Platforms

A *platform* is a set of designs (the rectangles at the right, e.g., the set of all x86 binaries).

*Model-based design* is specification of designs in platforms with useful modeling properties (e.g., Simulink block diagrams for control systems).
Platforms

Where the Action Has Been:

Giving the red platforms useful modeling properties (e.g. UML, MDA)

Getting from red platforms to blue platforms.

Platforms

Where the Action Will Be:

Giving the red platforms useful modeling properties (via models of computation)

Getting from red platforms to blue platforms.
### Design Framework

A design framework is a collection of platforms and realizable relations between platforms where at least one of the platforms is a set of physically realizable designs, and for any design in a user platform, the transitive closure of the relations from that design includes at least one physically realizable design.

In model-based design, a specification is a point in a platform with useful modeling properties.

---

### Focus on Actor-Oriented Design

(vs. trying to give useful modeling properties to program-level designs)

- **Object orientation:**
  - class name
  - data
  - methods

  What flows through an object is sequential control

  - call
  - return

- **Actor orientation:**
  - actor name
  - data (state)
  - parameters
  - ports

  What flows through an object is streams of data

  - Input data
  - Output data
Examples of Frameworks with Actor-Oriented Mechanisms

- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- SystemC + Comm Libraries (Various)
- VHDL, Verilog (Various)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- OCP, open control platform (Boeing)
- Easy5 (Boeing)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- ...

Example of Actor-Oriented Design (in this case, with a visual syntax)

Director from a library defines component interaction semantics

Large, behaviorally-polymorphic component library.

Key idea: The model of computation is part of the framework within which components are embedded rather than part of the components themselves. Thus, components need to declare behavioral properties.

Model of Computation:
- Messaging schema
- Flow of control
- Concurrency
Contrast Actor Orientation with Object Orientation

**Actor oriented**

- Text to Speech
  - Text in -> speech out

- Actor-oriented interface definition says "Give me text and I'll give you speech"

**Object oriented**

- TextToSpeech
  - initialize(): void
  - notify(): void
  - isReady(): boolean
  - getSpeech(): double[]

- OO interface definition gives procedures that have to be invoked in an order not specified as part of the interface definition.

- Identified problems with object orientation:
  - Says little or nothing about concurrency and time
  - Concurrency typically expressed with threads, monitors, semaphores
  - Components tend to implement low-level communication protocols
  - Re-use potential is disappointing

- Actor orientation offers more potential for useful modeling properties, and hence for model-based design.

Actor Orientation vs. Object Orientation

- **Object Orientation**
  - procedural interfaces
  - a class is a type (static structure)
  - type checking for composition
  - separation of interface from implementation
  - subtyping
  - polymorphism

- **Actor Orientation**
  - concurrent interfaces
  - a behavior is a type
  - type checking for composition of behaviors
  - separation of behavioral interface from implementation
  - behavioral subtyping
  - behavioral polymorphism

This is a vision of the future: Few actor-oriented frameworks fully offer this view. Eventually, all will.
Will Actor-Oriented Design Yield Better Designs?

Not necessarily.

“Why isn’t the answer UML, or XML, or IP, or something like that?”

Direct quote from a high-ranking decision maker at a large embedded systems company with global reach.

“New” is not better than “good”

Mandating use of the wrong platform is far worse than tolerating the use of multiple platforms.

Better Architecture is Enabled but not Guaranteed by Actor-Oriented Design

• Understandable concurrency
• Systematic heterogeneity
• More re-usable component libraries