

# Agenda

---

- 8:00-8:30 Continental breakfast
- 8:30-9:00 Overview of Mobies Phase 1 effort  
(Edward A. Lee)
- 9:00-9:40 Introduction
  - Mixing models of computation (Jie Liu) 0:15
  - Ducted Fan Model (Steve Neuendorffer) 0:15
  - Code Generation Introduction  
(Steve Neuendorffer) 0:05
- 9:40-10:00 MoBIES Applications I
  - ETC model using Giotto (Haiyang Zheng) 0:15
  - Giotto code generation (Haiyang Zheng) 0:05
- 10:00-10:15 Break

# Agenda (Cont'd)

---

## 10:15-11:15 MoBIES Applications II

V2V model as a hybrid system (Xiaojun Liu) 0:15

Charon code generation (Haiyang Zheng) 0:05

Networked real-time systems (Win Williams) 0:10

Ptolemy/Matlab integration (Edward A. Lee) 0:05

Giotto/Simulink integration (Christoph Kirsch)

afternoon

## 11:15-12:00 Principles

Data types and behavioral types (Yuhong Xiong)

Interface theories (Tom Henzinger)

Summary (Edward A. Lee)

# Process-Based Software Components

Mobies Phase 1, UC Berkeley

Edward A. Lee (PI) and Tom Henzinger (Co-PI)

March 12, 2002

# Program Objectives

---

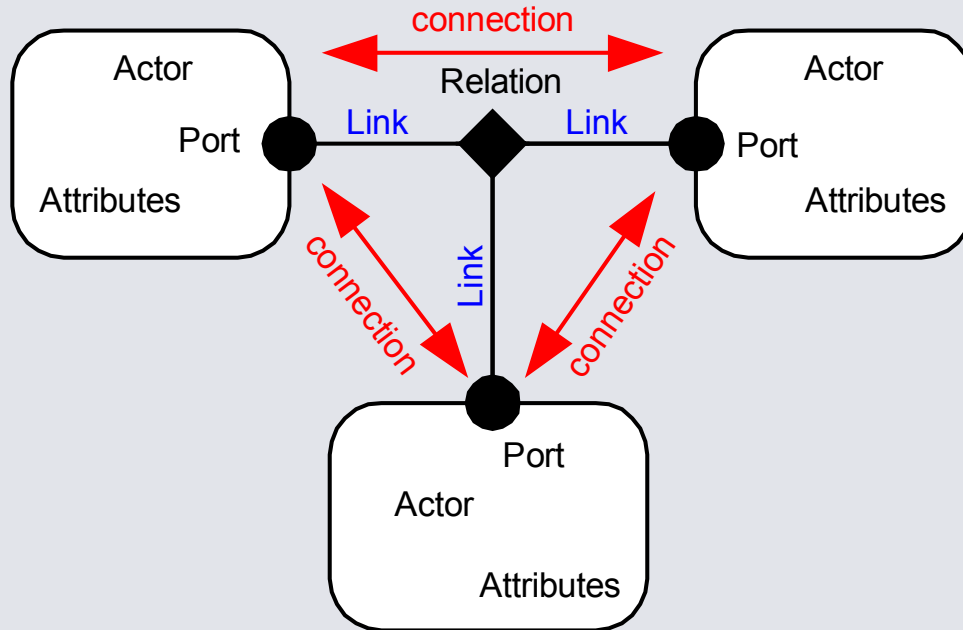
Our focus is on component-based design using principled *models of computation* and their *runtime environments* for embedded systems. The emphasis of this project is on the dynamics of the components, including the communication protocols that they use to interface with other components, the modeling of their state, and their flow of control. The purpose of the mechanisms we develop is to improve robustness and safety while promoting component-based design.

# Technical Approach Summary

---

- Models of computation
  - supporting heterogeneity
  - supporting real-time computation
  - codifications of design patterns
  - definition as *behavioral types*
- Co-compilation
  - joint compilation of components and architecture
  - vs. code generation
  - supporting heterogeneity

# View of Concurrent Components: *Actors with Ports and Attributes*



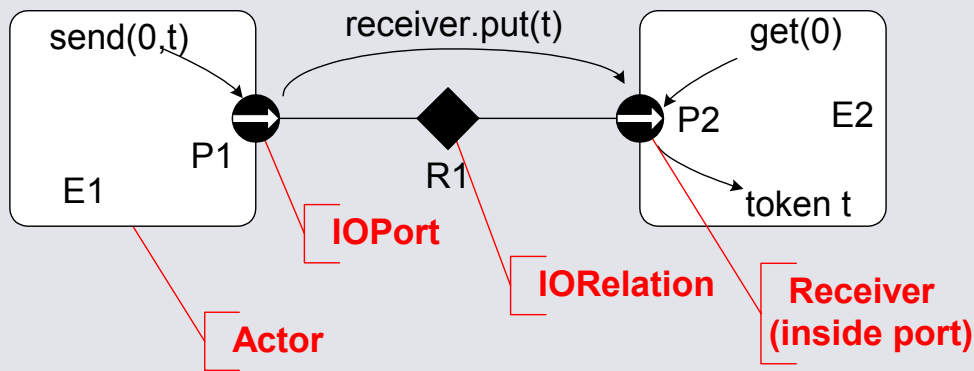
Model of Computation:

- Messaging schema
- Flow of control
- Concurrency

*Key idea:* The model of computation is part of the framework within which components are embedded not part of the components themselves. It enforces patterns.

# Actor View of Producer/Consumer Components

## Basic Transport:



## Models of Computation:

- continuous-time
- dataflow
- rendezvous
- discrete events
- synchronous
- time-driven
- publish/subscribe
- ...

# Examples of Actor-Oriented Component Frameworks

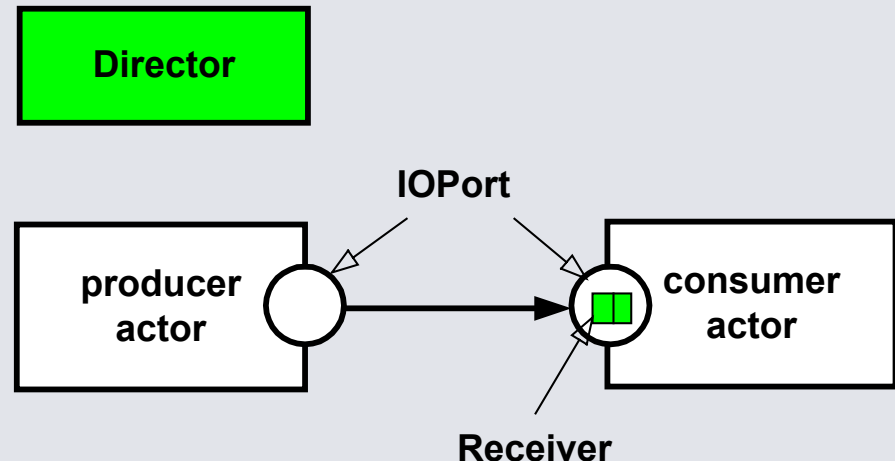
---

- Simulink (The MathWorks)
- Labview (National Instruments)
- 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)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- ...



# Ptolemy II Domains

- Define the flow(s) of control
  - "execution model"
  - Realized by a *Director* class
- Define communication between components
  - "interaction model"
  - Realized by a *Receiver* class



Task 1.3: Demonstrate ability to customize generic modeling tools

# Example Domains

---

- Time Driven (Giotto):
  - synchronous, time-driven multitasking - *built for Mobies.*
- Synchronous Data Flow (SDF):
  - stream-based communication, statically scheduled
- Discrete Event (DE):
  - event-based communication
- Continuous Time (CT):
  - continuous semantics, ODE solver simulation engine
- Synchronous/Reactive (SR):
  - synchronous, fixed point semantics
- Timed Multitasking (TM):
  - priority-driven multitasking, deterministic communication - *built for SEC.*
- Communicating Sequential Processes (CSP):
  - rendezvous-style communication
- Process Networks (PN):
  - asynchronous communication, determinism

# Design Pattern: Periodic/Time-Driven Inside Continuous Time

Giotto director indicates a new model of computation.

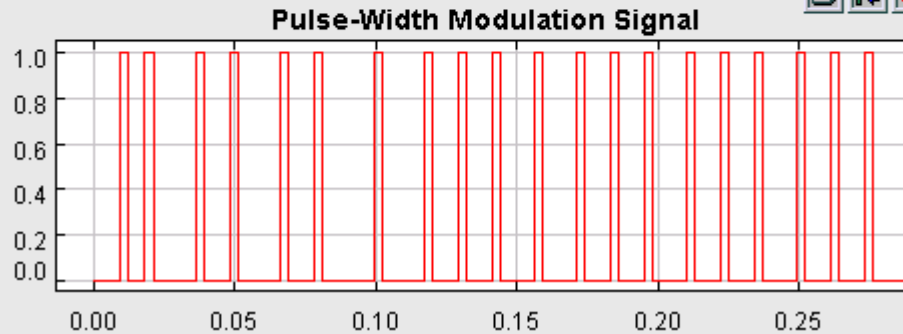
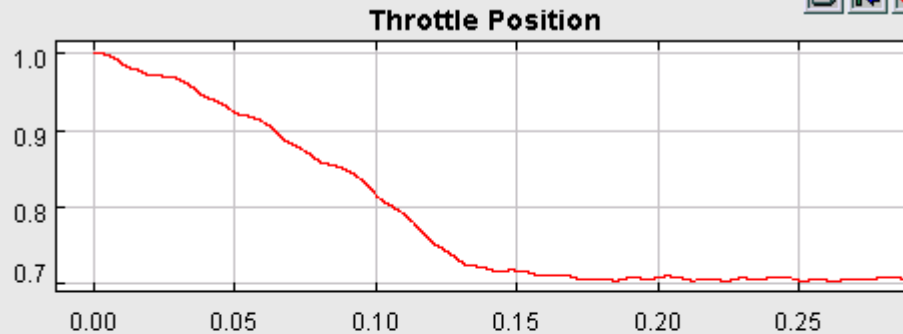
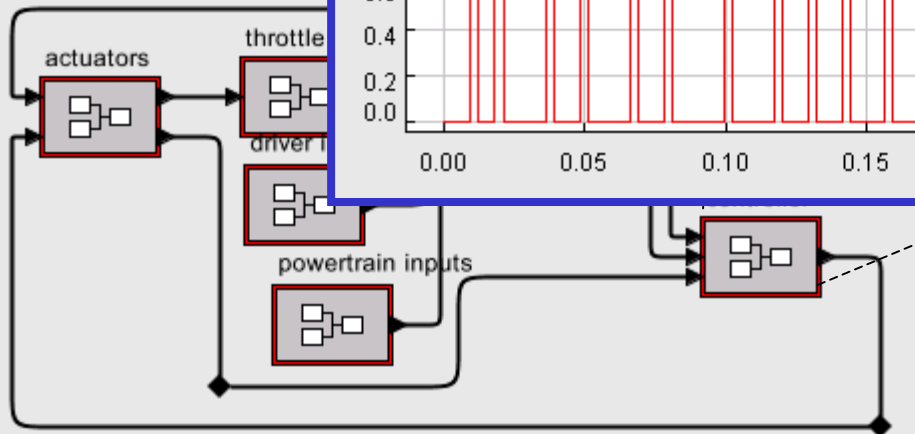
Giotto Director



Heterogeneous model of the Electronic Throttle Control

by Paul Griffiths, Christoph  
Last updated January 15, 2012

CT Director



ent.

Task 1.7: Demonstrate ability to compose multiple view models.

Task 1.1: Demonstrate ability of modeling cross cutting physical constraints.

# Controller Heterogeneity

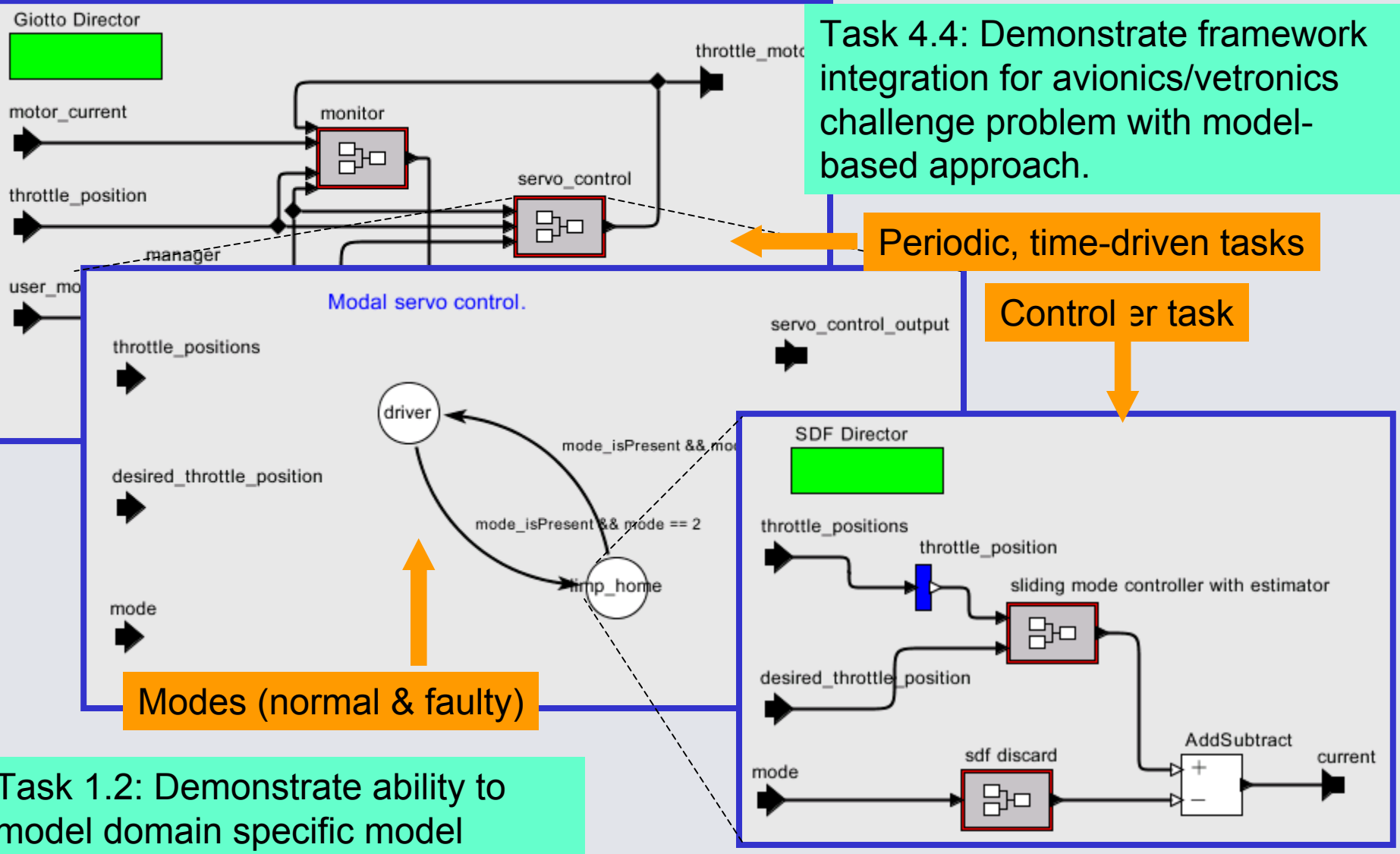
Task 4.4: Demonstrate framework integration for avionics/vetronics challenge problem with model-based approach.

Periodic, time-driven tasks

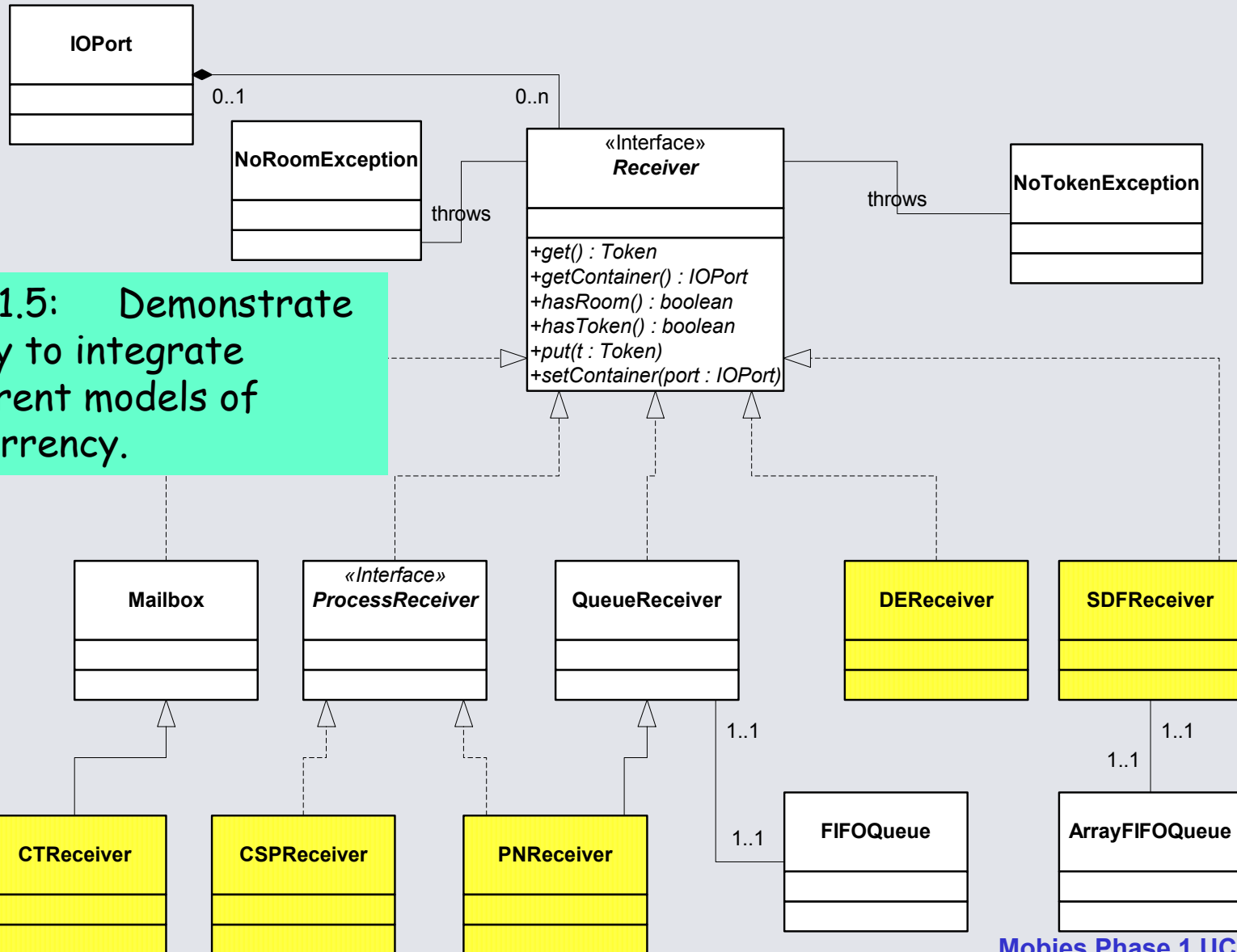
Control or task

Modes (normal & faulty)

Task 1.2: Demonstrate ability to model domain specific model semantics.

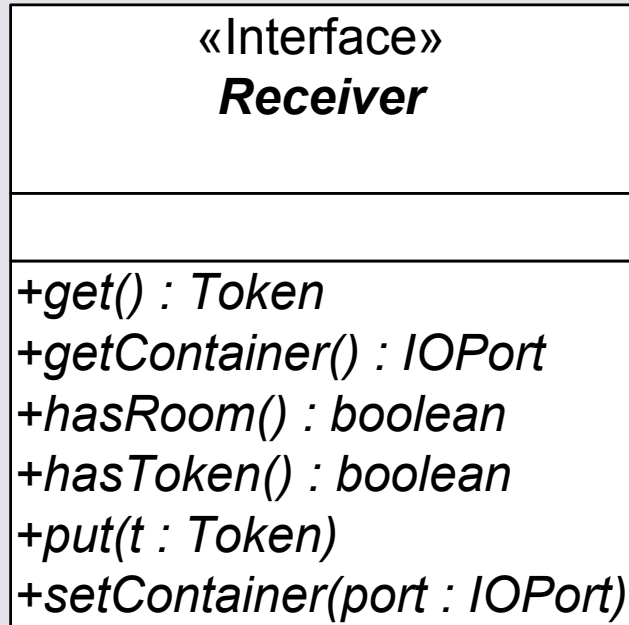


# Key to Domain Polymorphism: Receiver Object Model

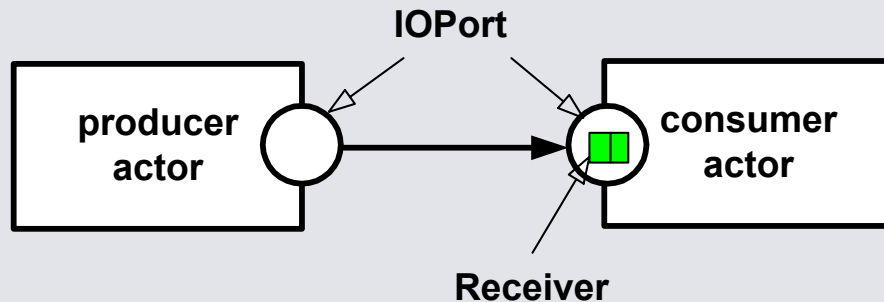


Task 1.5: Demonstrate ability to integrate different models of concurrency.

# Receiver Interface



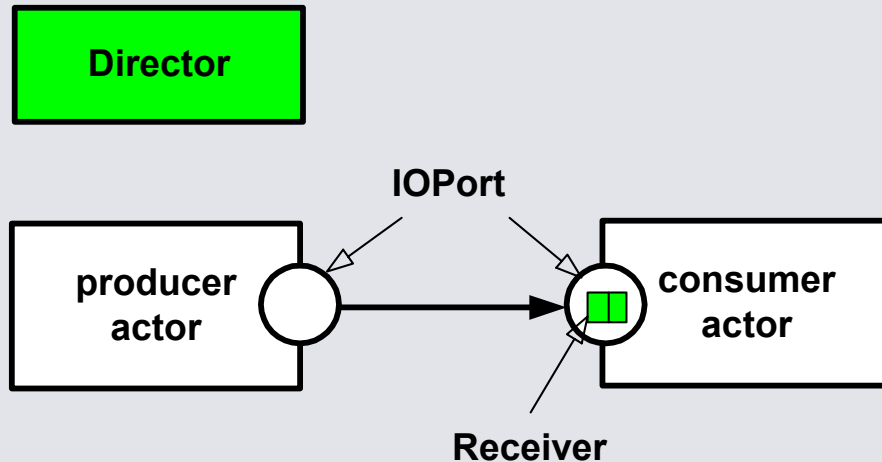
These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is supplied by the director, not by the component.



Task 3.1: Demonstrate ability to generate interface code to couple frameworks

# Behavioral Types - Codification of Domain Semantics

- Capture the dynamic interaction of components in *types*
- Obtain benefits analogous to data typing.
- Call the result *behavioral types*.



Task 3.3: Demonstrate ability to model framework constraints.

Task 3.4: Demonstrate ability to verify composability of frameworks.

- Communication has
  - data types
  - behavioral types
- Components have
  - data type signatures
  - domain type signatures
- Components are
  - data polymorphic
  - domain polymorphic

# Summary of Behavioral Types Results

---

- We capture patterns of component interaction in a type system framework: *behavioral types*
- We describe interaction types and component behavior using *interface automata*.
- We do type checking through *automata composition* (detect component incompatibilities)
- Subtyping order is given by the alternating simulation relation, supporting *polymorphism*.

To be presented later by Yuhong Xiong.



# Code Generation

- MoC semantics defines
  - flow of control across actors
  - communication protocols between actors
- Actors define:
  - functionality of components
- Actors are compiled by a MoC-aware compiler
  - generate specialized code for actors in context
- Hierarchy & heterogeneity:
  - Code generation at a level of the hierarchy produces a new actor definition

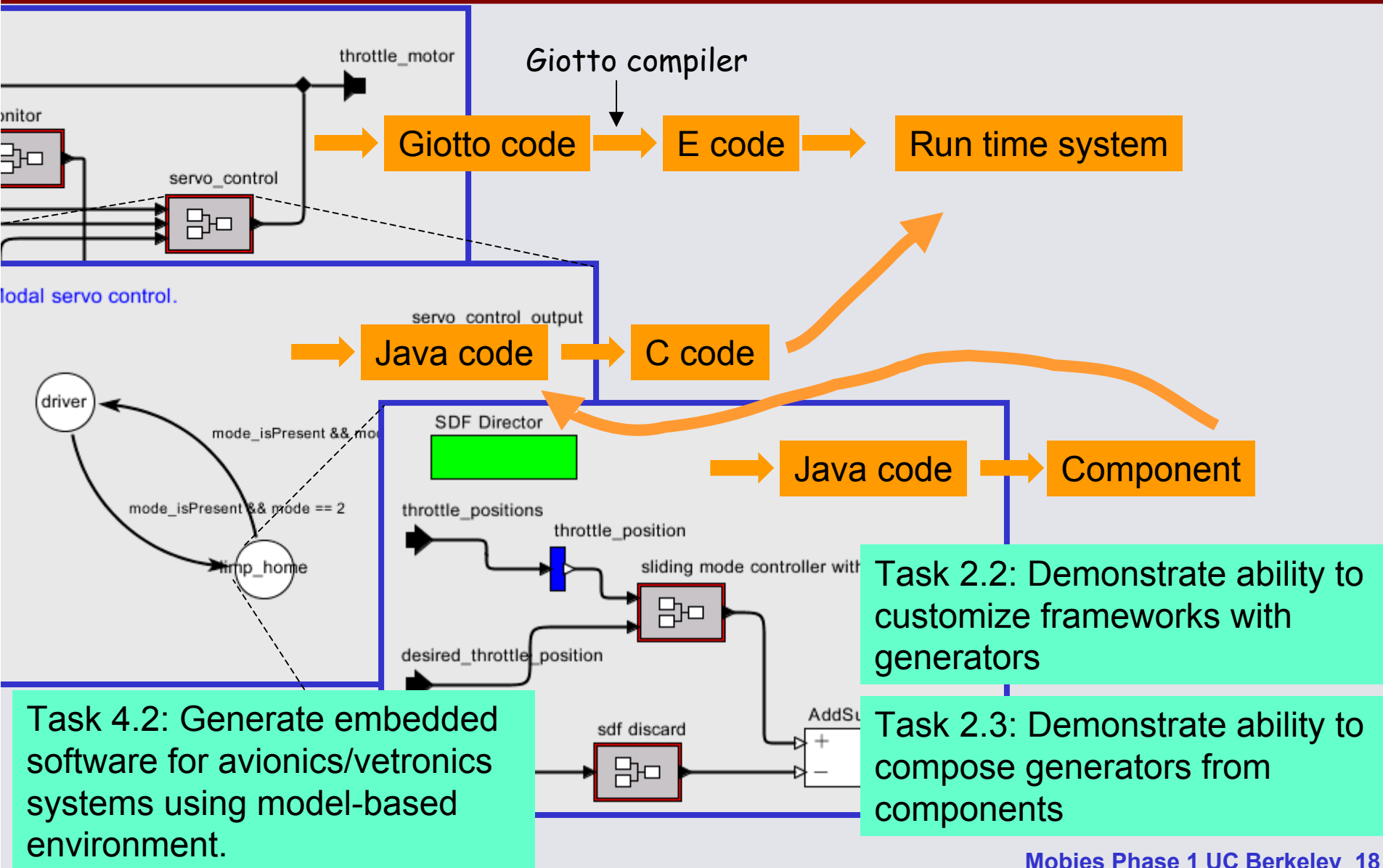
Task 2.4: Demonstrate ability to generate embedded software from models.

Task 4.3: Demonstrate superior performance of the synthesized system.

*We call this co-compilation.*

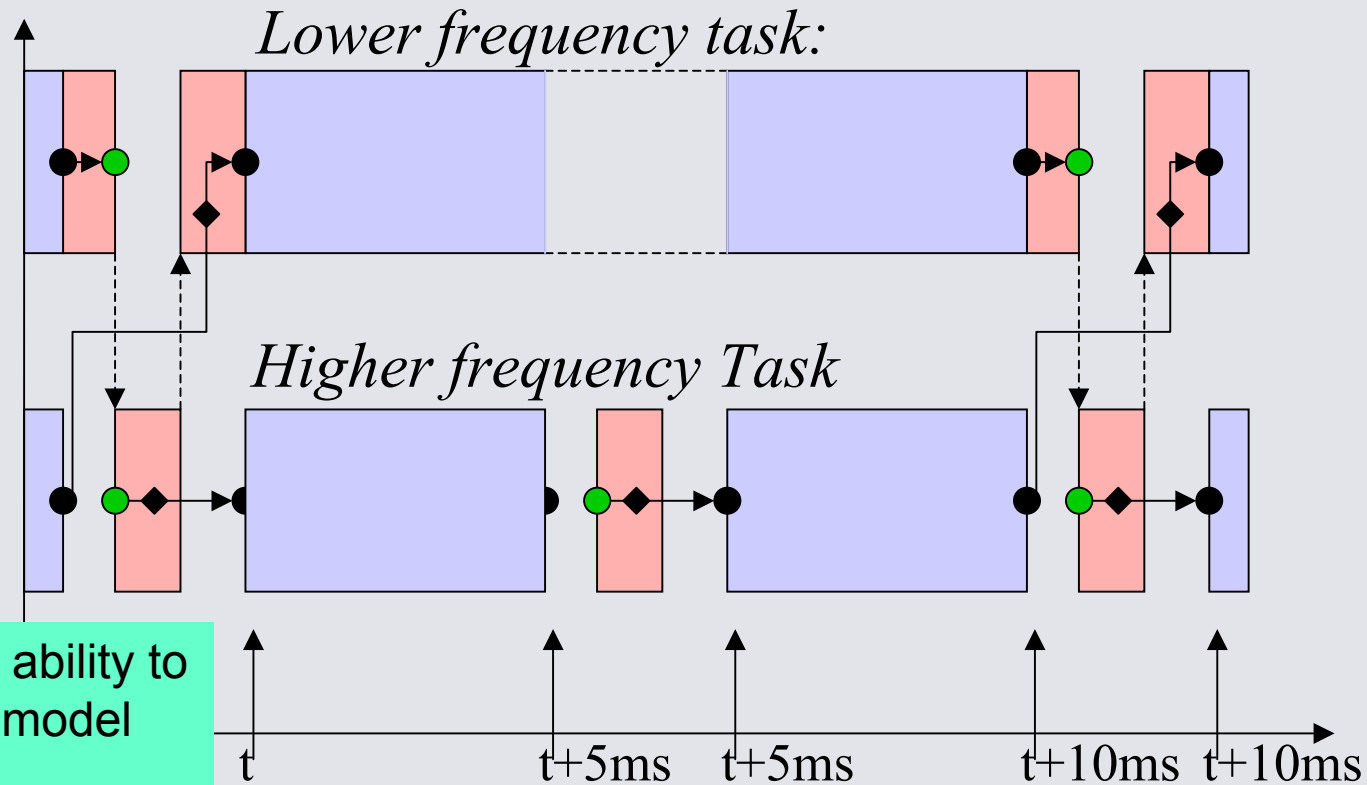
Multiple domains may be used in the same model

# Integrated Code Generation



# Giotto - Periodic Hard-Real-Time Tasks with Precise Mode Changes

Major part of the Mobies effort was to interface this domain to others: CT above, FSM below for modal modeling, and SDF for task definition.



Task 1.2: Demonstrate ability to model domain specific model semantics

- Giotto compiler targets the E Machine
- Giotto/Simulink integration
- Ptolemy II Giotto code generator

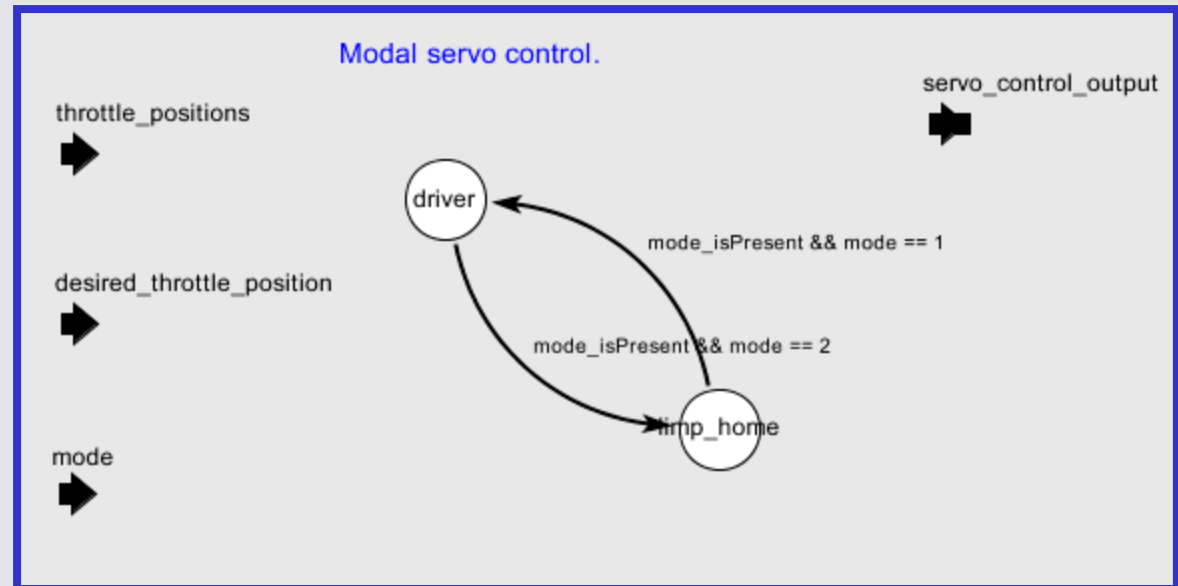
To be presented in the afternoon by Tom Henzinger and Christoph Kirsch.

# Modal Models - The FSM Domain

- Refines components in *any* domain
  - with CT, get hybrid systems
  - with Giotto, get on-line schedule customization
  - with SR, get statecharts semantics
  - with PN, get SDL-style semantics
  - ...

Task 1.3: Demonstrate ability to customize generic modeling tools

Design of Giotto domain was greatly simplified by leveraging the FSM domain. We improved the Giotto semantics by introducing modes with limited scope. We learned how to integrate Giotto with other MoCs.

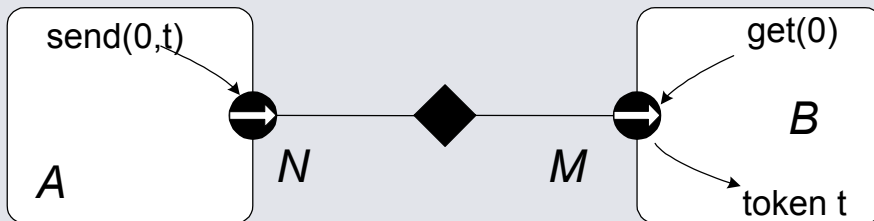


# Synchronous Dataflow (SDF) Preferred Domain for Task Definition

- Balance equations (one for each channel):

$$F_A N = F_B M$$

- Scheduled statically
- Decidable resource requirements



Domains like Giotto, TM, orchestrate large-grain components. The components themselves need not be designed at the low level in C. They can be designed using other Ptolemy II domains.

Available optimizations:

- eliminate checks for input data
- statically allocate communication buffers
- statically sequence actor invocations (and inline)

Task 4.3: Demonstrate superior performance of the synthesized system.

# Code Generation Objective

---

- It is not sufficient to build a mechanism for generating code from one, fixed, modeling environment.
- Modeling strategies must be nested hierarchically.
- Code generators have to be heterogeneously composable.

Task 2.3: Demonstrate ability to compose generators from components

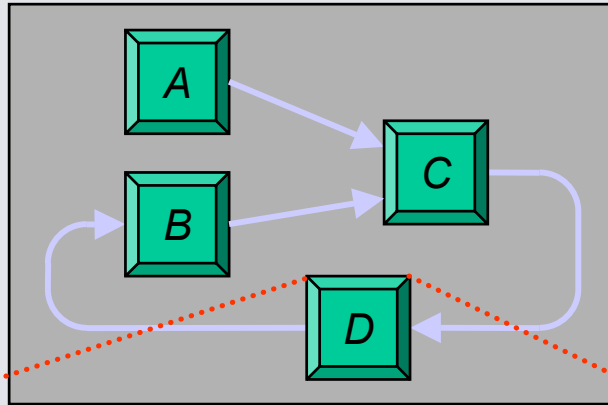
We aren't there yet, but we have a plan...

# Code Generation Status

---

- Giotto code generator from Giotto domain
  - still need code generation from FSM to get modal models
- Java code generator from SDF domain
  - based on Soot compiler infrastructure (McGill)
  - 80% of SDF test suite passes
  - type specialization
  - static scheduling, buffering
  - code substitution using model of computation semantics
- C code generation from Java
  - University of Maryland subcontract
  - based on Soot compiler infrastructure (McGill)
  - preliminary concept demonstration built
- Configurable hardware synthesis
  - targeted Wildcard as a concept demonstration
  - collaborative with BYU (funded by another program)

# Actor Code is the Component Spec



Code generate  
a domain-  
polymorphic  
component  
definition.

```
public TypedIOPort input;  
public TypedIOPort output;  
public Parameter constant;  
public void fire() {  
    Token t = input.get(0);  
    Token sum = t.add(constant.getToken());  
    output.send(0, t2);  
}
```



# Actor Definition: Cal

---

- Java is not the ideal actor definition language. Key meta-data is hard to extract:
  - token production/consumption patterns
  - firing rules (preconditions)
  - state management (e.g. recognize stateless actors)
  - type constraints must be explicitly given
  - modal behavior
- Defining an actor definition format (Cal):
  - enforce coding patterns
  - make meta-data available for code generation
  - infer behavioral types automatically
  - analyze domain compatibility
  - support multiple back-ends (C, C++, Java, Matlab)

# Summary of Accomplishments to Date

---

- Heterogeneous modeling
  - Domain polymorphism concept & realization
  - Behavioral type system
  - Giotto semantics & integration with other MoCs
  - Component definition principles (Caltrop)
- Code generation
  - Co-compilation concept
  - Giotto program generation
  - Java code generation from SDF
    - 80% of SDF test suite passes
  - C code generation from Java
    - Early phase, concept demonstration

# Plans

- Midterm experiment
  - ETC and V2V models and code generators
- Complete actor definition framework
  - define the meta-semantics for domain-polymorphic actors
- Behavioral types
  - reflection
  - real-time properties as dependent types
- Complete SDF code generation
  - token unboxing
  - elimination of memory management
  - 100% of test suite must pass
- Code generate Ptolemy II expressions
  - use of expression actor simplifies models
- Implement FSM code generation
  - support modal models
- Complete C code generation
  - support key subset of Java libraries
- Integrate heterogeneous code generators
  - systematize hierarchy support
  - define Java subset that generates well to C