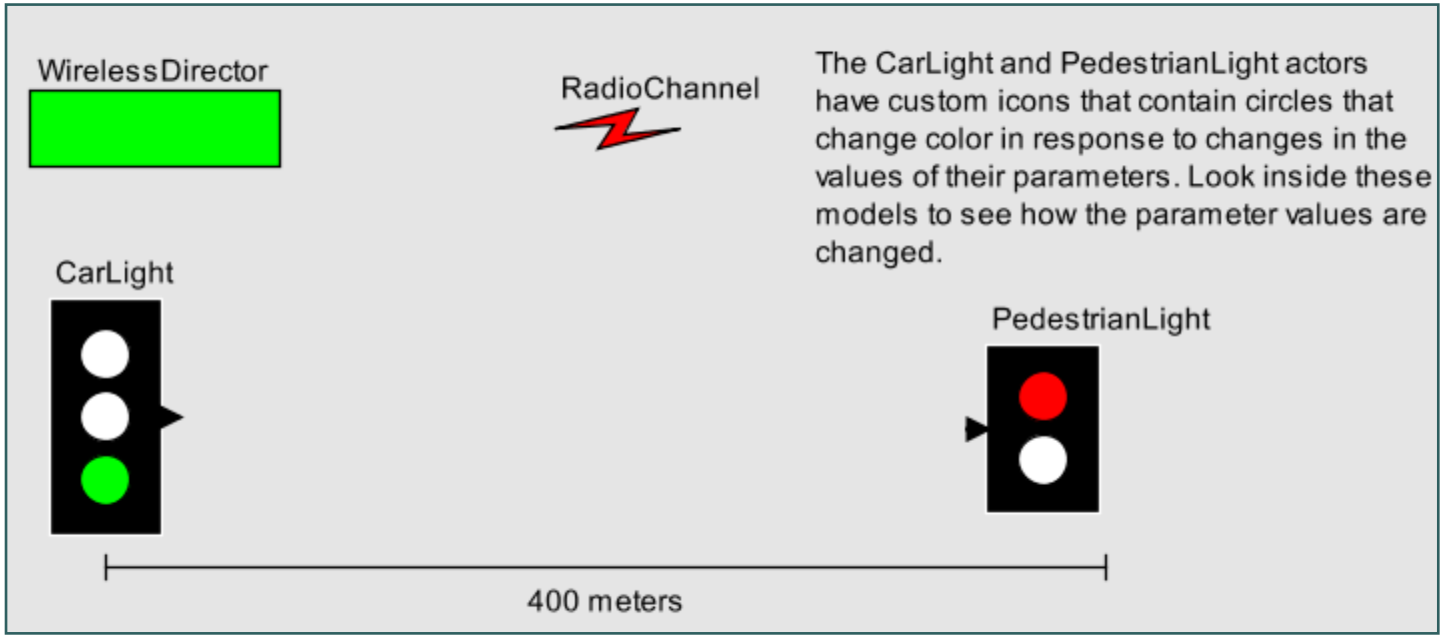
Model synthesis is one way to maintain model consistency



Verification model in SMV

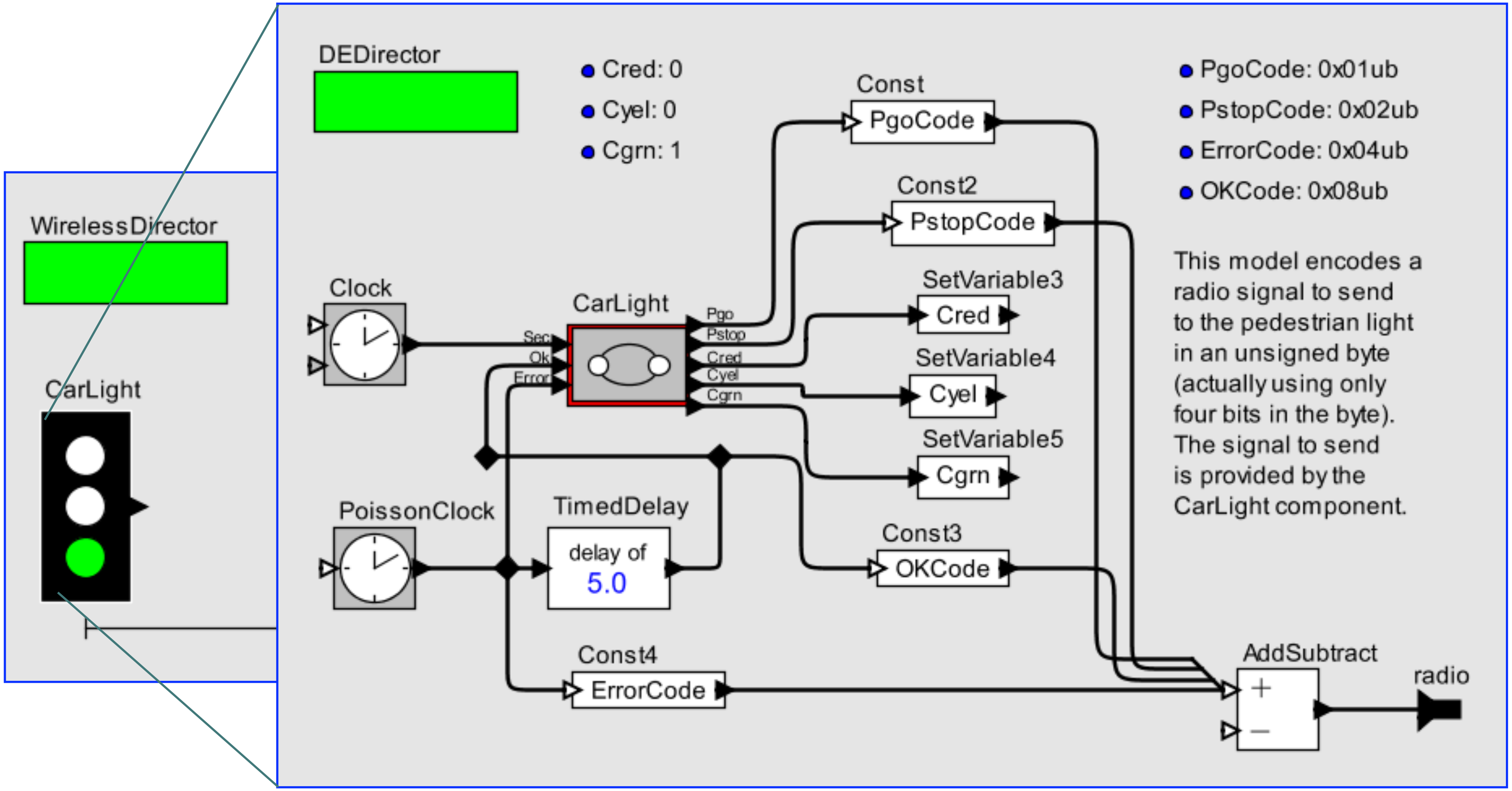
```
MODULE CarLightNormal( Sec_isPresent )
    VAR
        state : {Cyel,Credyel,Cred,Cinit,Cgrn};
        count : { ls,0,1,2,gt };
    ASSIGN
        init(state) := Cinit;
        next(state) :=
            case
                state=Cinit & count=ls :{ Cred };
                ...
                Sec_isPresent & state=Cred
                              & count=ls :{ Cred };
                ...
                1             : state;
            esac;
        init(count) := 0;
        next(count) :=
            case
                state=Cinit & count=ls :{ 0 };
                ...
                Sec_isPresent & state=Cred & count=ls :{ ls };
                ...
                1             : count;
            esac;
    DEFINE
        Pstop_isPresent :=  ( Sec_isPresent
                              & state=Cred & count=2 ) ;
        Cred_isPresent :=  ...
        Cgrn_isPresent :=  ...
        Pgo_isPresent :=  ...
        Cyel_isPresent :=  ...
 MODULE PedestrianLightNormal( Pstop_isPresent,Pgo_isPresent )
    VAR
        state : {Pinit,Pgreen,Pred};
    ASSIGN
        init(state) := Pinit;
        next(state) :=
            case
                state=Pinit :{ Pred };
                Pgo_isPresent & state=Pred :{ Pgreen };
                Pstop_isPresent & state=Pgreen :{ Pgreen };
                1             : state;
            esac;
    DEFINE
        Pred_isPresent :=  ...
        Pgrn_isPresent :=  ...
 MODULE main
    VAR
        CarLightNormal: CarLightNormal( 1);
        PedestrianLightNormal: PedestrianLightNormal(
          CarLightNormal.Pstop_isPresent,
          CarLightNormal.Pgo_isPresent );
    SPEC
        ! EF (CarLightNormal.state = Cgrn
            & PedestrianLightNormal.state = Pgreen)
```

But Model Synthesis is not always possible. Constructing a Deployment Model



This is the top level of a deployment model, which maps the car light and pedestrian light logic into two distinct compute platforms that communicate via a wireless link. The same models are used for the functional logic, leveraging actor-oriented classes in Ptolemy II.

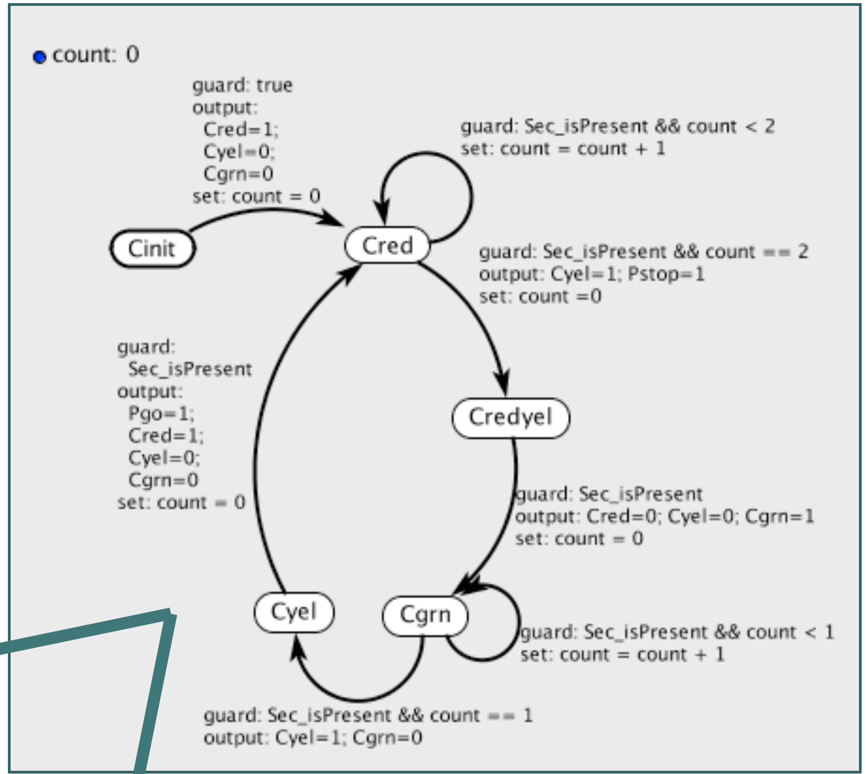
Inside The Car Light Model

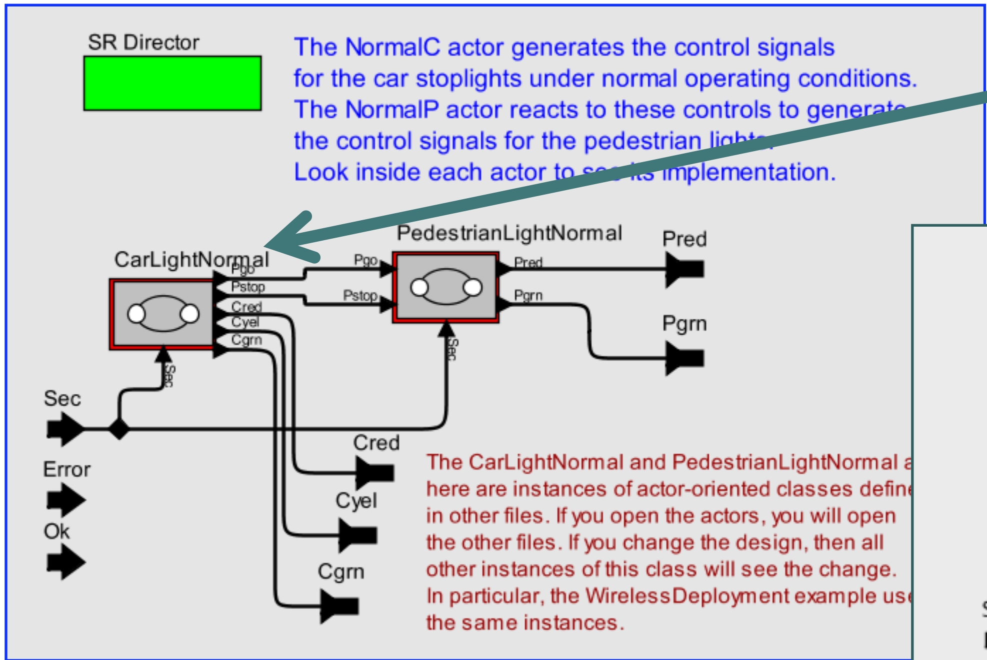


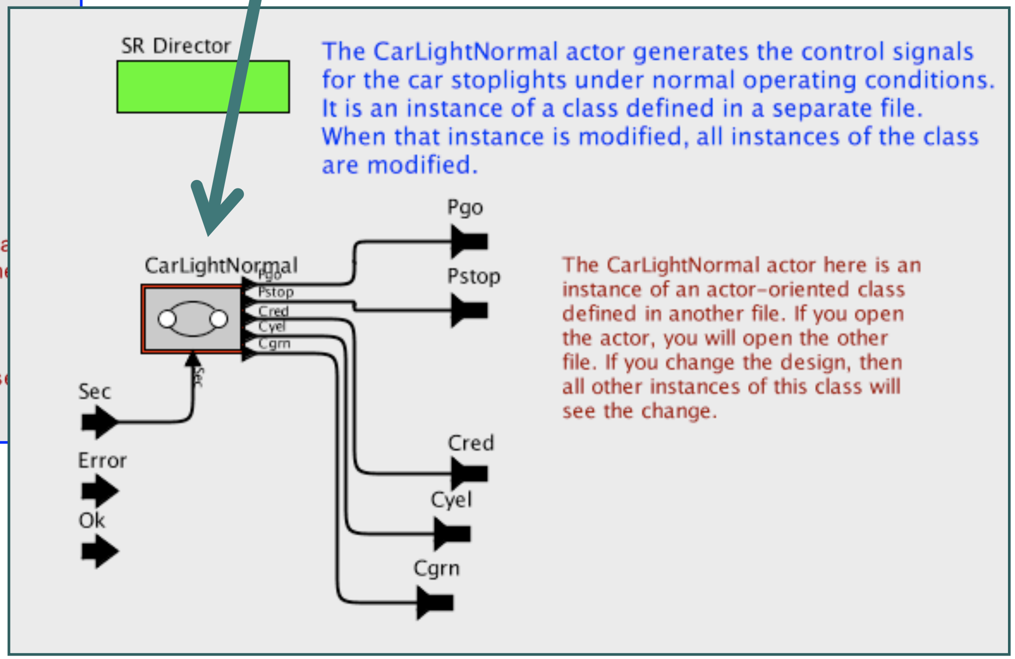
The above model shows the construction of a radio packet for transmission on the wireless link. Inside, it eventually uses the same behavioral model of the traffic light, so changing the behavior in one model is automatically reflected in the other.

Actor-Oriented Classes
[Lee, Liu, Neuendorffer 07]

A class definition (right) has instances in multiple models. Changes to the class definition automatically propagate to the instances.







In the functional model above, an instance communicates directly with the pedestrian light. The deployment model (right) constructs a radio packet and models wireless communication.

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.

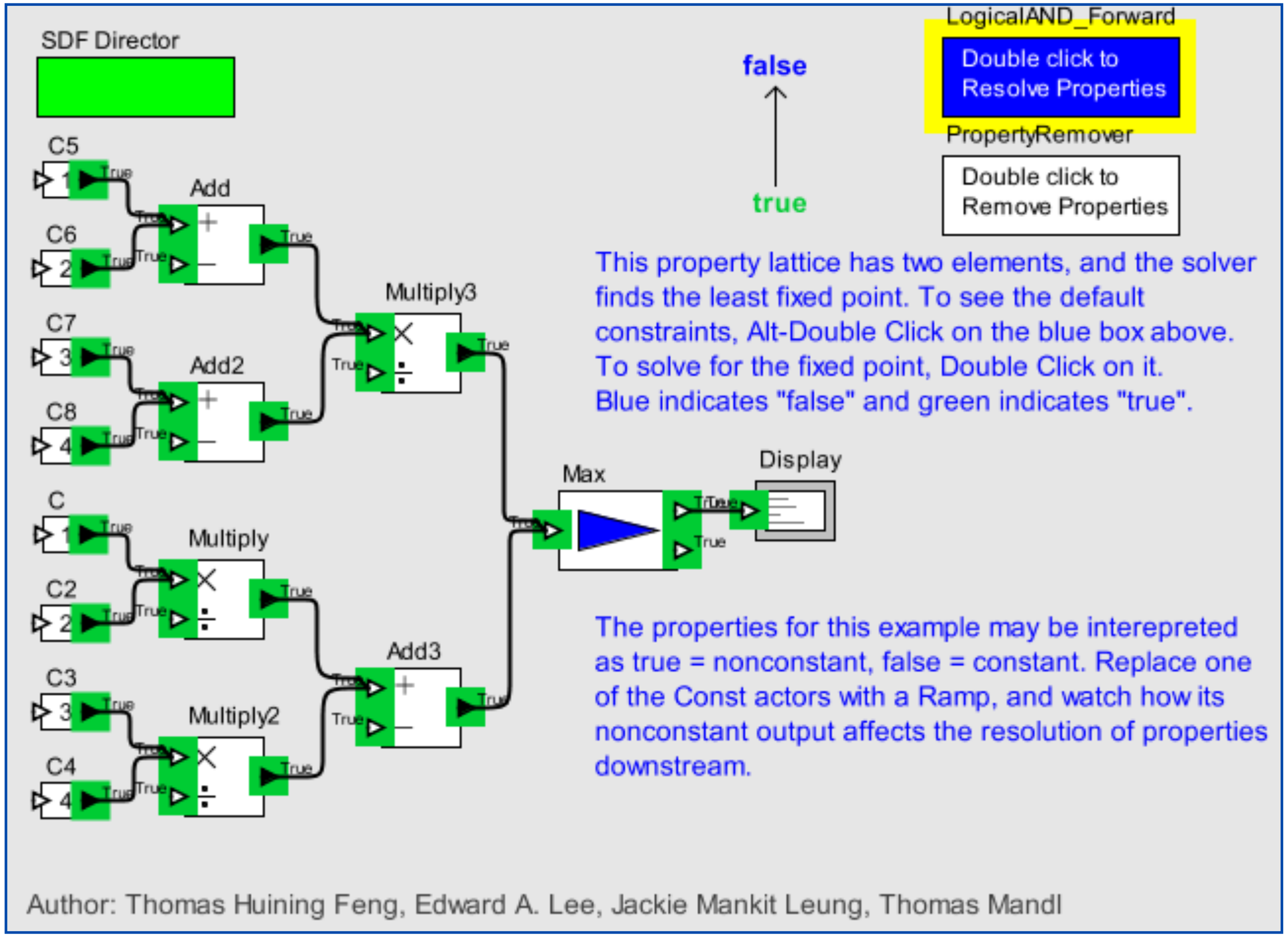
- **multi-specialization:** a single model is used to synthesize multiple distinct specialized models.

To support multi-specialization, we have built an extensible type system for model ontologies that performs "property inference".

In the system at the right, green indicates that a port is inferred or declared to be "constant"

Thanks to Thomas Mandl, Research & Technology Center, Bosch, Palo Alto.

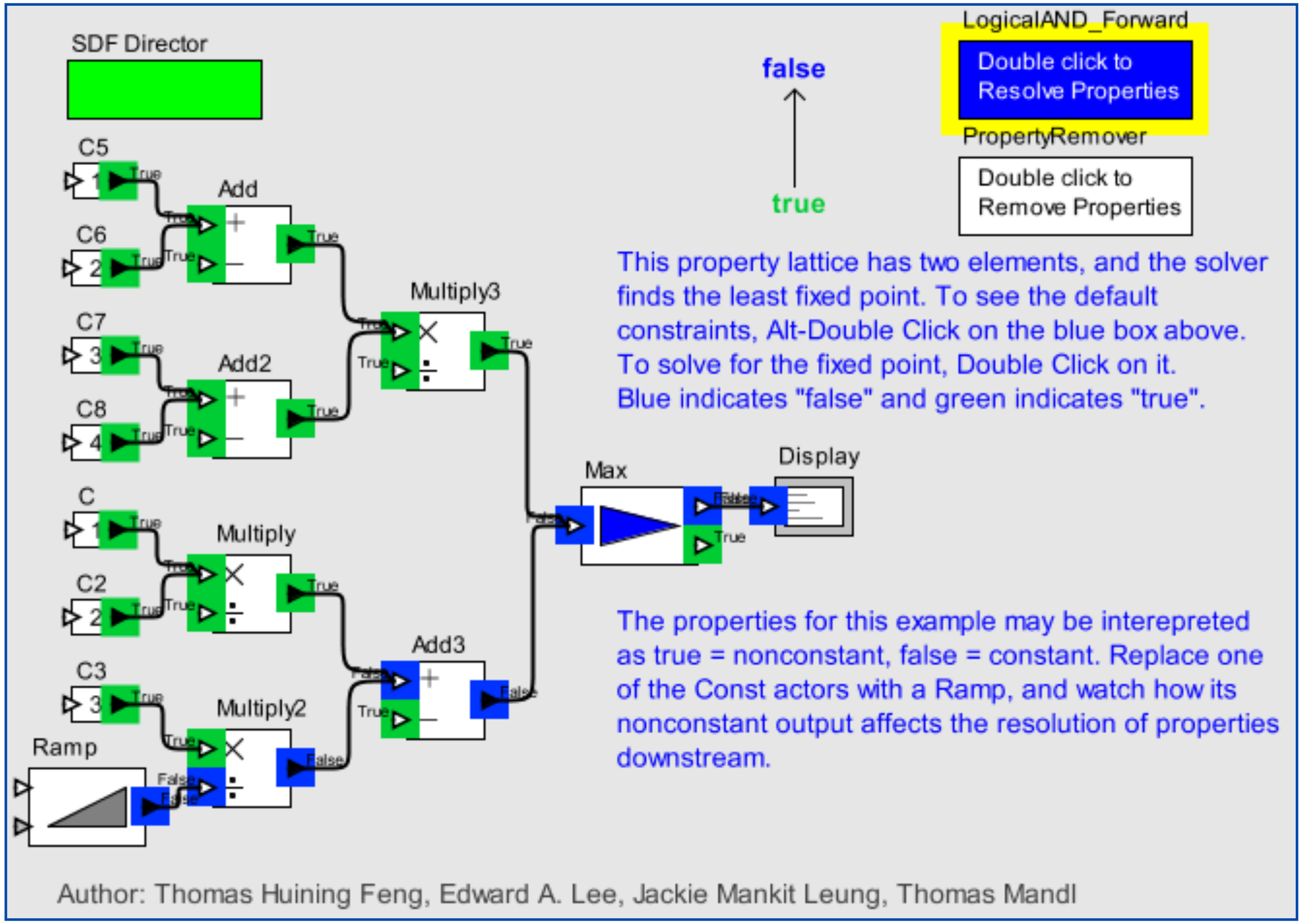
BOSCH



Demo: Model Properties as a Type Inference Problem

In the system at the right, one of the constant sources has been replaced with a non-constant source. This affects the inferred properties downstream.

Thanks to Thomas Mandl, Research & Technology Center, Bosch, Palo Alto.



BOSCH

Conclusion

- Multimodeling takes distinct forms.
- An abstract semantics can support this rigorously
  - This is not the same as just being noncommittal about the semantics!
- Tool support still needs a lot of work…

Syntax Comparisons

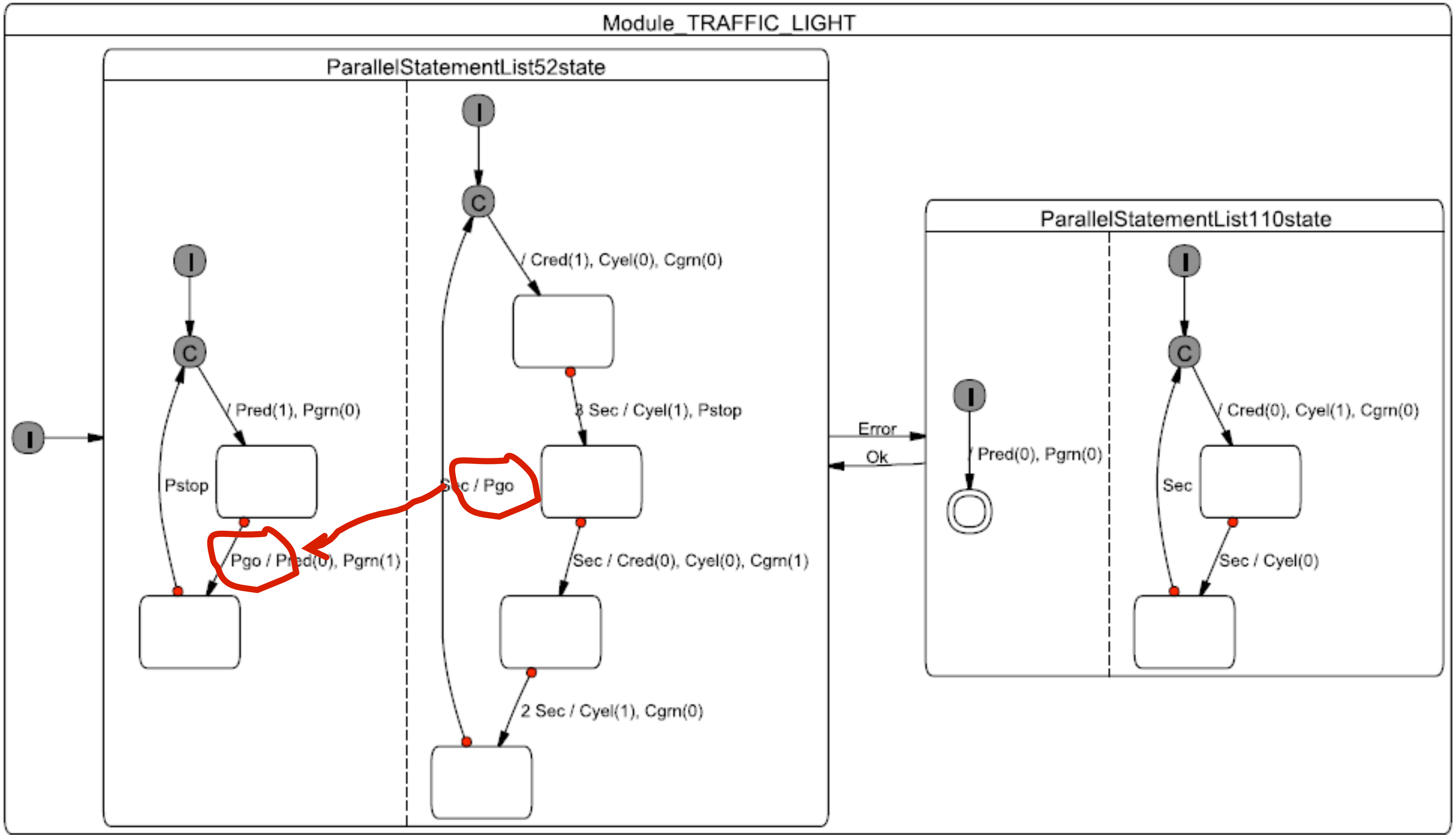
The Ptolemy II model and the Statecharts model differ in syntax. Some issues to consider when evaluating a syntax:

- Rendering on a page
- Showing dependencies in concurrent models
- Scalability to complex models
- Reusability (e.g. with other concurrency models)
- Special notations (e.g. “3 Sec”).

Simple Traffic Light Example in Statecharts, from Reinhard von Hanxleden, Kiel University

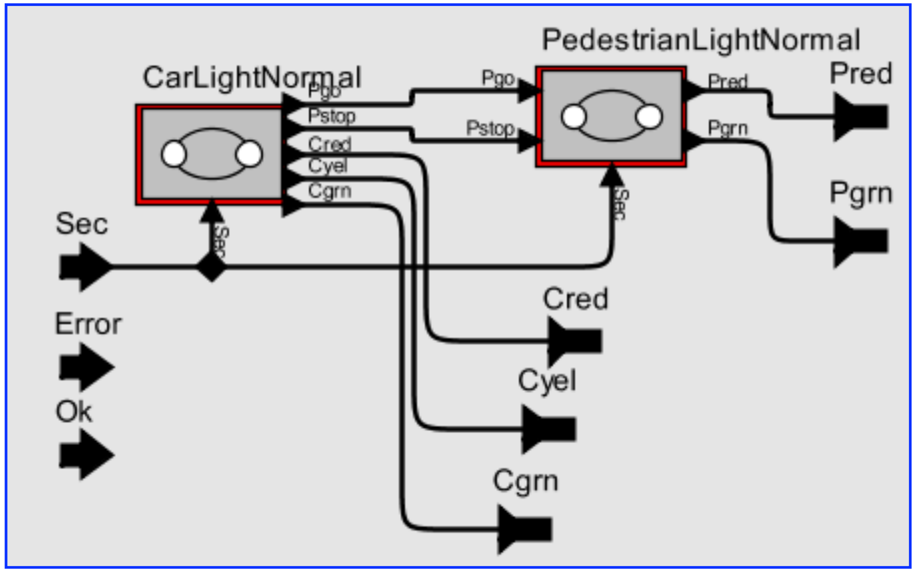
Case study for Ptolemy II Design

In StateCharts, the communication between concurrent components is not represented graphically, but is rather represented by name matching. Can you tell whether there is feedback?



Syntax comparisons

Now can you tell whether there is feedback?



Semantics Comparisons

The Ptolemy II model and the Statecharts model have similar semantics, but combined in different ways. Some issues to consider:

- Separation of concurrency from state machines
- Nesting of distinct models of computation
- Expanding beyond synchronous + FSM to model the (stochastic) environment and deployment to hardware.
- Styles of synchronous semantics (Ptolemy II realizes a true fixed-point constructive semantics).