The Ptolemy Project

Modeling and Design of Reactive Systems

Edward A. Lee
Professor

UC Berkeley
Dept. of EECS

Abstract

Ptolemy is a research project and software environment focused on the design and modeling of reactive systems, providing high-level support for signal processing, communication, and real-time control. The key underlying principle in the project is the use of multiple models of computation in a hierarchical heterogeneous design and modeling environment. This talk gives an overview of some of the models of computation of interest, with a focus on their concurrency, thier ability to model and specify real-time systems, and their ability to mix control logic with signal processing.

Organizational

Staff

Diane Chang, administrative assistant
Kevin Chang, programmer
Christopher Hylands, programmer analyst
Edward A. Lee, professor and PI
Mary Stewart, programmer analyst

Postdocs

Praveen Murthy
Seehyun Kim
John Reekie
Dick Stevens (on leave from NRL)

Students

Cliff Cordeiro
John Davis
Stephen Edwards
Ron Galicia
Mudit Goel
Michael Goodwin
Bilung Lee
Jie Liu
Michael C. Williamson
Yuhong Xiong

Undergraduate Students

Sunil Bhave
Luis Gutierrez

Key Outside Collaborators

Shuvra Bhattacharyya (Hitachi)
Joseph T. Buck (Synopsys)
Brian L. Evans (UT Austin)
Soonhoi Ha (Seoul N. Univ.)
Tom Lane (SSS)
Thomas M. Parks (Lincoln Labs)
José Luis Pino (Hewlett Packard)

Sponsors

DARPA
MICRO
The Alta Group of Cadence
Hewlett Packard
Hitachi
Hughes
LG Electronics
NEC
Philips
Rockwell
SRC

Types of Computational Systems

Transformational

- transform a body of input data into a body of output data

Interactive

- interact with the environment at their own speed

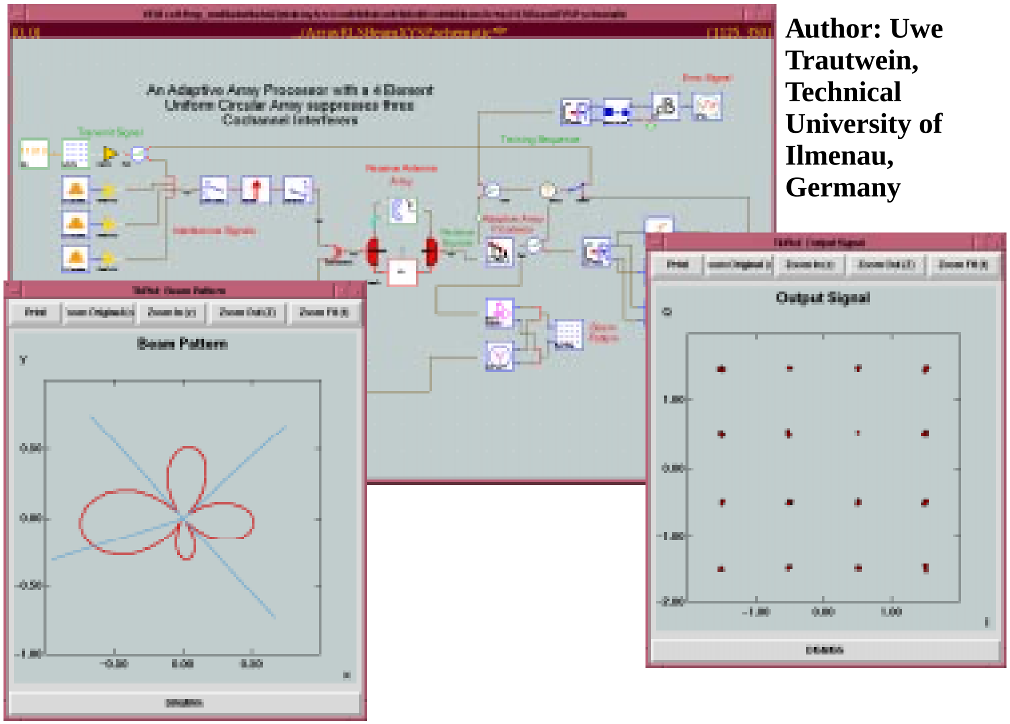
Reactive

- react continuously at the speed of the environment

This project focuses on design of reactive systems

- real-time
- embedded
- concurrent
- network-aware
- adaptive
- heterogeneous

Interactive, High-Level Simulation and Specification



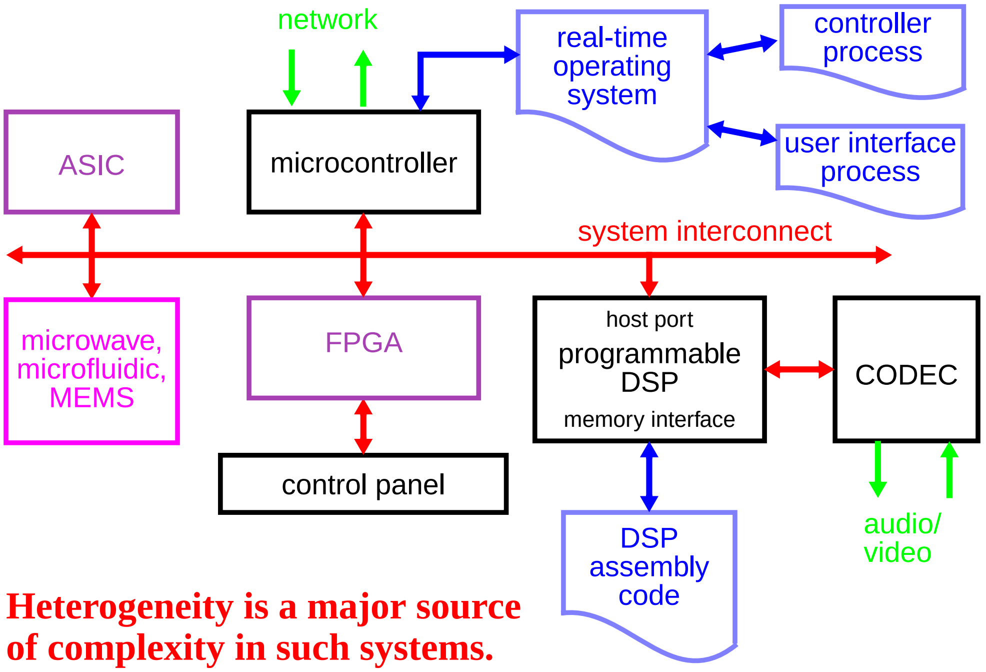
Author: Uwe Trautwein, Technical University of Ilmenau, Germany

---

Properties of Such Specifications

- Modular
  - Large designs are composed of smaller designs
  - Modules encapsulate specialized expertise
- Hierarchical
  - Composite designs themselves become modules
  - Modules may be very complicated
- Concurrent
  - Modules logically operate simultaneously
  - Implementations may be sequential or parallel or distributed
- Abstract
  - The interaction of modules occurs within a “model of computation”
  - Many interesting and useful MoCs have emerged
- Domain Specific
  - Expertise encapsulated in MoCs and libraries of modules.

Heterogeneous Implementation Architectures



network

real-time operating system

controller process

user interface process

ASIC

microcontroller

system interconnect

microwave, microfluidic, MEMS

FPGA

host port

programmable DSP

memory interface

CODEC

control panel

DSP assembly code

audio/ video

Heterogeneity is a major source of complexity in such systems.

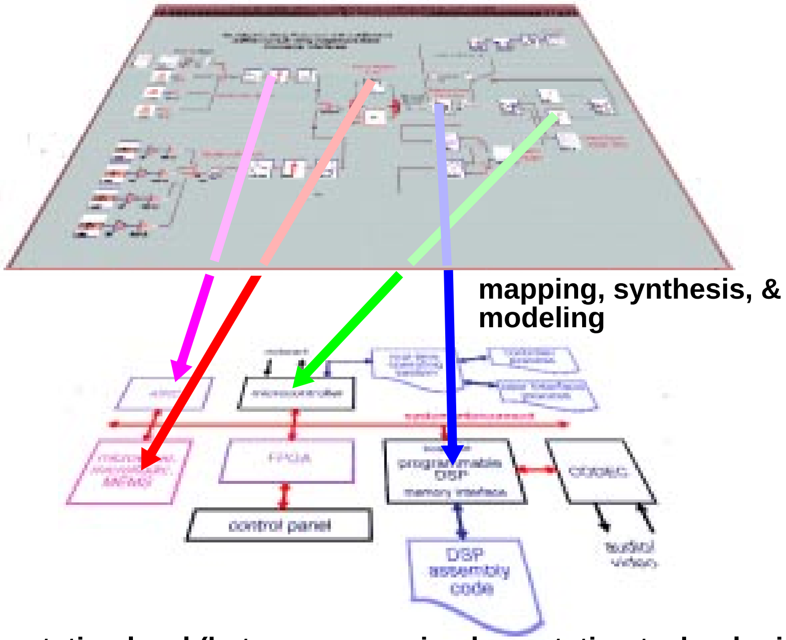
Two Approaches to the Design of Such Systems

- **The grand-unified approach**
  - **Find a common representation language for all components**
  - **Develop techniques to synthesize diverse implementations from this**
- **The heterogeneous approach**
  - **Find domain-specific *models of computation* (MoC)**
  - **Hierarchically mix and match MoCs to define a system**
  - **Retargetable synthesis techniques from MoCs to diverse implementations**

The Ptolemy project is pursuing the latter approach

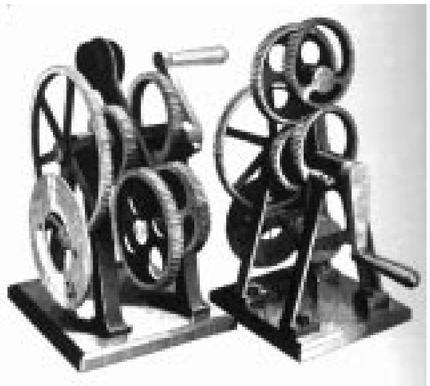
- **Domain specific MoCs match the applications better**
- **Choice of MoC can profoundly affect system architecture**
- **Choice of MoC can limit implementation options**
- **Synthesis from specialized MoCs is easier than from GULs.**

Heterogeneous System-Level Specification & Modeling

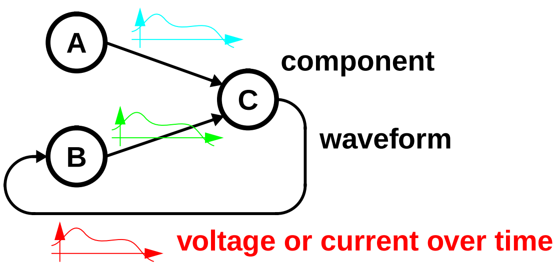


Some Problem-Level Models of Computation

- Gears
- Differential equations
- Difference equations
- Discrete-events
- Petri nets
- Dataflow
- Process networks
- Actors
- Threads
- Synchronous/reactive languages
- Communicating sequential processes
- Hierarchical communicating finite state machines



Example — Analog Circuit Modeling



Strengths:

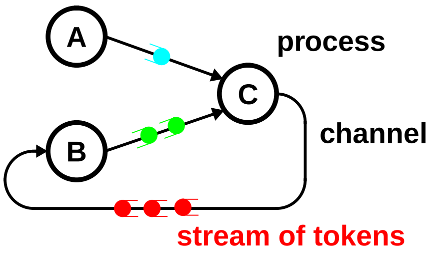
- Accurate model for many physical systems
- Declarative
- Determinate

Weaknesses:

- Tightly bound to an implementation
- Expensive to simulate
- Difficult to implement in software

Example — Process Networks

Note: Dataflow is
a special case.



Strengths:

- Good match for signal processing
- Loose synchronization (distributable)
- Determinate
- Maps easily to threads
- Dataflow special cases map well to hardware and embedded software

Weakness:

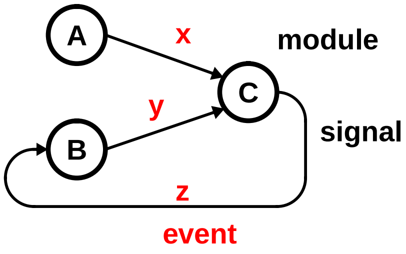
- Control-intensive systems are hard to specify

Our Contributions to Dataflow Modeling

— the most mature parts of Ptolemy —

- **Compile-time scheduling of *synchronous dataflow* graphs with optimized partitioning and memory utilization.**
- **Specification of the *Boolean dataflow (BDF) model*, which is Turing complete.**
- **Proof that the existence of a finite complete cycle and a bounded memory implementation for BDF is *undecidable*.**
- ***Heuristics* for constructing finite complete cycles and bounded memory schedules most of the time.**
- ***Multidimensional* generalization to dataflow models.**
- ***Process network* model generalization to dataflow.**
- ***Visual programming* formulation and use of *higher-order functions*.**

Example — 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:

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

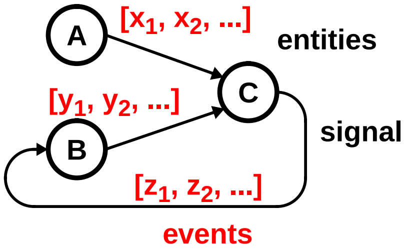
Strengths:

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

Weaknesses:

- **Computation-intensive systems are overspecified**
- **Modularity is compromised**

Example — Discrete-Event Models



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

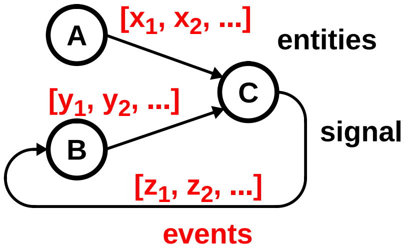
Strengths:

- Natural description of digital hardware
- Global synchronization
- Can be made determinate (often is not, however)

Weaknesses:

- Expensive to implement in software
- May over-specify and/or over-model systems (global time)

Rendezvous Models



Events represent rendezvous of a sender and a receiver. Communication is unbuffered and instantaneous. Examples include CSP and CCS.

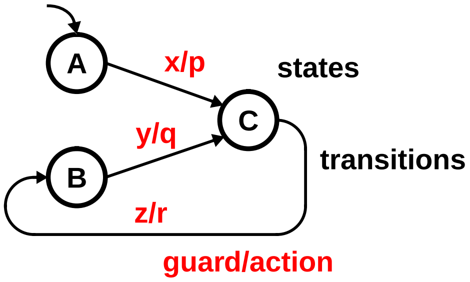
Strengths:

- Models resource sharing well.
- Partial-order synchronization.
- Supports naturally nondeterminate interactions.

Weaknesses:

- Oversynchronizes some systems.

Sequential Example — Finite State Machines



Guards determine when a transition may be made from one state to another, in terms of events that are visible, and outputs assert other events.

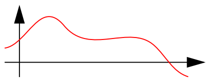
Strengths:

- **Natural description of sequential control**
- **Behavior is decidable**
- **Can be made determinate (often is not, however)**
- **Good match to hardware or software implementation**

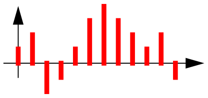
Weaknesses:

- **Awkward to specify numeric computation**
- **Size of the state space can get large**

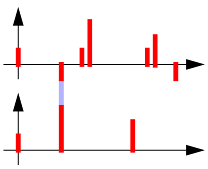
Essential Differences — Models of Time



continuous time

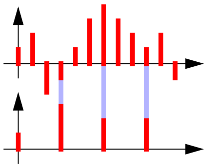


discrete time

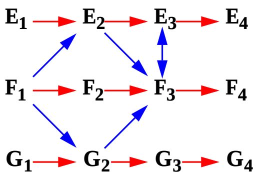


totally-ordered discrete events

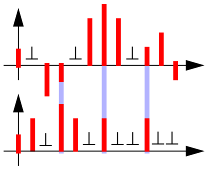
Salvador Dali, *The Persistence of Memory*, 1931



multirate discrete time



partially-ordered discrete events



synchronous/reactive

Key Issues in these Models of Computation

- Maintaining determinacy.
- Supporting nondeterminacy.
- Bounding the queueing on channels.
- Scheduling processes.
- Synthesis: mapping to hardware/software implementations.
- Providing scalable visual syntaxes.
- Resolving circular dependencies.
- Modeling causality.
- Achieving fast simulations.
- Supporting modularity.
- Composing multiple models of computation.

Choosing Models of Computation

Validation methods

- By construction
  - property is inherent.
- By verification
  - property is provable syntactically.
- By simulation
  - check behavior for all inputs.
- By testing
  - observation of a prototype.
- By intuition
  - property is true, I think.
- By assertion
  - property is true. That’s an order.

Meret Oppenheim, *Object*, 1936

It is generally better to be higher in this list

Usefulness of Modeling Frameworks

The following objectives are at odds with one another:

- **Expressiveness**
- **Generality**

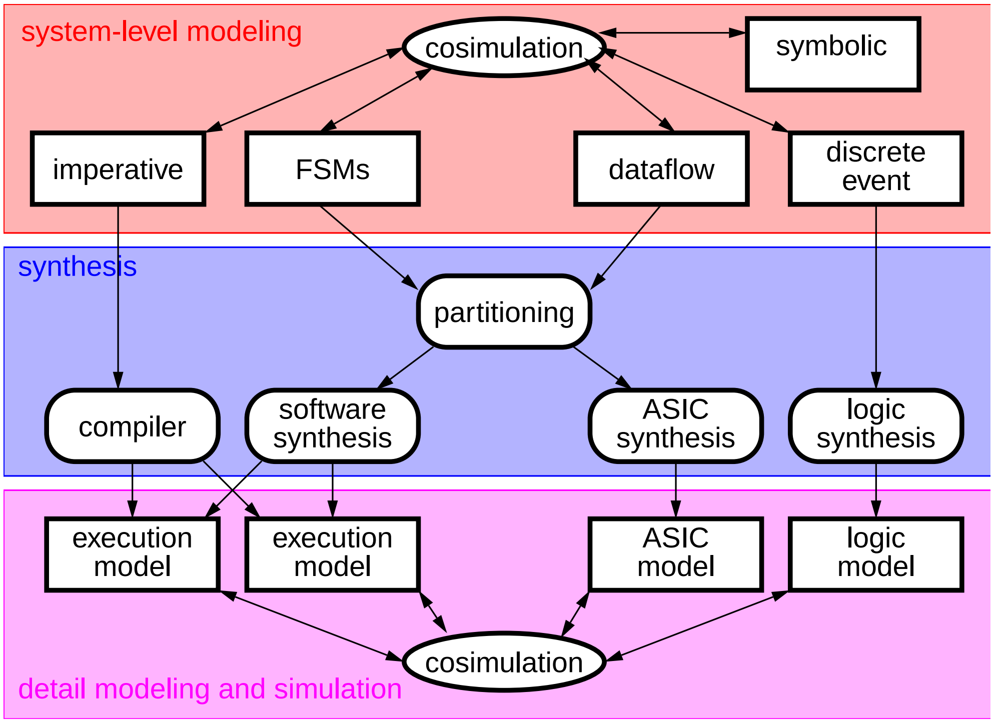
vs.

- **Verifiability**
- **Compilability/Synthesizability**

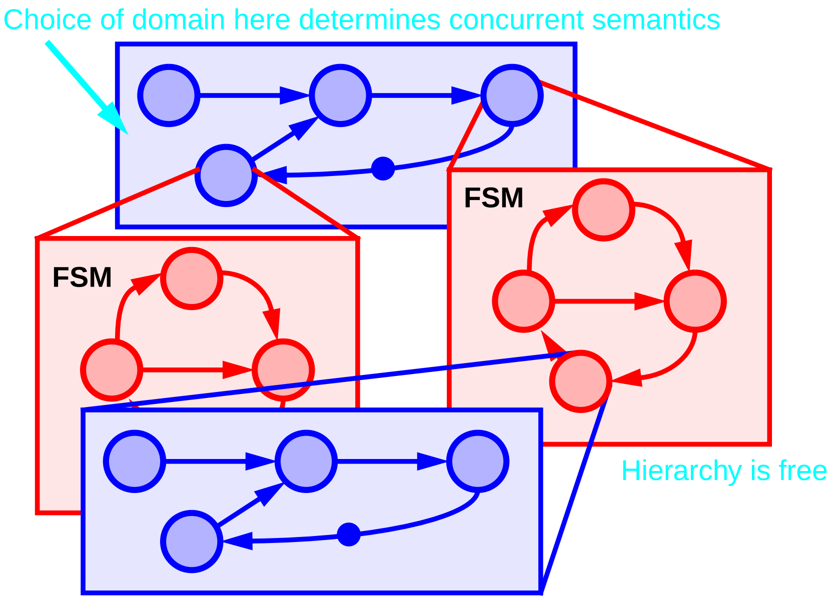
The Conclusion?

Heterogeneous modeling.

A Mixed Design Flow



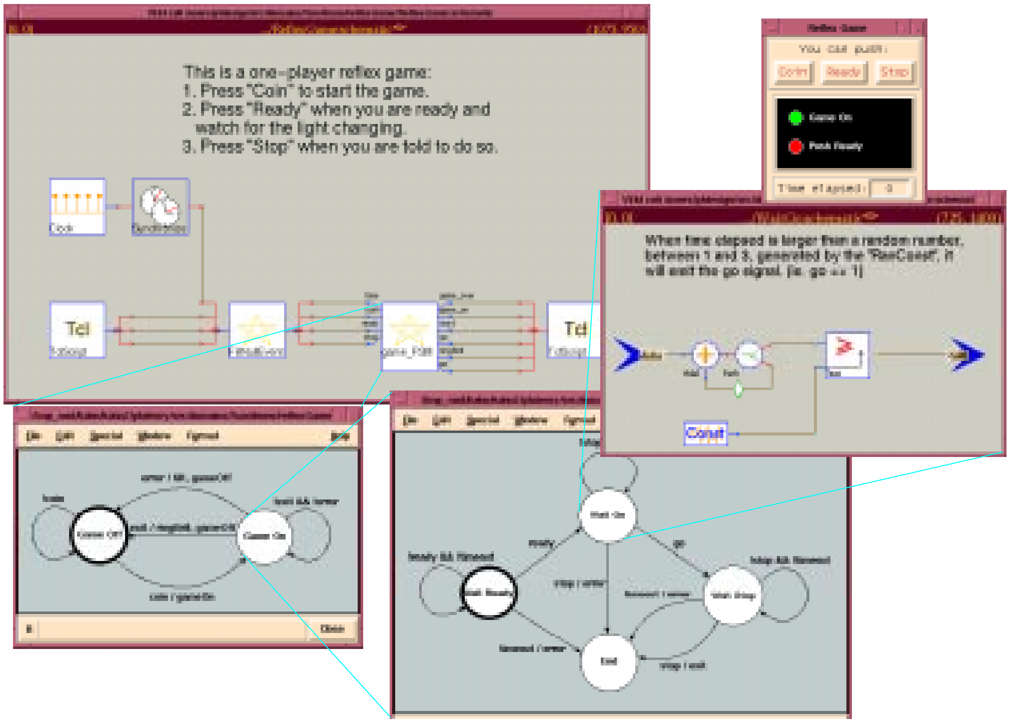
Mixing Control and Signal Processing — *Charts



Choice of domain here determines concurrent semantics

Hierarchy is free

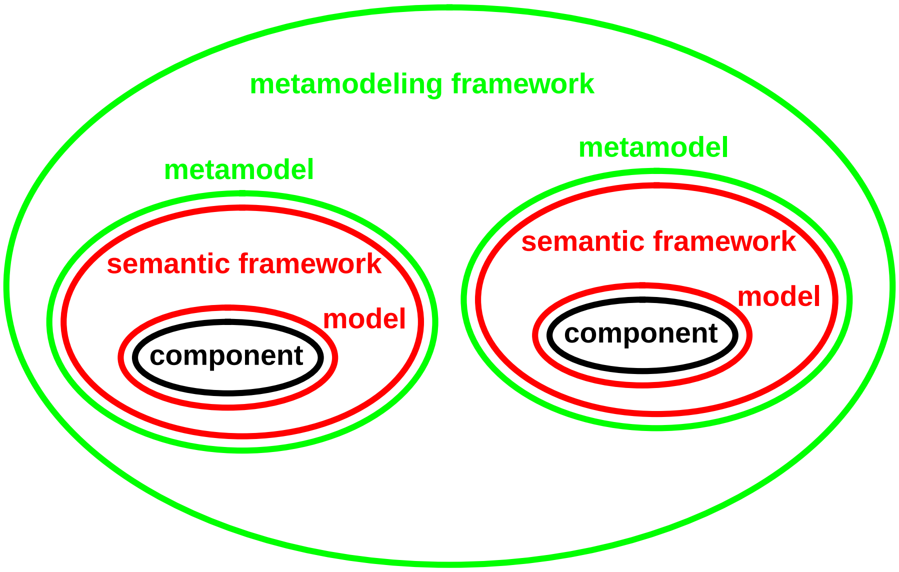
Example: DE, Dataflow, and FSMs



This is a one-player reflex game:
1. Press "Coin" to start the game.
2. Press "Ready" when you are ready and watch for the light changing.
3. Press "Stop" when you are told to do so.

When time elapsed is larger than a random number, between 1 and 3, generated by the "RanConst", it will emit the go signal. (ie. go == 1)

Metamodeling



metamodeling framework

metamodel

semantic framework

model

component

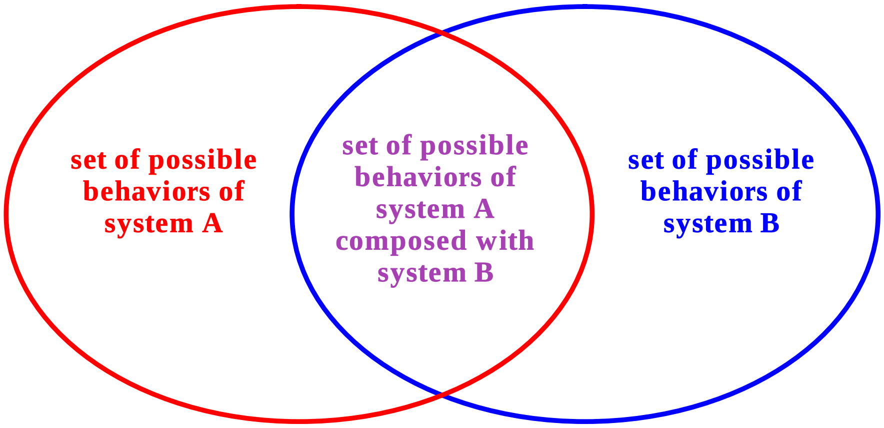
metamodel

semantic framework

model

component

Constraint-Based Metamodeling Frameworks



set of possible behaviors of system A

set of possible behaviors of system A composed with system B

set of possible behaviors of system B

These sets might be deterministic or random, exact or approximate.

Uses for Metamodeling

- Heterogeneous mixtures of semantic frameworks
  - heterogeneous systems
  - multiple views of the same system
- Design analysis
  - check aspects of correctness
  - discover opportunities for optimization
- Design refinement
  - the set of all possible design refinements gives the concretization operator
- Run-time modeling
  - reflection
  - model discovery and adaptation
  - model-driven control

Milestones in the Ptolemy Project

- 1990 — started with seed support from DARPA VLSI program. Focus on embedded DSP software and communication networks.
- 1993 — joined DARPA RASSP program. Focus on high-throughput embedded real-time signal processing systems.
- 1995 — The Alta Group at Cadence announces software using Ptolemy dataflow and mixed dataflow/discrete-event technology (SPW).
- 1997 — joined DARPA Composite CAD program. Focus on distributed adaptive reactive systems with mixed implementation technologies and modeling techniques.
- 1997 — Hewlett-Packard (EEsof) announces “HP Ptolemy,” an integration of Ptolemy dataflow technology with analog RF and microwave design and modeling tools.

Ptolemy Software as a Tool and as a Laboratory

Ptolemy software is

- Extensible
- Publicly available
- An open architecture
- Object-oriented

Allows for experiments with:

- Models of computation
- Heterogeneous design
- Domain-specific tools
- Design methodology
- Software synthesis
- Hardware synthesis
- Cosimulation
- Cosynthesis
- Visual syntaxes (Tycho)

Further Information

Welcome to the Ptolemy Project

Ptolemy Project

Objectives – People – Sponsors – ILP Research Summaries – Rapid Prototyping (RASSP) Project – Distributed Adaptive Signal Processing (DASP) Project – Publications – Technology Transfer – News

Ptolemy Software

Copyright – Almagest – FAQ – Downloads – Tycho – Current Development – Mailing Lists – Tutorial – Quick Tour – Undergraduate Projects

DSP Design Group

Seminars – Education – Related research – Software

Ptolemy is a research project and software environment focused on the design of reactive systems, providing high-level support for signal processing, communications, and real-time control. The key underlying principle in the project is the use of multiple models of computation in a hierarchical heterogeneous design environment. The work is conducted by the DSP Design Group of the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley. The project is directed by Prof. Edward Lee and Prof. David Messerschmitt. The project is named after the second century Greek astronomer, mathematician, and geographer.

- Software distributions
- Small demonstration versions
- Project overview
- *The Almagest* (software manual)
- Current projects summary
- Project publications
- Keyword searching
- Project participants
- Sponsors
- Copy of the FAQh
- Newsgroup info
- Mailing lists info

http://ptolemy.eecs.berkeley.edu