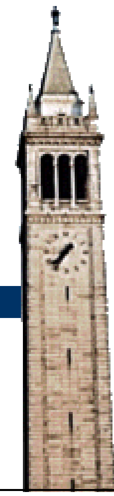
Concurrent Models of Computation in System Level Design

Forum on Design Languages

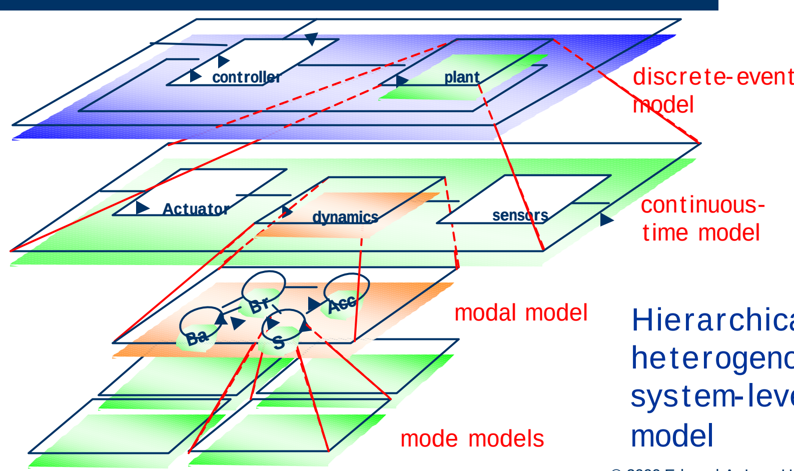
Workshop on System Specification & Design Languages

Edward Lee
UC Berkeley

September 4-8, 2000 - Tübingen, Germany



Components and Composition



Hierarchical, heterogenous, system-level model

Component Frameworks

- What is a component? (ontology)
  - States? Processes? Threads? Differential equations? Constraints? Objects (data + methods)?
- What knowledge do components share? (epistemology)
  - Time? Name spaces? Signals? State?
- How do components communicate? (protocols)
  - Rendezvous? Message passing? Continuous-time signals? Streams? Method calls? Events in time?
- What do components communicate? (lexicon)
  - Objects? Transfer of control? Data structures? ASCII text?

A Laboratory for Exploring Component Frameworks

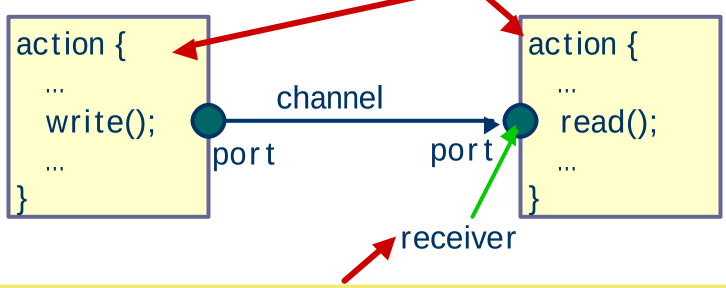
Ptolemy II –
- Java based, network integrated
- Several frameworks implemented

A realization of a framework is called a "domain." Multiple domains can be mixed hierarchically in the same model.

http://ptolemy.eecs.berkeley.edu

A Class of Concurrent Frameworks: Producer / Consumer

Are actors active? passive? reactive?
Flow of control is mediated by a *director*.



Are communications timed? synchronized? buffered?
Communications are mediated by *receivers*.

Domain – Realization of a Component Framework

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DT – discrete time (cycle driven)
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive

Each is realized as a director and a receiver class

Each of these defines a component ontology and an interaction semantics between components. There are many more possibilities!

1. Continuous Time (Coupled ODEs)

Semantics:

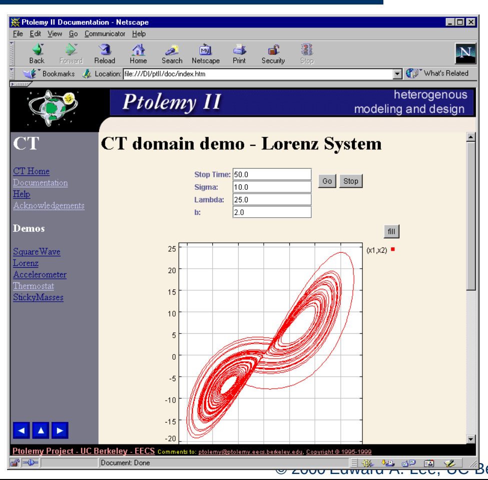
- actors define relations between functions of time (ODEs or algebraic equations)
- a behavior is a set of signals satisfying these relations

Examples:

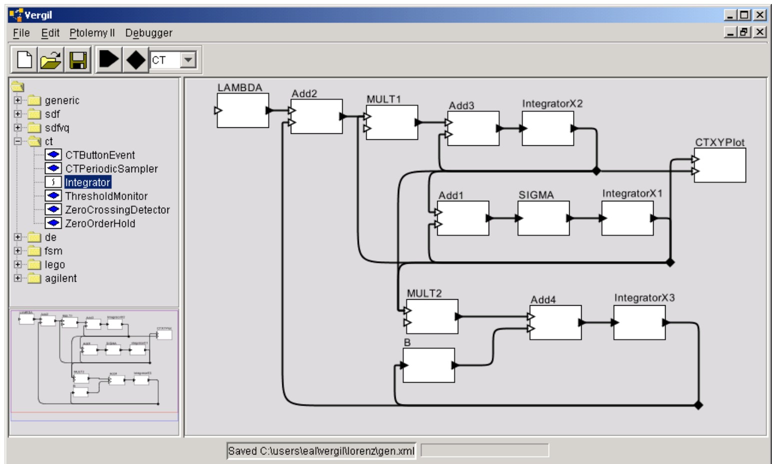
- Spice,
- HP ADS,
- Simulink,
- Saber,
- Matrix X,
- ...

1. Continuous Time in Ptolemy II

The continuous time (CT) domain in Ptolemy II models components interacting by continuous-time signals. A variable-step size, Runge-Kutta ODE solver is used, augmented with discrete-event management (via modeling of Dirac delta functions).



1. CT Block Diagram



1. CT: Strengths and Weaknesses

Strengths:

- Accurate model for many physical systems
- Determinate under simple conditions
- Established and mature (approximate) simulation techniques

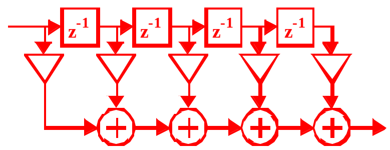
Weaknesses:

- Covers a narrow application domain
- Tightly bound to an implementation
- Relatively expensive to simulate
- Difficult to implement in software

2. Discrete Time

Semantics:

- blocks are relations between functions of discrete time (difference equations)
- a behavior is a set of signals satisfying these relations



Examples:

- System C
- HP Ptolemy,
- SystemView,
- ...

2. DT: Strengths and Weaknesses

Strengths:

- Useful model for embedded DSP
- Determinate under simple conditions
- Easy simulation (cycle-based)
- Easy implementation (circuits or software)

Weaknesses:

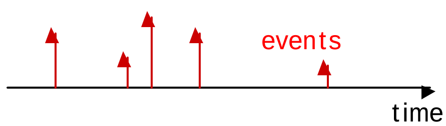
- Covers a narrow application domain
- Global synchrony may overspecify some systems

3. Discrete Events

Semantics:

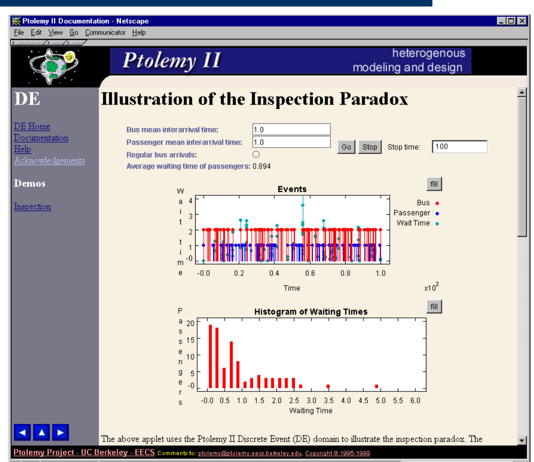
- Events occur at discrete points on a time line that is often a continuum. The components react to events in chronological order.

Examples:
- SES Workbench,
- Bones,
- VHDL
- Verilog
- ...



3. Discrete-Events in Ptolemy II

The discrete-event (DE) domain in Ptolemy II models components interacting by discrete events placed in time. A calendar queue scheduler is used for efficient event management, and simultaneous events are handled systematically and deterministically.



© 2000 Edward A. Lee, UC Berkeley

3. DE: Strengths and Weaknesses

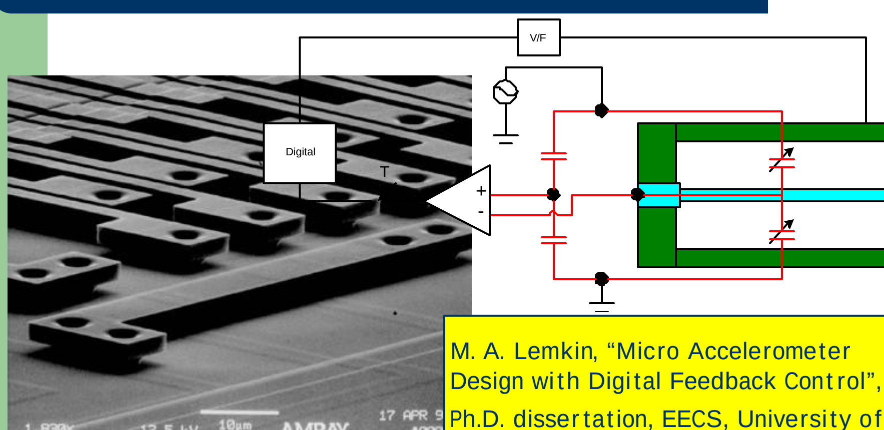
Strengths:

- Natural for asynchronous digital hardware
- Global synchronization
- Determinate under simple conditions
- Simulatable under simple conditions

Weaknesses:

- Expensive to implement in software
- May over-specify and/or over-model systems

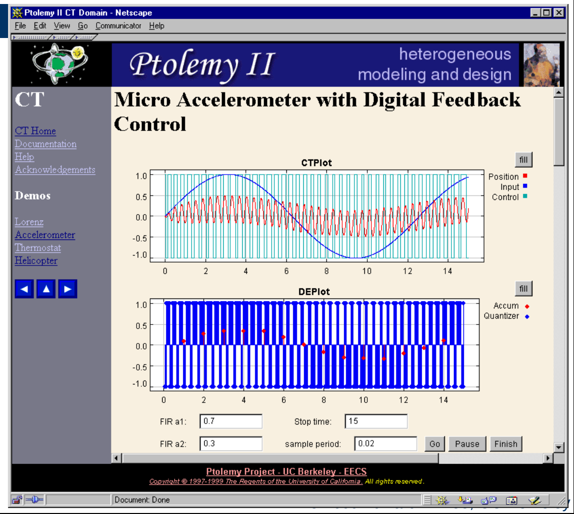
Mixing Domains
Example: MEMS Accelerometer



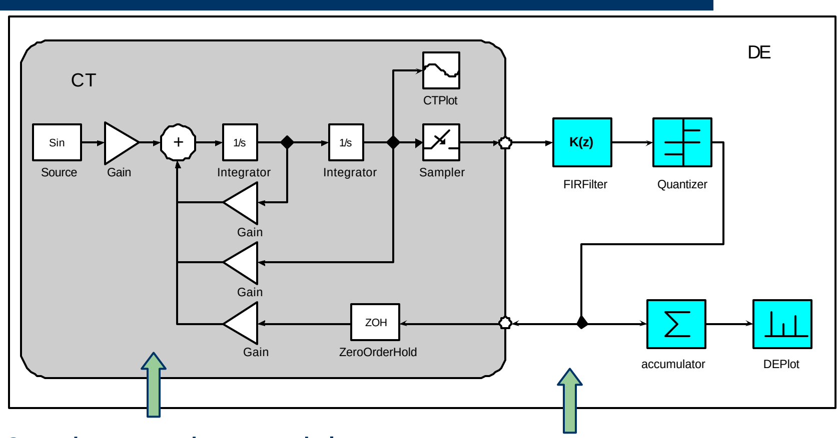
M. A. Lemkin, “Micro Accelerometer Design with Digital Feedback Control”, Ph.D. dissertation, EECS, University of California, Berkeley, Fall 1997

Accelerometer Applet

This model mixes two Ptolemy II domains, DE (discrete events) and CT (continuous time).



Hierarchical Heterogeneous Models

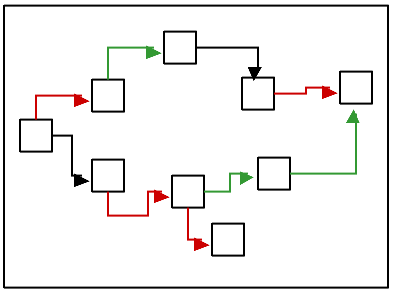


Continuous-time model

Discrete-event model

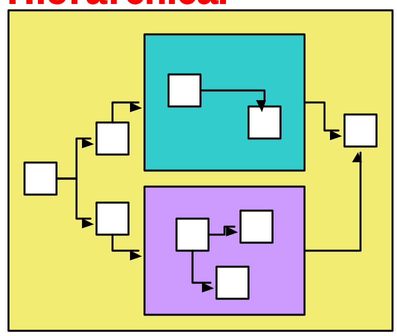
Hierarchical Heterogeneity vs. Amorphous Heterogeneity

Amorphous



Color is a communication protocol only, which interacts in unpredictable ways with the flow of control.

Hierarchical

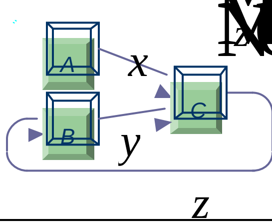


Color is a domain, which defines both the flow of control and interaction protocols.

4. Synchronous/Reactive Models

- A discrete model of time progresses as a sequence of "ticks." At a tick, the signals are defined by a fixed point equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_{A,t}(1) \\ f_{B,t}(z) \\ f_{C,t}(x, y) \end{bmatrix}$$



Examples:
- Esterel,
- Lustre,
- Signal,
- Argos,
- ...

4. SR: Strengths and Weaknesses

Strengths:

- Good match for control-intensive systems
- Tightly synchronized
- Determinate in most cases
- Maps well to hardware and software

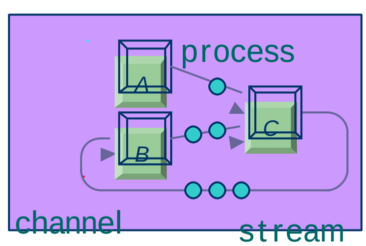
Weaknesses:

- Computation-intensive systems are overspecified
- Modularity is compromised
- Causality loops are possible
- Causality loops are hard to detect

5. Process Networks

- Processes are prefix-monotonic functions mapping sequences into sequences.
- One implementation uses blocking reads, non-blocking writes, and unbounded FIFO channels.

Examples:
• SDL,
• Unix pipes,
• ...



5. Strengths and Weaknesses

Strengths:

- Loose synchronization (distributable)
- Determinate under simple conditions
- Implementable under simple conditions
- Maps easily to threads, but much easier to use
- Turing complete (expressive)

Weaknesses:

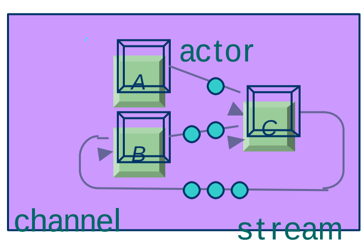
- Control-intensive systems are hard to specify
- Bounded resources are undecidable

6. Dataflow

- A special case of process networks where a process is made up of a sequence of firings (finite, atomic computations).
- Similar to Petri nets, but ordering is preserved in places.

Examples:

- SPW,
- HP Ptolemy,
- Cossap,
- ...



6. Strengths and Weaknesses

Strengths:

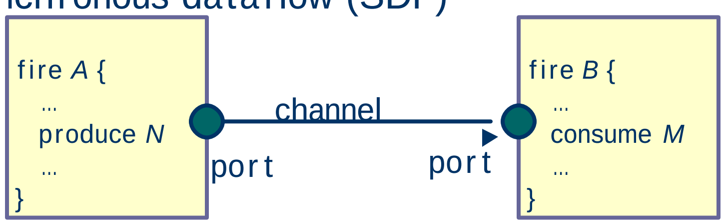
- Good match for signal processing
- Loose synchronization (distributable)
- Determinate under simple conditions
- Special cases map well to hardware and embedded software

Weakness:

- Control-intensive systems are hard to specify

6. Special Case: SDF

Synchronous dataflow (SDF)



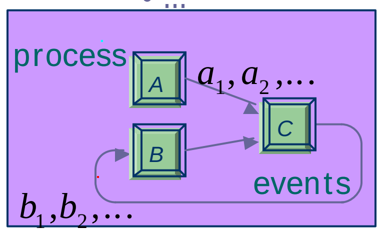
- Balance equations (one for each channel):

  $F_A N = F_B M$
- Schedulable statically
- Decidable resource requirements

7. Rendezvous Models

- Events represent rendezvous of a sender and a receiver. Communication is unbuffered and instantaneous.
- Often implicitly assumed with "process algebra" or even "concurrent."

Examples:
- CSP,
- CCS,
- Occam,
- Lotos,
- ...



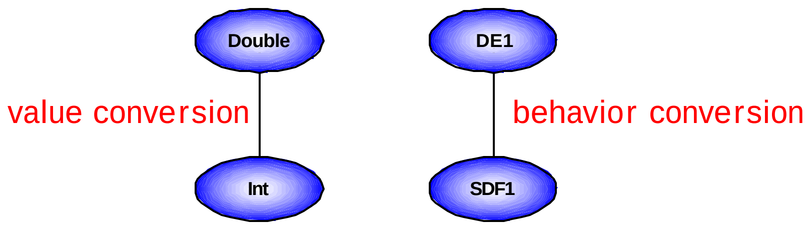
7. Strengths and Weaknesses

Strengths:
- Models resource sharing well
- Partial-order synchronization (distributable)
- Supports naturally nondeterminate interactions

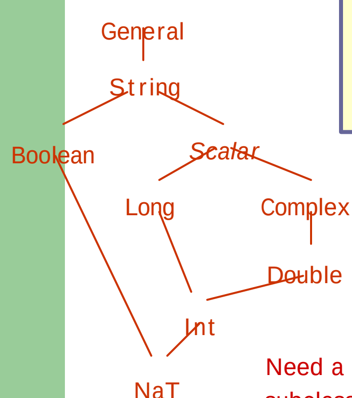
Weaknesses:
- Oversynchronizes some systems
- Difficult to make determinate (and useful)
- Difficult to avoid deadlock

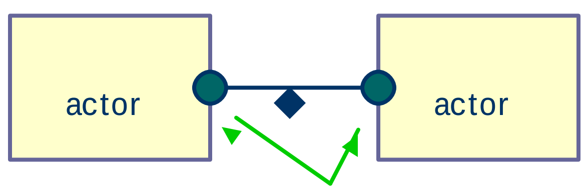
Making Sense of the Options: Component Interfaces

- Represent not just data types, but interaction types as well.



Approach – System-Level Types

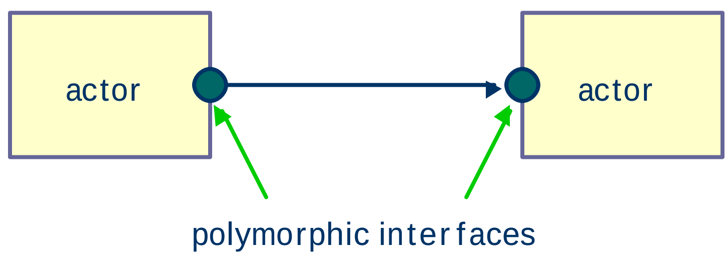




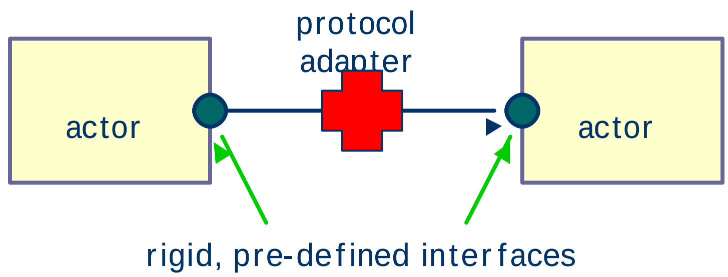
represent interaction semantics as types on these ports.

Need a new type lattice representing subclassing & ad-hoc convertibility.

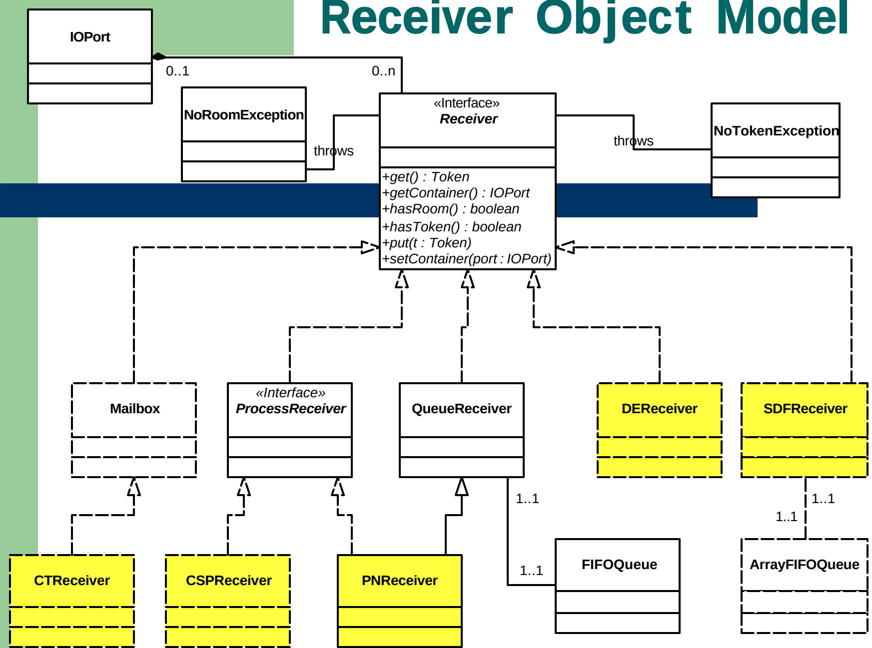
Our Hope –
Polymorphic Interfaces



More Common Approach –
Interface Synthesis



Receiver Object Model



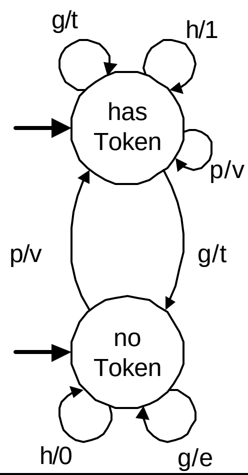
Receiver Interface

- get() : Token
- put(t : Token)
- hasRoom() : boolean
- hasToken() : boolean

The common interface makes it possible to define components that operate in multiple domains.

SDF Receiver Type Signature

SDF1

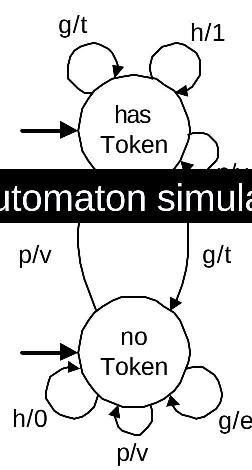


Input alphabet:
- g: get
- p: put
- h: hasToken

Output alphabet:
- 0: false
- 1: true
- t: token
- v: void
- e: exception

DE Receiver Type Signature

DE1



Input alphabet:
- g: get
- p: put
- h: hasToken

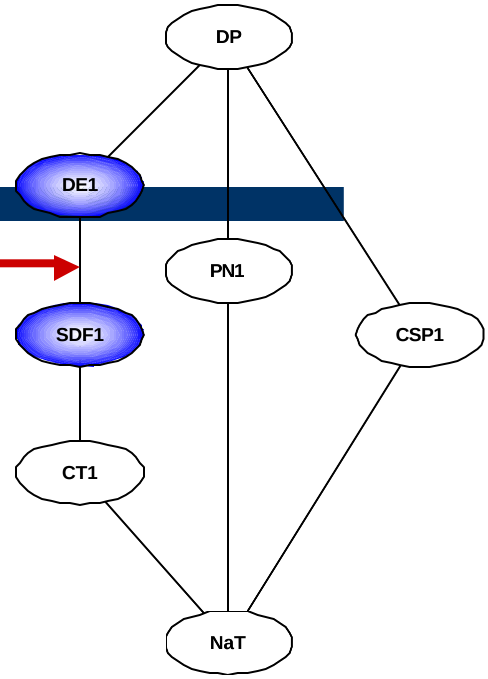
This automaton simulates the previous one

Put does not necessarily result in immediate availability of the data.

Output alphabet:
- 0: false
- 1: true
- t: token
- v: void
- e: exception

Type Lattice

Simulation relation



Simulation relation:

A relation between state spaces so that the upper machine simulates the behavior of the lower one.

Domain Polymorphism

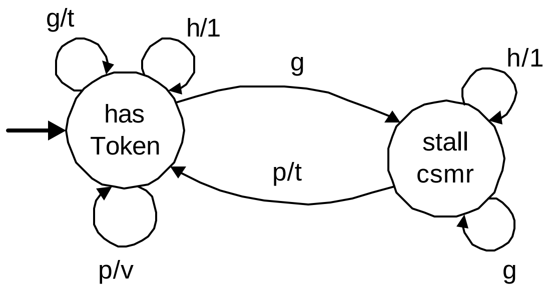
Components have meaning in multiple domains.

- Make the inputs as general as possible
  - Design to a receiver automaton that simulates that of several domains.
- Make the outputs as specific as possible
  - Design to a receiver automaton that is simulated by that of several domains.

Resolve to the most specific design that meets all the constraints.

Formulation: Least fixed point of a monotonic function on a type lattice.

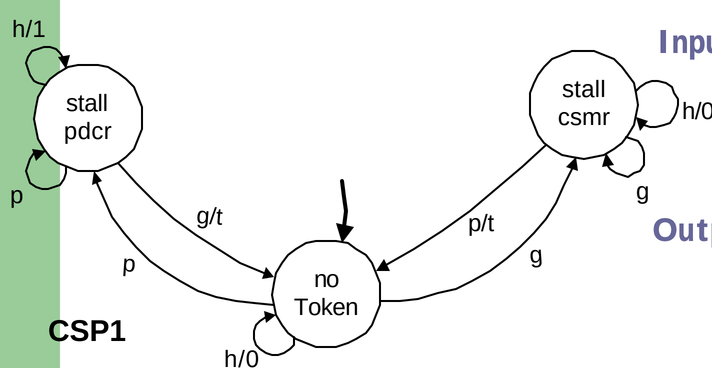
PN Receiver Type Signature



Input alphabet:
- g: get
- p: put
- h: hasToken

Output alphabet:
- 0: false
- 1: true
- t: token
- v: void
- e: exception

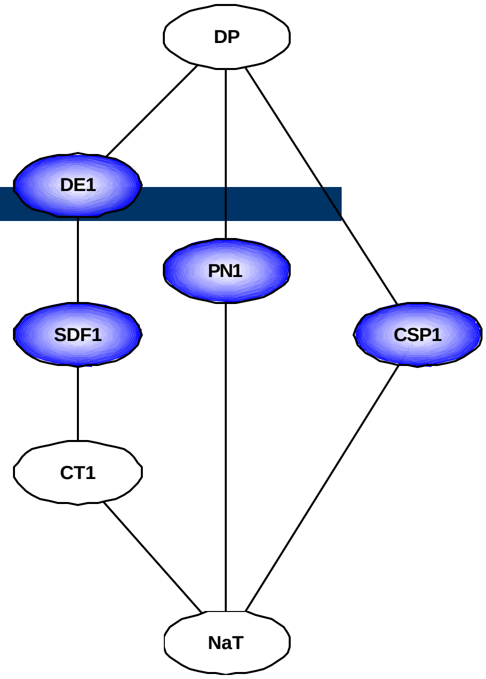
CSP Receiver Type Signature



Input alphabet:
- g: get
- p: put
- h: hasToken

Output alphabet:
- 0: false
- 1: true
- t: token
- v: void
- e: exception

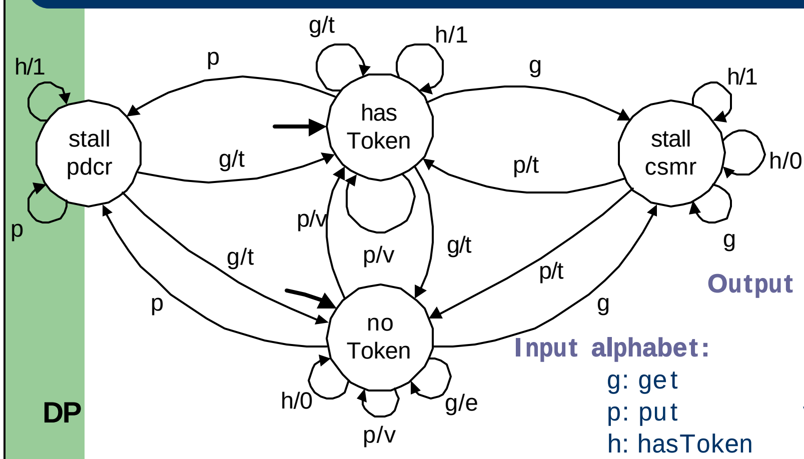
Type Lattice



Incomparable types:

PN and CSP are incomparable with DE and SDF. Does this mean we cannot design polymorphic components? No, it means we need to design them to the least upper bound.

Domain Polymorphic Type Signature



Input alphabet:
- g: get
- p: put
- h: hasToken

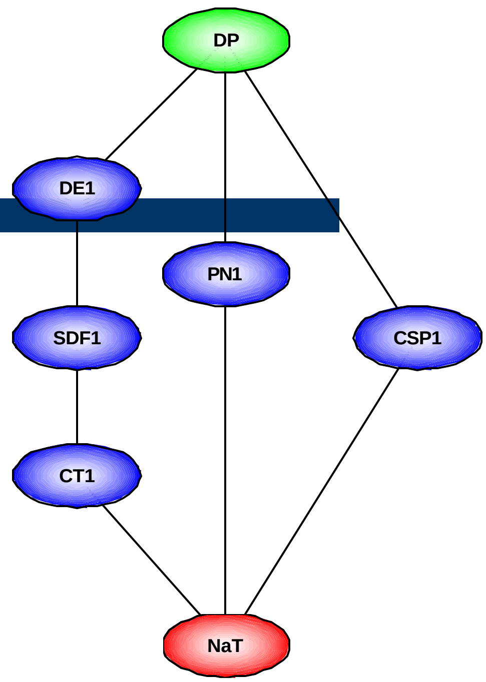
Output alphabet:
- 0: false
- 1: true
- t: token
- v: void
- e: exception

Type Lattice

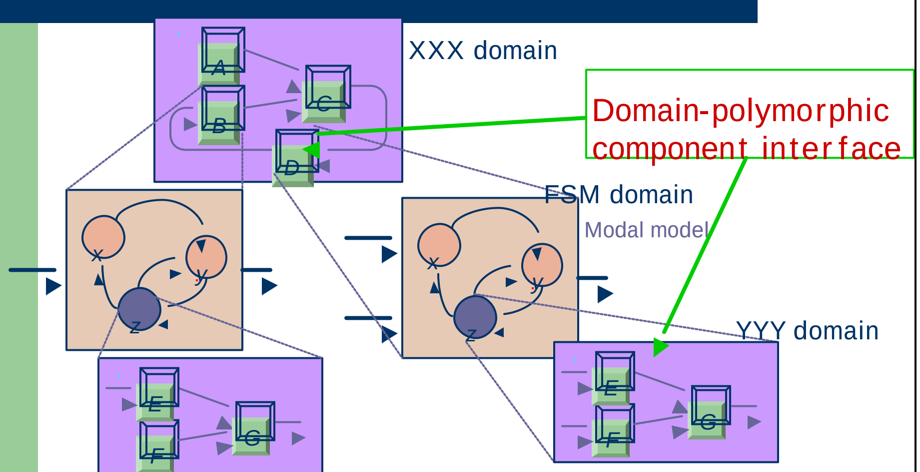
Constraints:

Actors impose inequality constraints w.r.t. this lattice. Connectivity also imposes constraints. Find the least solution that satisfies all constraints.

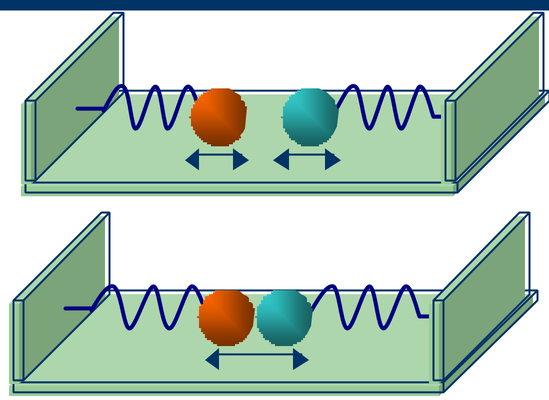
Finding the bottom element identifies a type conflict.



*Charts: Exploiting Domain Polymorphism



Special Case: Hybrid Systems

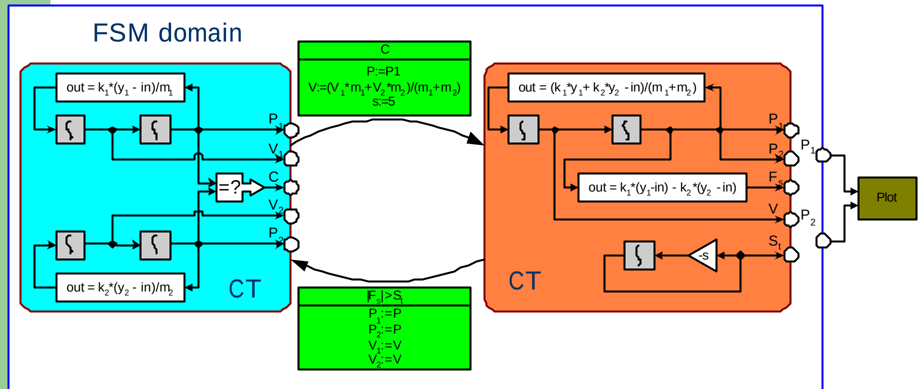


Example: Two point masses on springs on a frictionless table. They collide and stick together.

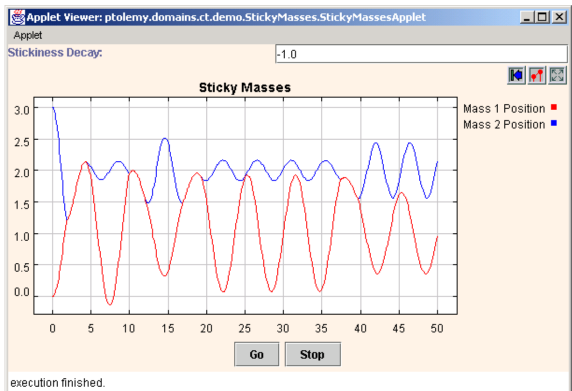
The stickiness is exponentially decaying with respect to time.

Hybrid System: Block Diagram

CT domain



Ptolemy II Execution



Because of domain polymorphism, Ptolemy II can combine FSMs hierarchically with any other domain, delivering models like statecharts (with SR) and SDL (with process networks) and many other modal modeling techniques.

Summary

- There is a rich set of component interaction models
- Hierarchical heterogeneity
  - more understandable designs than amorphous heterogeneity
- System-level types
  - Ensure component compatibility
  - Clarify interfaces
  - Provide the vocabulary for design patterns
  - Promote modularity and polymorphic component design
- Domain polymorphism
  - More flexible component libraries
  - A very powerful approach to heterogeneous modeling

Acknowledgements

The entire Ptolemy project team contributed immensely to this work, but particularly

- John Davis
- Chamberlain Fong
- Tom Henzinger
- Christopher Hylands
- Jie Liu
- Xiaojun Liu
- Steve Neuendorffer
- Neil Smyth
- Kees Vissers
- Yuhong Xiong