The Ptolemy Project

Edward A. Lee
Professor and
Principal Investigator

UC Berkeley
Dept. of EECS

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

Adaptive Systems

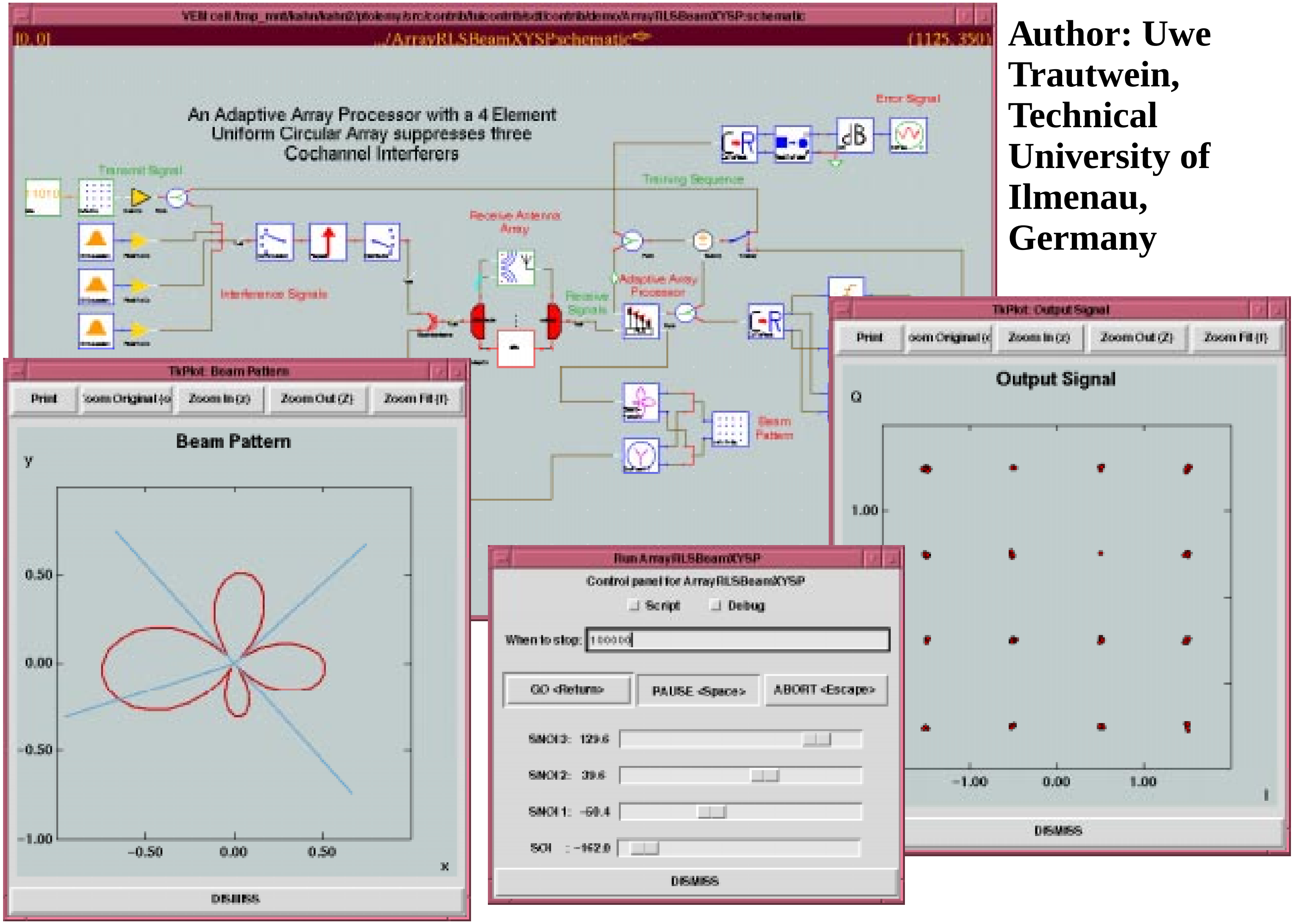
Classical adaptive signal processing

- **system identification**
- **interference nulling**
- **reversing distortion**

Resource adaptive signal processing

- **conserving power**
- **meeting changing latency and QOS requirements**
- **using available sensor data**
- **using network resources (memory, cycles, bandwidth)**

Interactive, High-Level Simulation and Specification

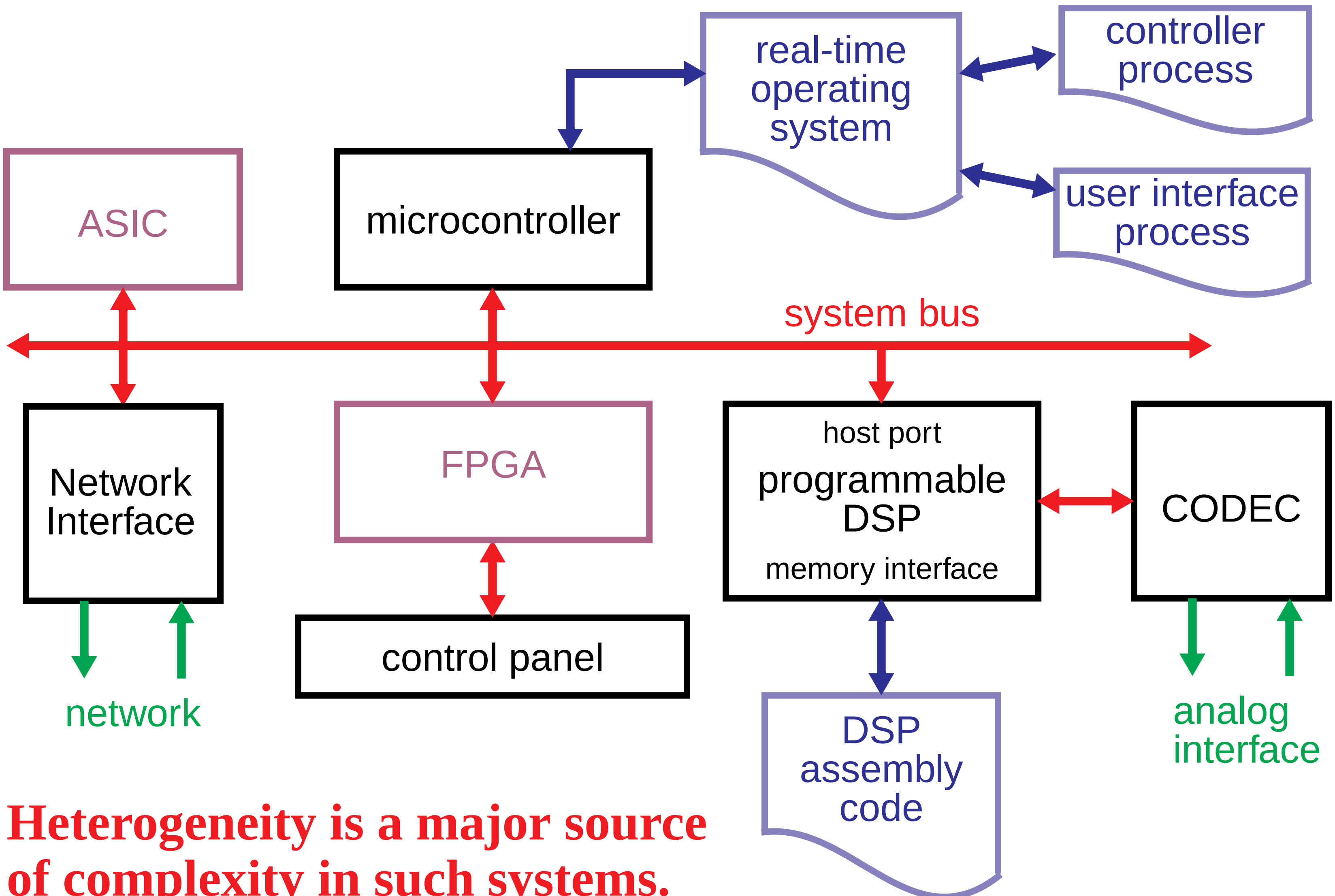


Author: Uwe Trautwein, Technical University of Ilmenau, Germany

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

Typical Implementation



Heterogeneity is a major source of complexity in such systems.

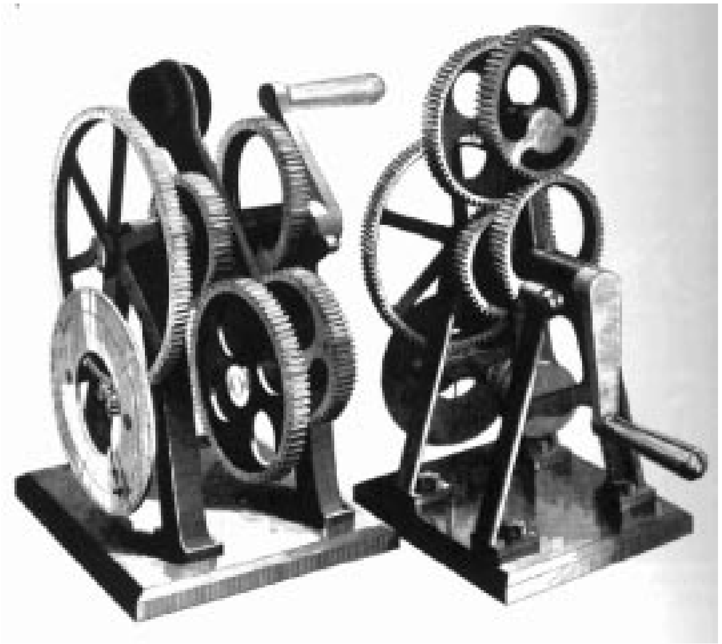
Two Approaches to the Design of Such Systems

- **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**
  - **Retargettable 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.**

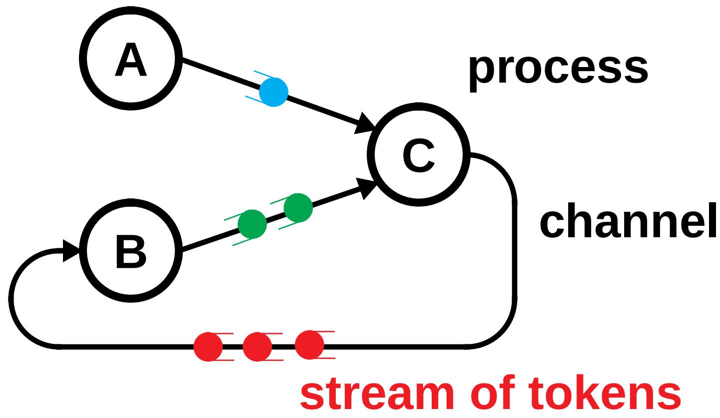
Some Concurrent Models of Computation



- Gears
- Threads
- Petri nets
- Synchronous dataflow
- Dynamic dataflow
- Process networks
- Concrete data structures
- Discrete-events
- Synchronous/Reactive languages
- Communicating sequential processes
- Hierarchical communicating finite state machines

Example — Process Networks

Note: Dataflow is
a special case.



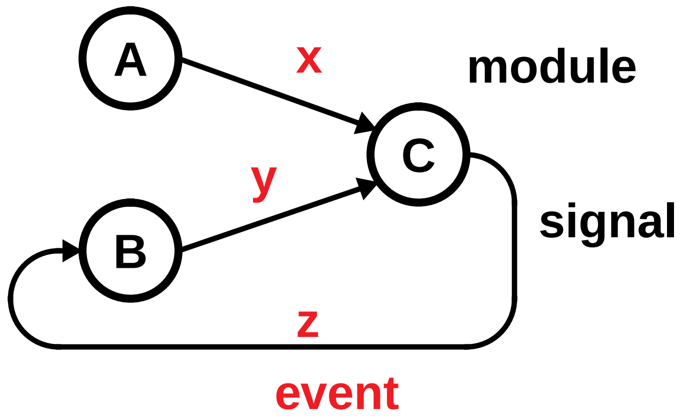
Strengths:

- 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

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:

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

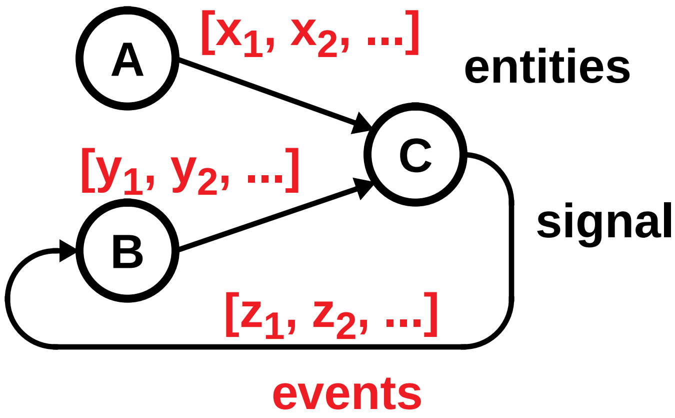
Strengths:

- 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.

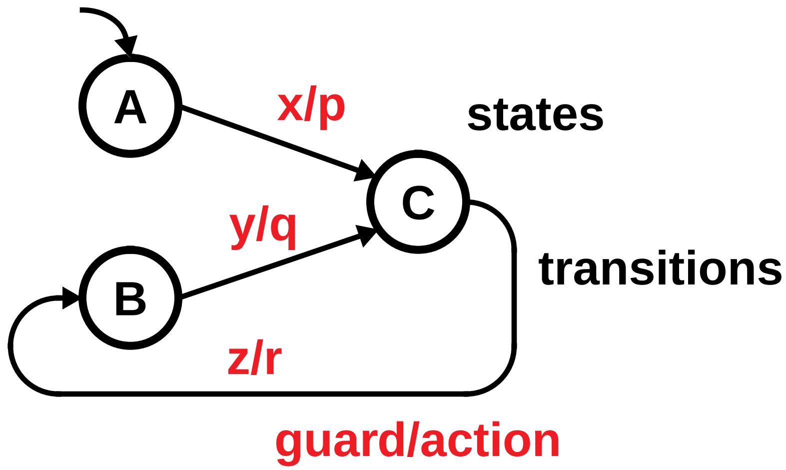
Strengths:

- Natural description of 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)

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.

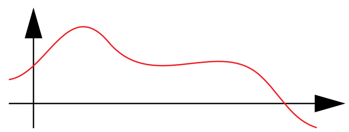
Strengths:

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

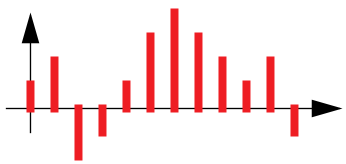
Weaknesses:

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

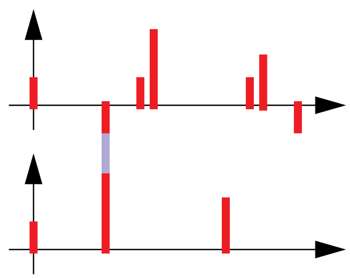
Essential Differences — Models of Time

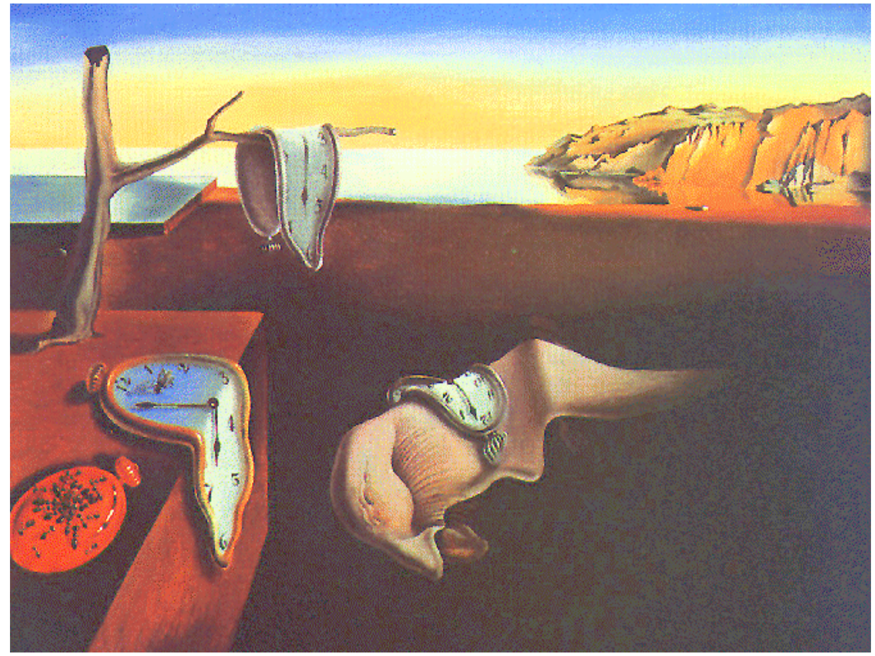


continuous time



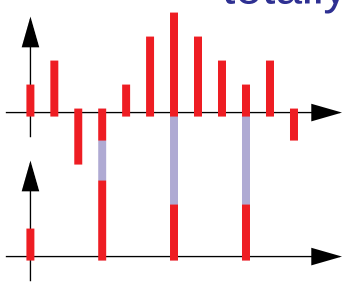
discrete time



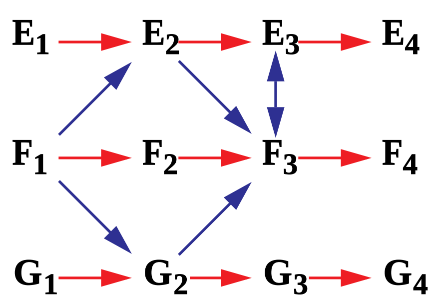


Salvador Dali, ***The Persistence of Memory***, 1931

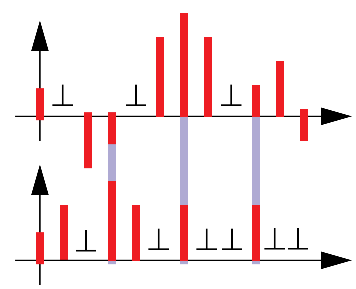
totally-ordered discrete events



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 (gray box model for modules).
- Composing multiple models of computation.

Validation methods

- **By construction**
  - property is inherent.
- **By verification**
  - theorem proving or algorithm.
- **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

The following objectives are at odds with one another:

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

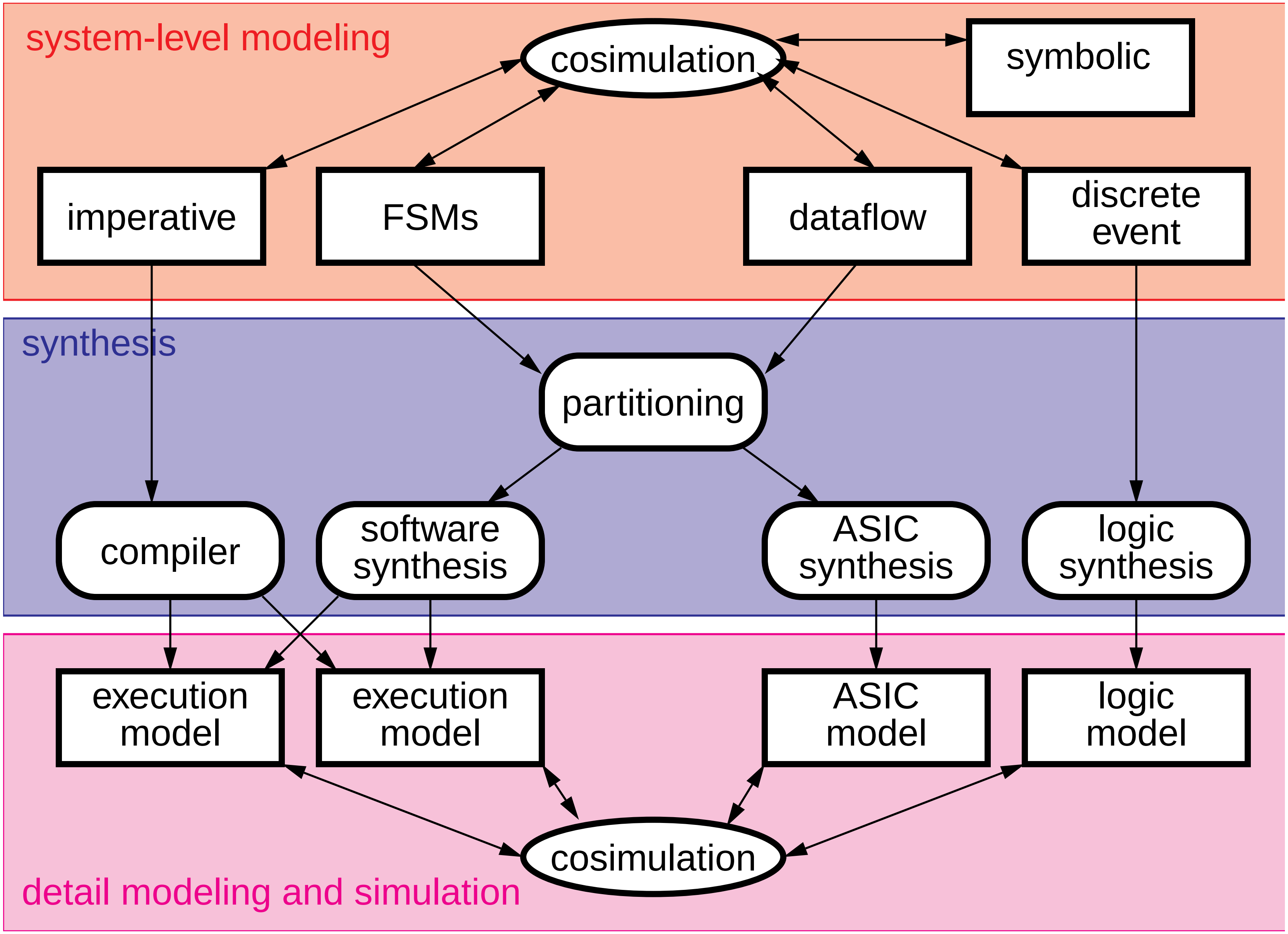
vs.

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

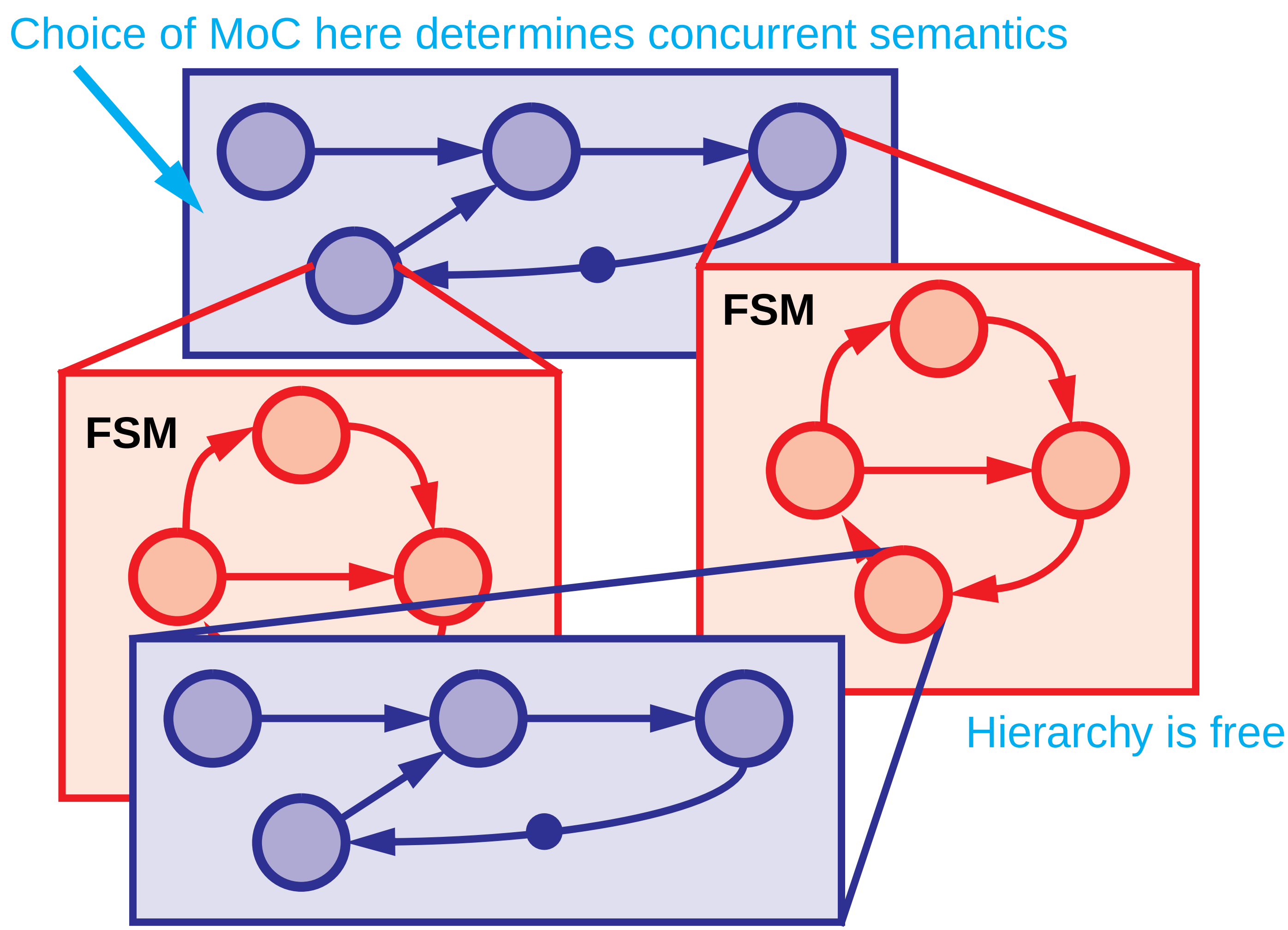
The Conclusion?

Heterogeneous modeling.

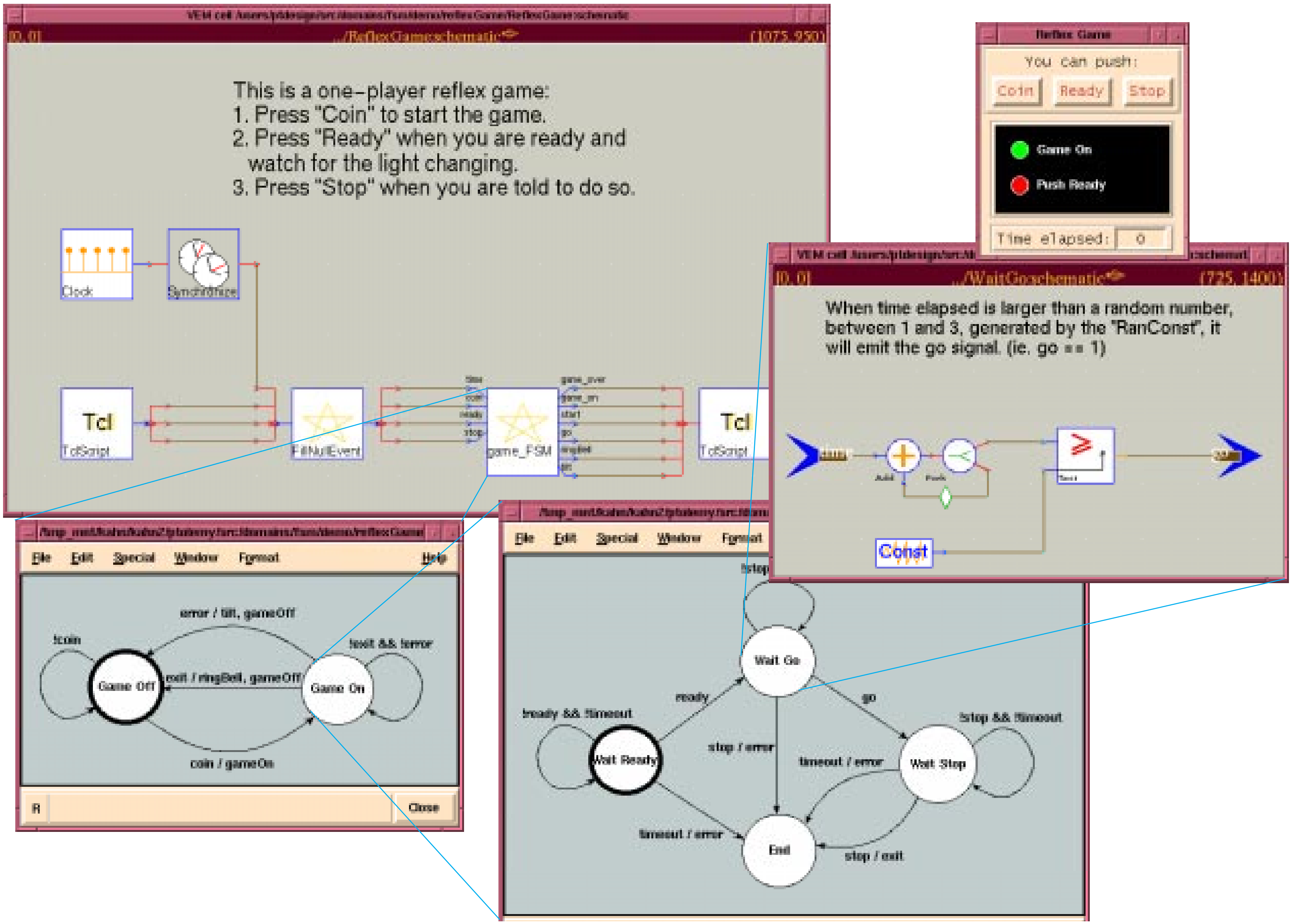
A Mixed Design Flow



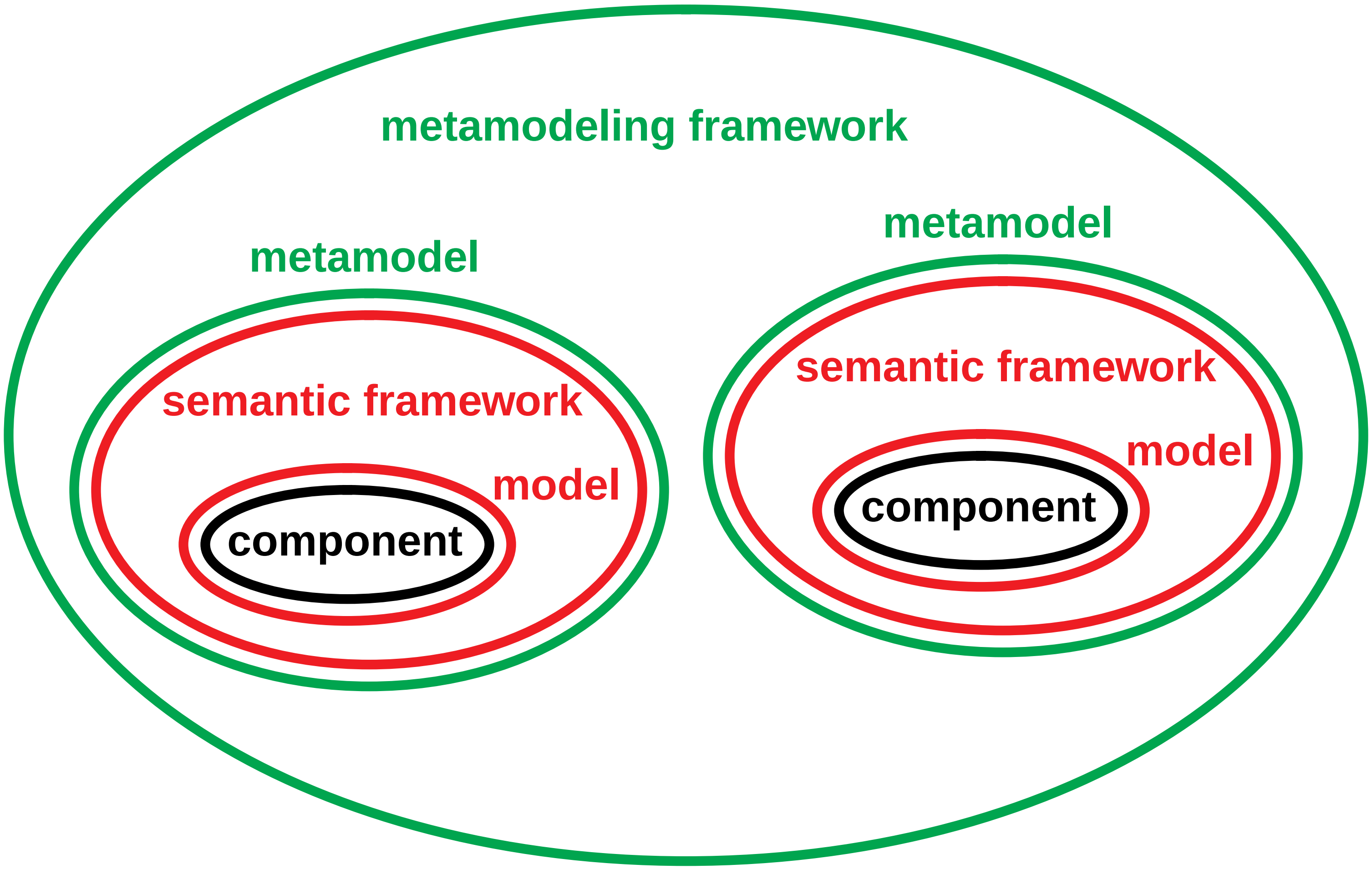
An Example of Hiearchical Heterogeneity: *Charts



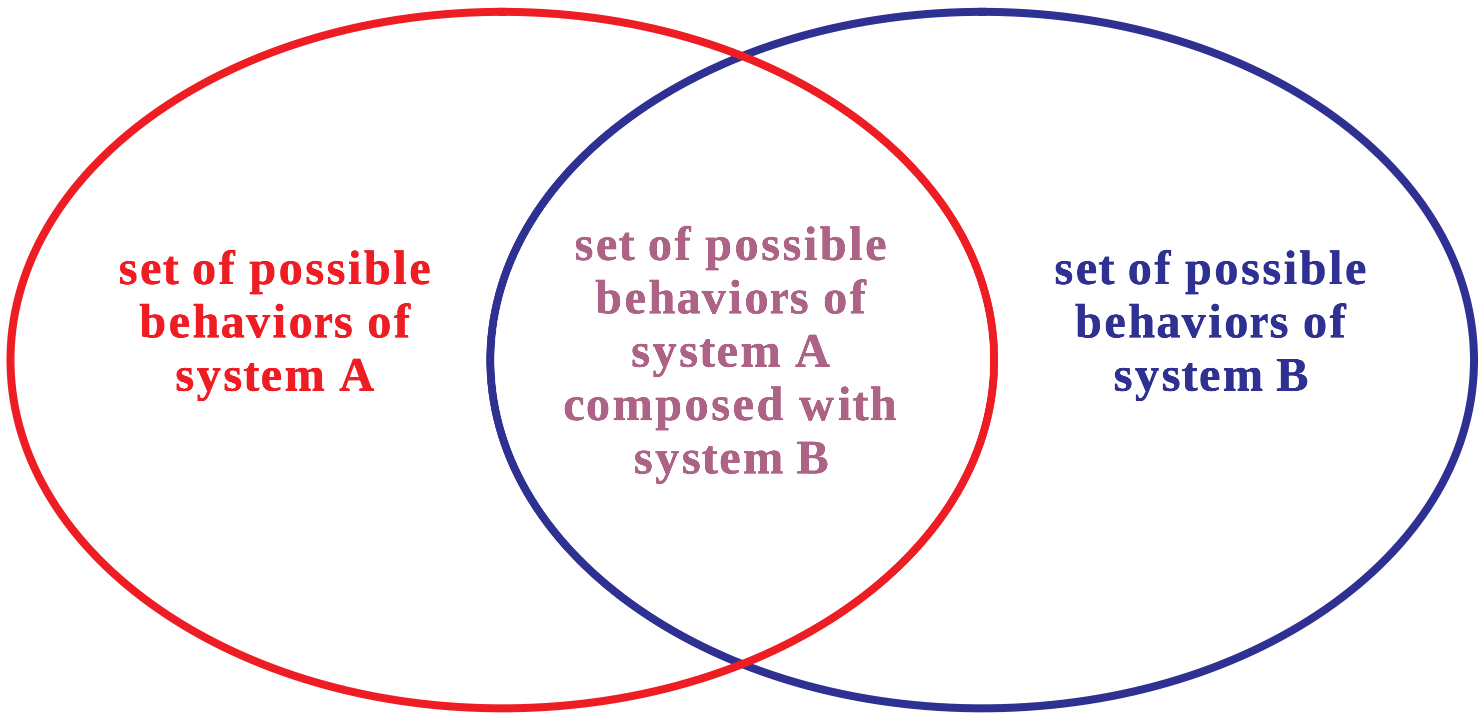
Example: DE, Dataflow, and FSMs



Metamodeling



Constraint-Based Metamodeling Frameworks



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**

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)

Modular Deployable Design Tools

Past design software:

- Monolithic
- Huge
- Back-room use

Future design software:

- Modular
- Deployable
- In-the-field evolution

Initial Strategy

Toolkit approach to design, creating an environment that is

- **safe (no core dumps)**
- **extensible**
- **distributable**
- **concurrent**
- **portable**

Deployed designs must minimize the use of

- **C, C++**
- **Thus, most of the existing Ptolemy kernel**

Initial Languages

In addition to satisfying all the above,

Tcl/Tk/Itcl

- **scripting language**
- **high-level, object-oriented**
- **universal, communicable data type (strings)**
- **extensive graphical user interface toolkits**

Java

- **faster (we have measured up to 8x)**
- **lower-level, object-oriented**
- **modularity built in**
- **concurrent (threads), although at a very low level**

Tycho

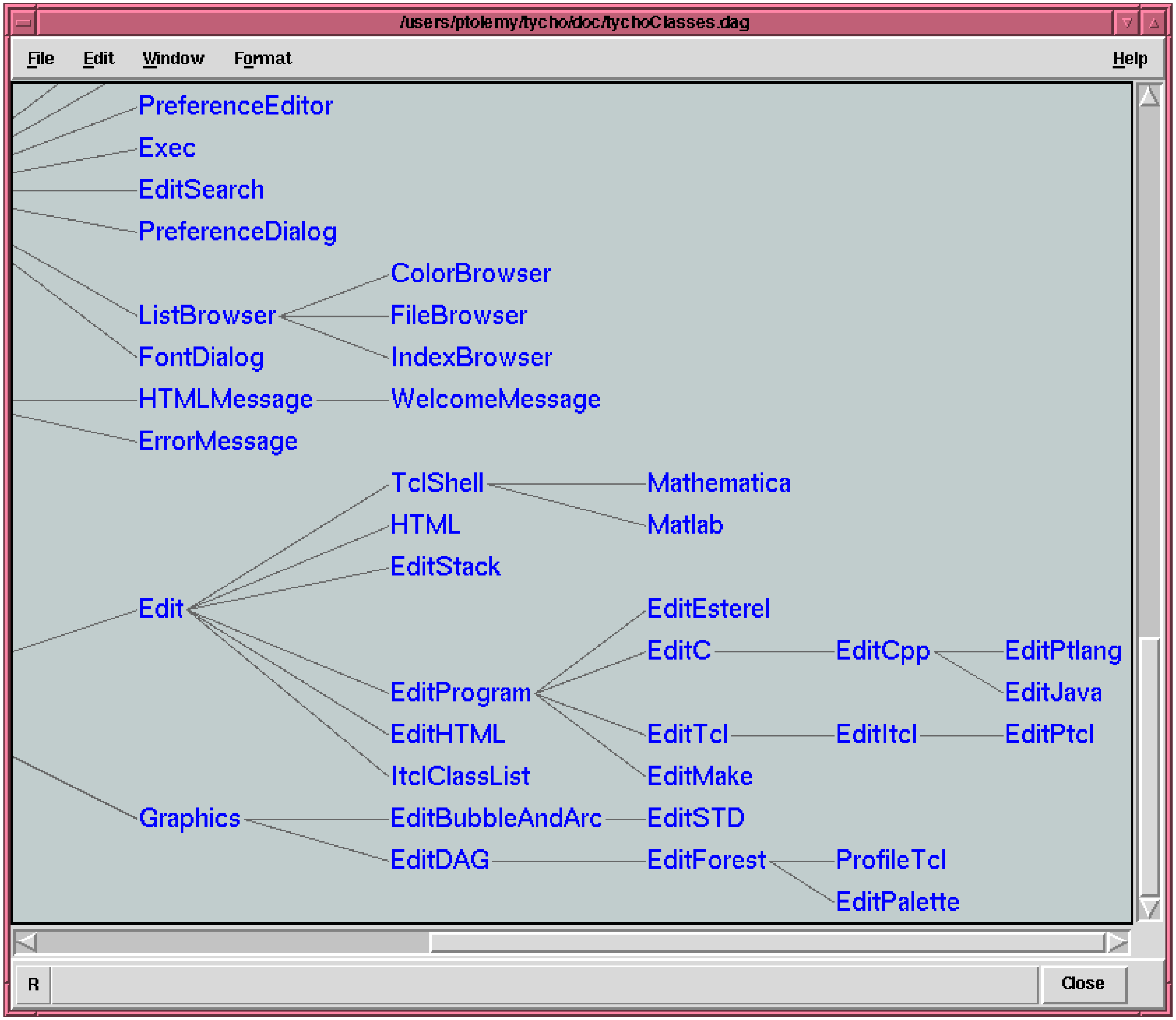
Modular Itcl class library

- **system control**
- **configuration**
- **user interface**

Current facilities:

- **context-sensitive text editors**
- **scripting shells (Tcl, Matlab, Mathematica)**
- **graphics toolkit (the Tycho Slate)**
- **integrated, interactive, HTML documentation**
- **preferences manager, version control, widget library**

A Portion of the Class Hierarchy (displayed in Tycho)



The Tycho Slate

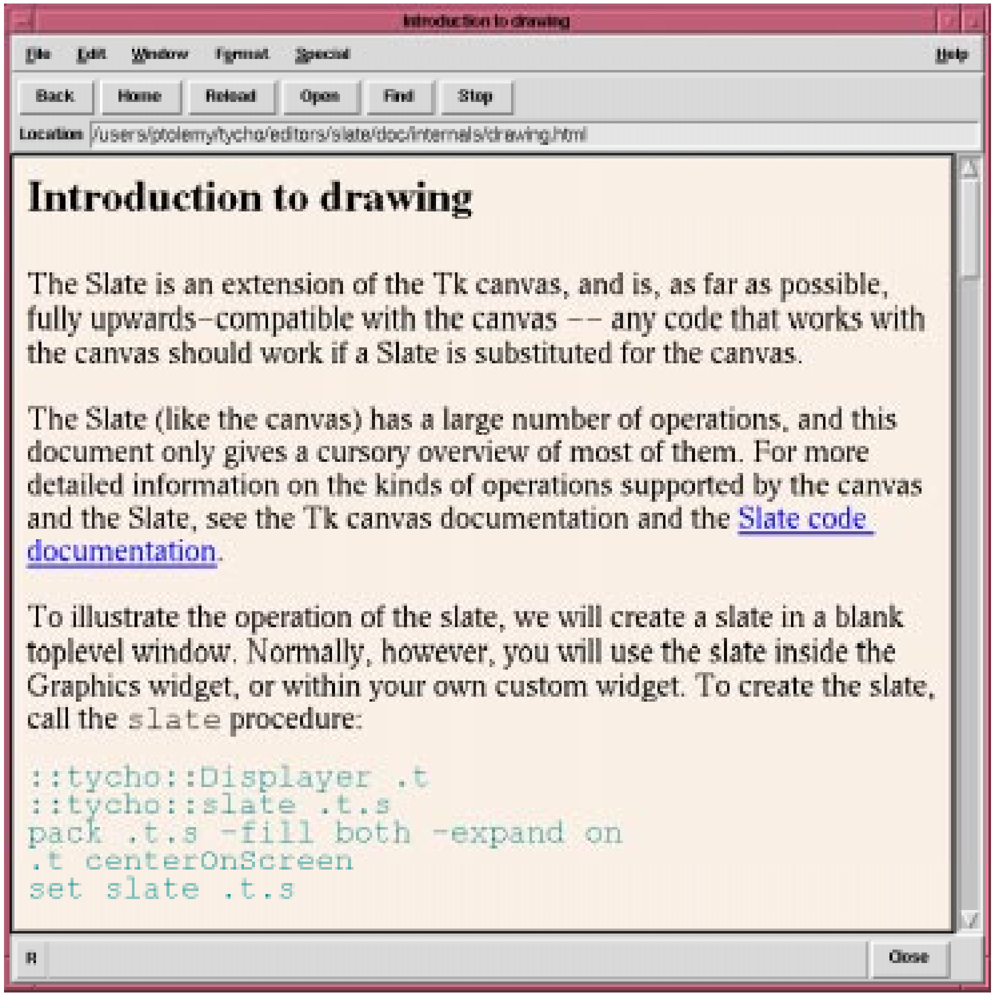
Extends the Tk canvas supporting

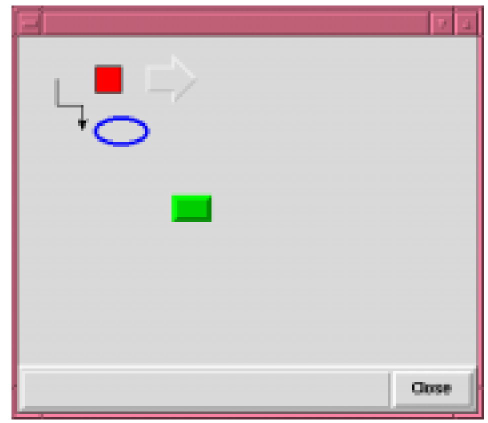
- creating complex items,
- re-using common patterns of user interaction.

There are two key uses of the Slate:

- As a higher-level canvas for building graphical displays and editors. The Slate is used this way within the Graphics class and subclasses.
- As a toolbox for rapidly building custom widgets. The Slate is used this way to create some of the custom widgets used in Ptolemy C-code-generated systems.

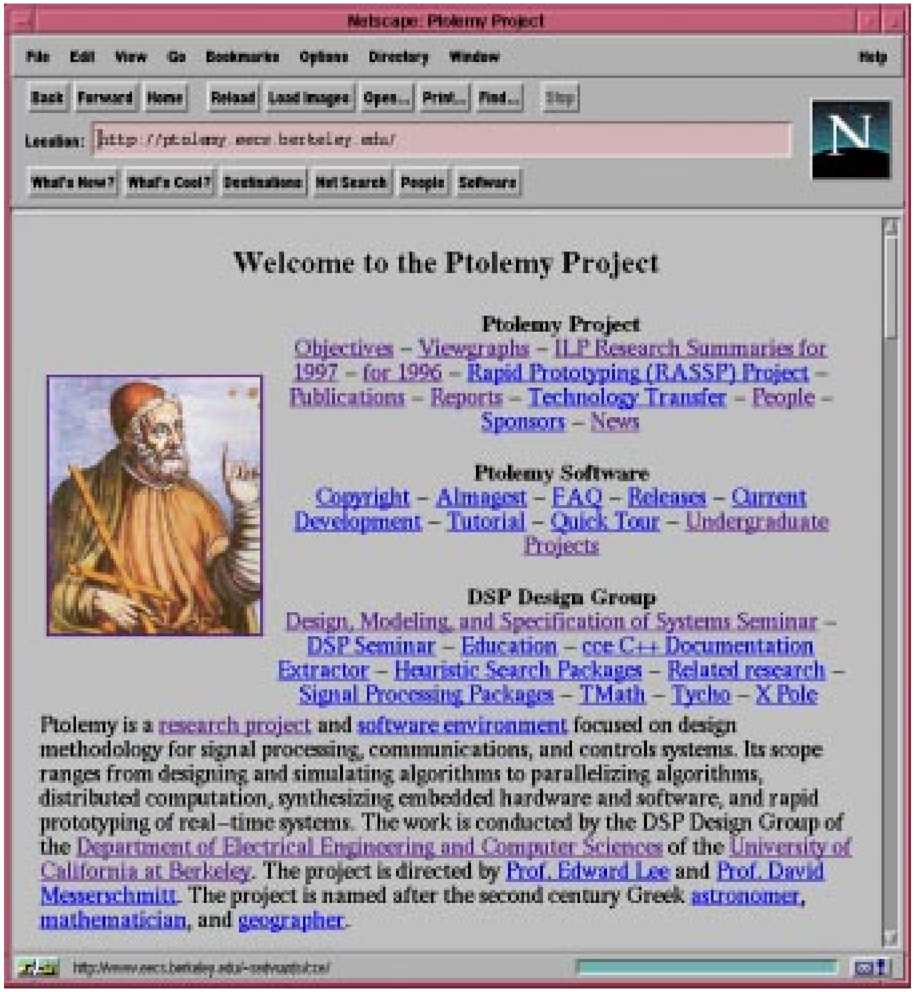
Integrated, Interactive Documentation





In the above example, clicking on the Tcl code at the bottom executes the code, creating the example slate on the right.

Further Information



- Software distribution
- Small demonstration version
- Project overview
- *The Almagest* (the manual)
- Current projects summary
- Project publications
- Keyword searching
- Project participants
- Sponsors
- Copy of the FAQ
- Newsgroup info
- Mailing lists info

http://ptolemy.eecs.berkeley.edu