Ptolemy II - Heterogeneous Modeling and Design in Java

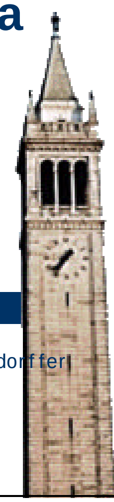
The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components.

Principal Investigator
Edward A. Lee

Technical Staff
Christopher Hylands
Mary P. Stewart

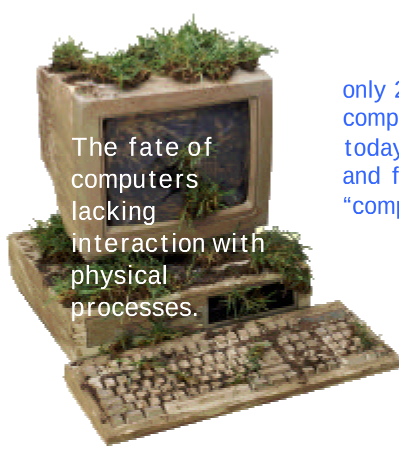
Postdocs
Bart Kienhuis

Grad Students
John Davis, II
Chamberlain Fong
Bilung Lee
Jie Liu
Xiaojun Liu
Steve Neuendorffer
Jeff Tsay
Yuhong Xiong



Embedded Systems

- Telephones
- Pagers
- Cars
- Audio equipment
- Aircraft
- Trains
- Appliances
- Toys
- Security systems
- Games
- PDAs
- Medical diagnostics
- Weapons
- Pacemakers
- Television
- Network switches
- …



only 2% of computers today are first and foremost "computers"

What we are trying to avoid:

Embedded software may end up like this as it scales up.

Poor common infrastructure. Weak specialization. Poor resource management and sharing. Poor planning.

Elegant Federation

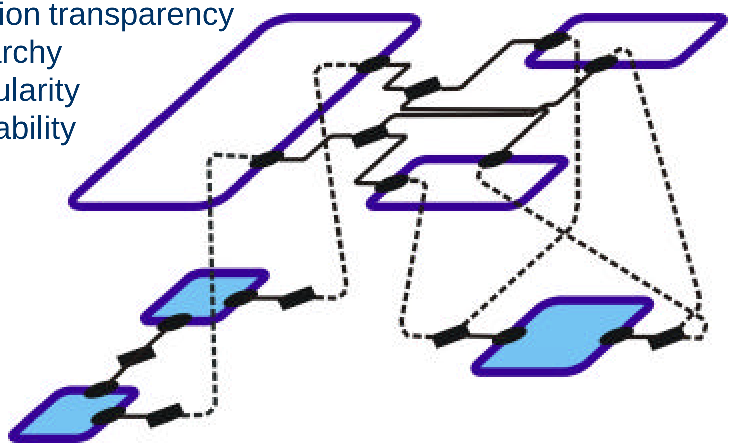
Elegant federation of heterogeneous models.

Source: *Kaplan McLaughlin Diaz*, R. Rappaport, Rockport, 1998

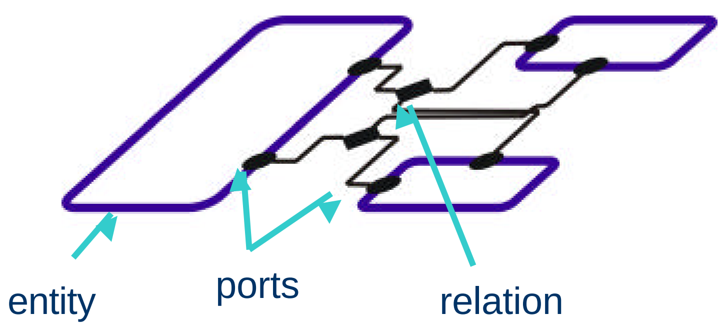
Two Rodeo Drive, Kaplan, McLaughlin, Diaz

Component-Based Design

location transparency
hierarchy
modularity
reusability

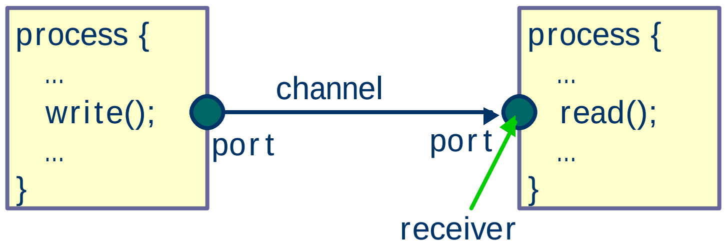


Abstract Syntax



- Ports and relations in black
- Entities in blue

One Class of Semantic Models: Producer / Consumer



- Are actors active? passive? reactive?
- Are communications timed? synchronized? buffered?

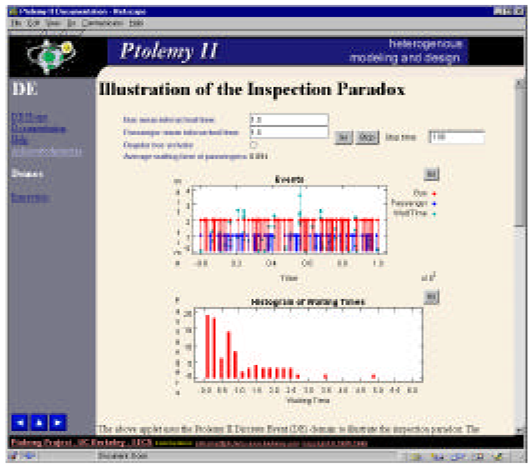
Domains – Provide semantic models for component interactions

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

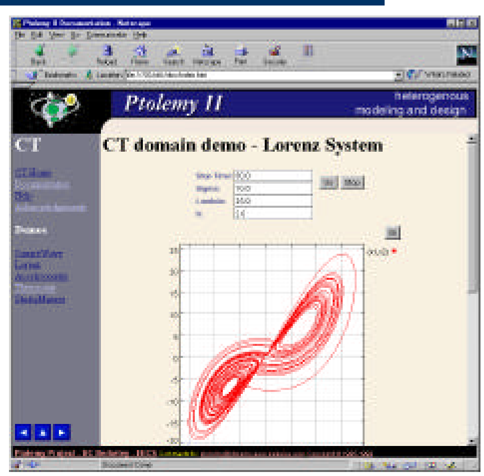
Discrete-Event Modeling

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.

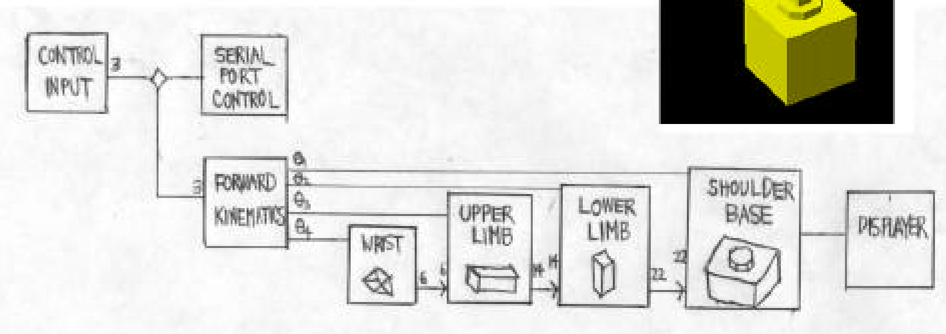


Continuous-Time Modeling

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).



DT Diagram for Robotic Arm Control



- SDF graph is used instead of an object hierarchy tree
- 4 degrees of freedom (5 DOF if including gripper)
- angles and polygon vertices are used as tokens

What is a Domain

The definition of the interaction of components, and the software that supports this interaction.

Multi-domain modeling means:

- Hierarchical composition
  - heterogeneous models allowed
- Domains can be specialized
  - avoid creeping featurism
  - enable verification
- Data replication in OCP/Boldstroke is another domain
  - separation of communication mechanisms.

Ptolemy II – Our Software Laboratory

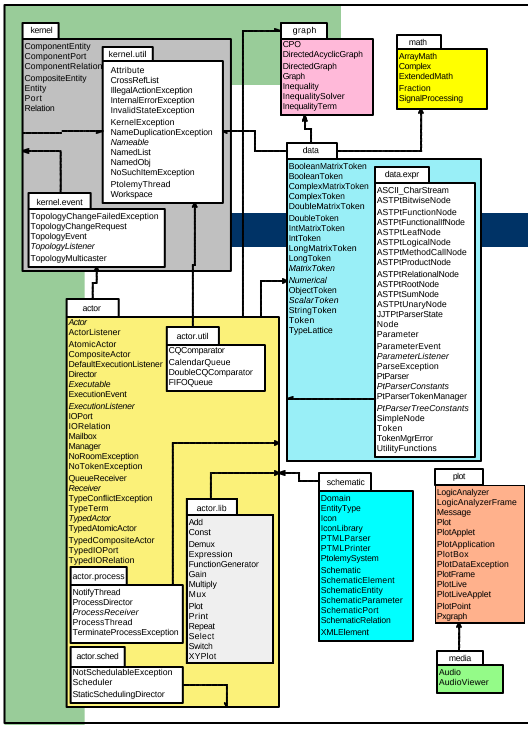
Ptolemy II –

- Java based, network integrated
- Many domains implemented
- Multi-domain modeling
- XML syntax for persistent data
- Block-diagram GUI
- Extensible type system
- Code generator on the way

http://ptolemy.eecs.berkeley.edu

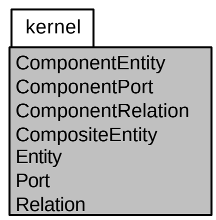
Embedded Software in Java ?!?!?!?!?

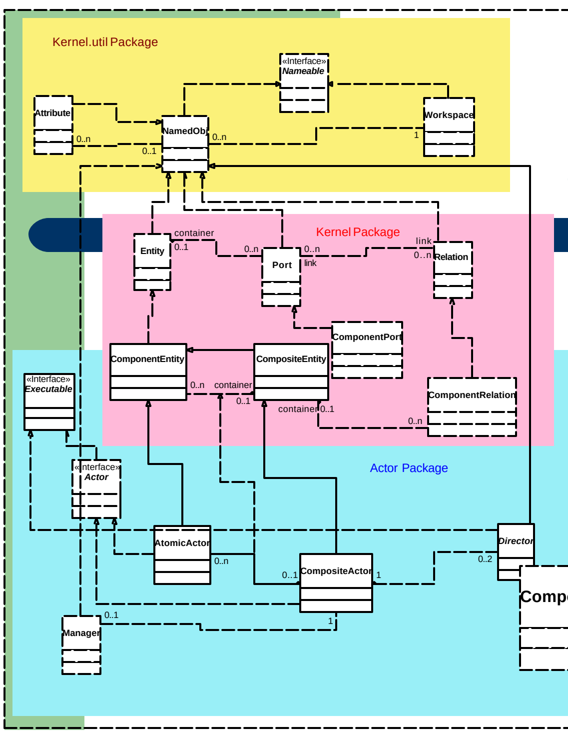
- Choosing the right design method has far more impact than faster software
- Multi-domain design permits using the best available modeling techniques
- Threads, objects, and UI infrastructure helps with both.
- Network integration of Java promotes sharing of modeling methods .
- Transportable code allows for service discovery and ad-hoc federation
- Java performance and infrastructure is rapidly improving.



Ptolemy II Packages

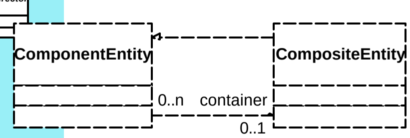
- kernel (clusterd graphs)
- actor (executable models)
- data (tokens, expressions)
- schematic (API for UIs)
- graph (graph algorithms)
- math (math algorithms)
- plot (plotting utilities)



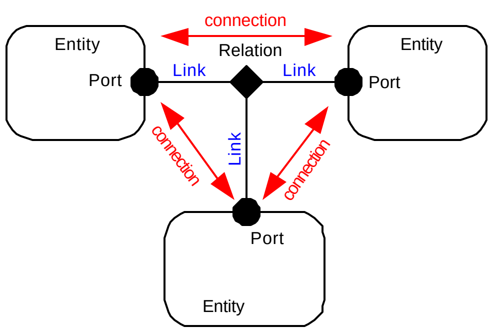


Ptolemy II Key Classes

UML static structure diagram for the key classes in the kernel, kernel.util, and actor packages.

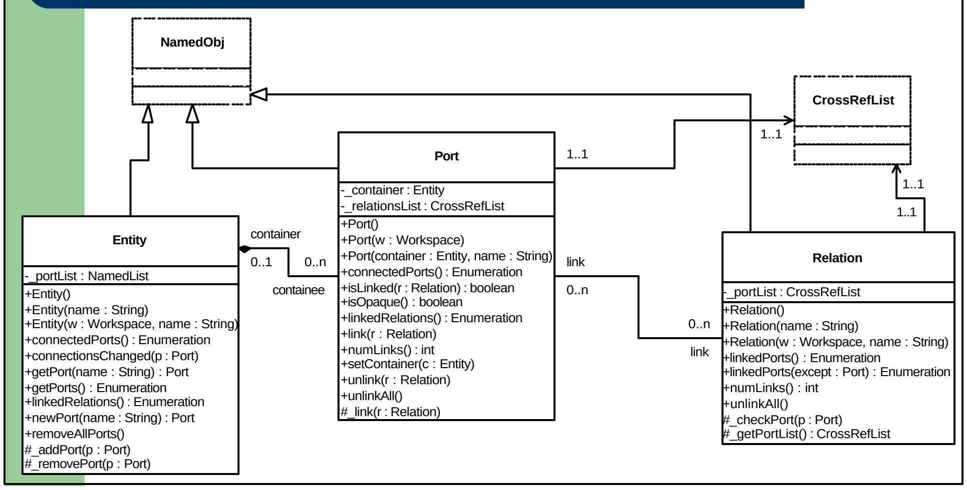


Kernel Package

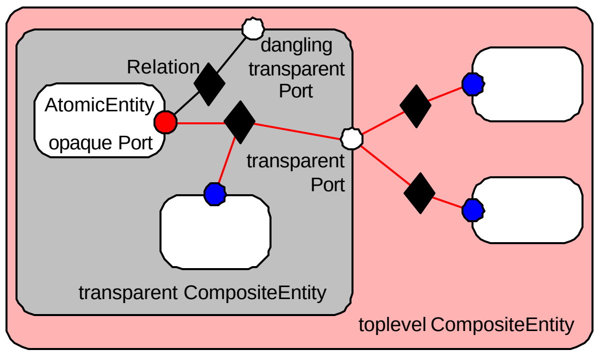


The Ptolemy II kernel provides an *abstract syntax* - clustered graphs - that is well suited to a wide variety of domains, ranging from state machines to process networks. Here is a simple graph with three interrelated entities.

Basic Kernel Classes



Clustering

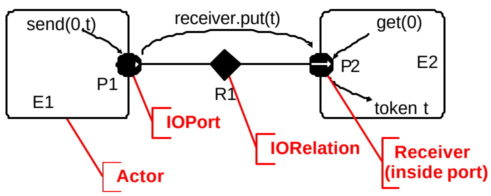


Composite entities and ports in Ptolemy II provide a simple and powerful, domain-independent abstraction mechanism

The ports deeply connected to the red port are the blue ones.

Actor Package

Basic Transport:

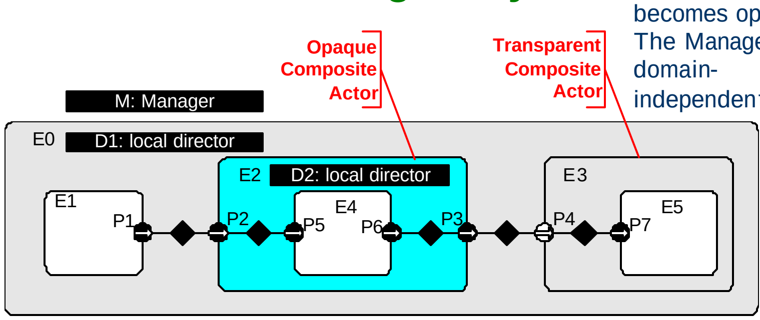


Services
- broadcast
- multicast
- busses
- cacheing topology info
- clustering
- parameterization
- typing
- polymorphism

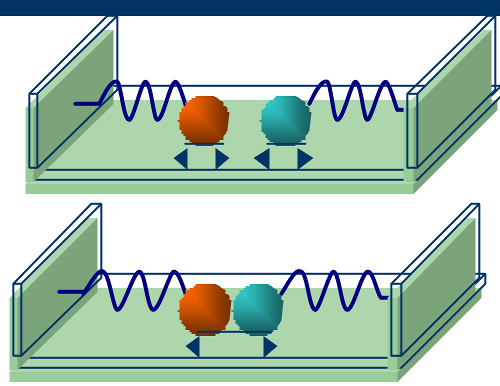
Manager and Directors

Hierarchical Heterogeneity:

Directors are domain-specific. A composite actor with a director becomes opaque. The Manager is domain-independent.

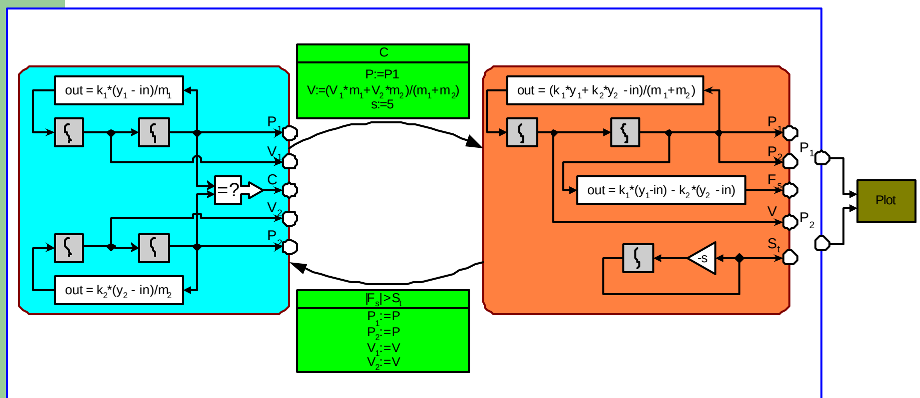


Example: Sticky Masses

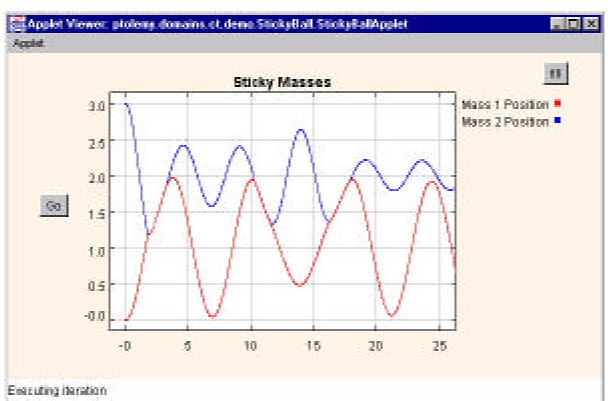


The stickiness is exponentially decaying with respect to time.

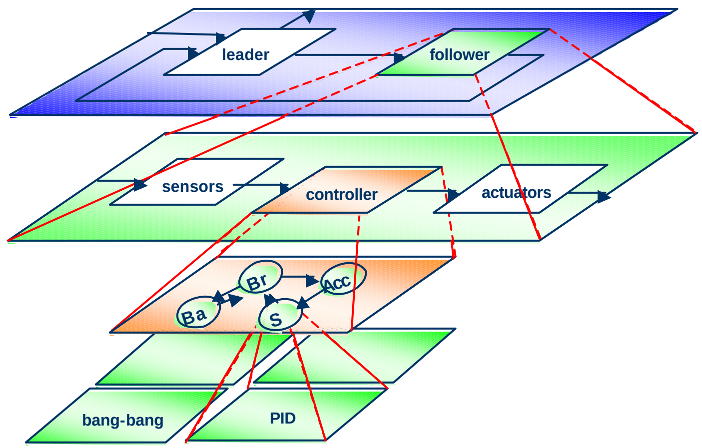
Sticky Masses: Block Diagram



Sticky Masses: Simulation



Hierarchical View



Mutations

The kernel.event package provides support for

- Queueing requests for topology changes
- Processing requests for topology changes
- Registering listeners
- Notifying listeners of changes

Thus, models with dynamically changing topologies are cleanly supported, and the director in each domain can control when mutations are implemented.

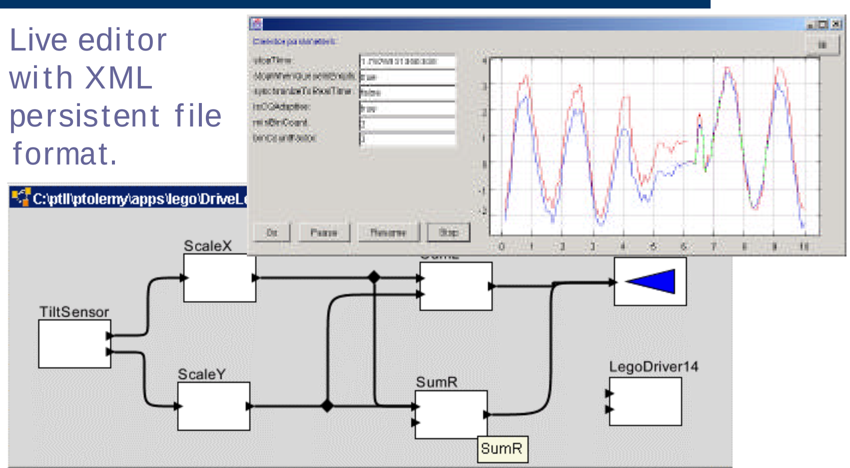
Creating a Model

- ✍ Pick one or more domains
- ✍ Choose applet or application
- ✍ Choose Vergil, MoML, or Java code
- ✍ Design control interface
- ✍ Soon: Choose distribution architecture

Ptolemy II uses features in JDK 1.2, and hence requires use of the Java plug-in with current released browsers.

Vergil – An Extensible Visual Editor

Live editor with XML persistent file format.



HTML

Internet explorer and Netscape have different plug-in architectures :-(

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
   width="700"
   height="300"
   codebase="http://java.sun.com/products/plugin/1.2/jinstall-12-win32.cab#Version=1,2,0,0">
<PARAM NAME="code" VALUE="doc.tutorial.TutorialApplet.class">
<PARAM NAME="codebase" VALUE="../..">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2"
   width="700"
   height="300"
   code="doc/tutorial/TutorialApplet.class"
   codebase="../.."
   pluginspage="http://java.sun.com/products/plugin/1.2/plugin-install.html">
</COMMENT>
<NOEMBED>
No JDK 1.2 support for applet!
</NOEMBED>
</EMBED>
</OBJECT>
```

Simple Applet – Directly in Java

```
package doc.tutorial;
import ptolemy.domains.de.gui.DEApplet;
import ptolemy.actor.lib.Clock;
import ptolemy.actor.gui.TimedPlotter;

public class TutorialApplet extends DEApplet {
    public void init() {
        super.init();
        try {
            Clock clock = new Clock(_toplevel,"clock");
            TimedPlotter plotter =
                    new TimedPlotter(_toplevel,"plotter");
            _toplevel.connect(clock.output, plotter.input);
        } catch (Exception ex) {}
    }
}
```

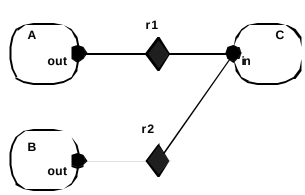
Compiling and Running

```
cd $PTII/doc/tutorial
cp TutorialApplet1.java TutorialApplet.java
javac -classpath .. TutorialApplet.java

appletviewer tutorial.htm
```

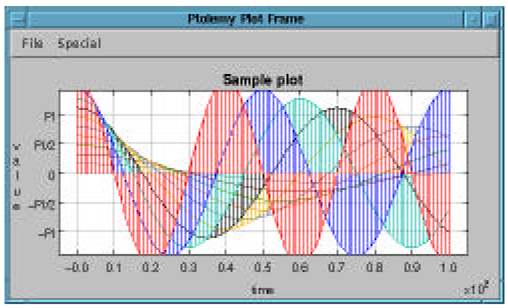
XML Model Specification (MoML)

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE model SYSTEM "DTD location">
<model class="classname">
  <entity name="A" class="classname"></entity>
  <entity name="B" class="classname"></entity>
  <entity name="C" class="classname"></entity>
  <relation name="r1"></relation>
  <relation name="r2"></relation>
  <link port="A.out" relation="r1"/>
  <link port="B.in" relation="r1"/>
  <link port="C.out" relation="r2"/>
  <link port="B.in" relation="r2"/>
</model>
```

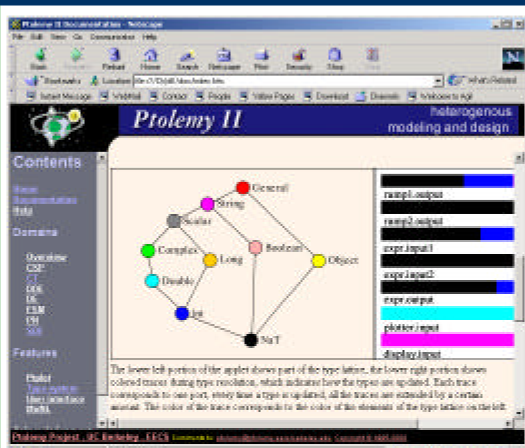


Infrastructure Support

- Expression language
- Type system
- Math package
- Graph package
- Plot package
- GUI package
- Actor library

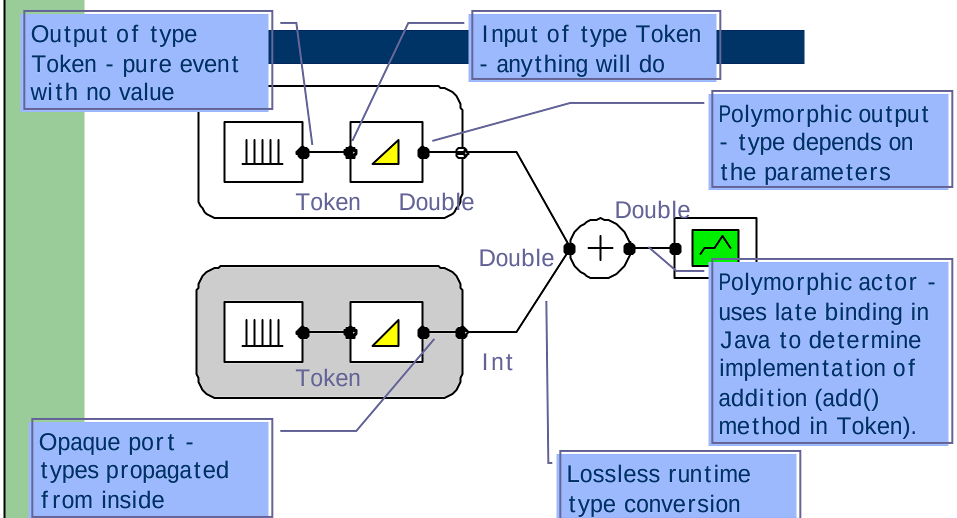


Type System Infrastructure

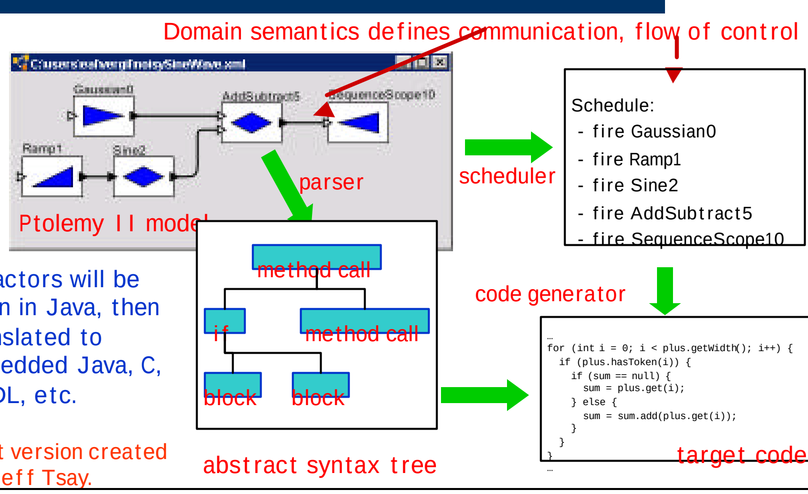


Ptolemy II has an extensible type system infrastructure with a plug-in interface for specifying a type lattice. At the left, an applet illustrates type resolution over a (simplified) type lattice representing data types exchanged between actors.

Example - Type Inference



Nascent Generator Infrastructure



All actors will be given in Java, then translated to embedded Java, C, VHDL, etc.

First version created by Jeff Tsay.

Generator Approach

- Actor libraries are built and maintained in Java
  - more maintainable, easier to write
  - polymorphic libraries are rich and small
- Java + Domain translates to target language
  - concurrent and imperative semantics
- Efficiency gotten through code transformations
  - specialization of polymorphic types
  - code substitution using domain semantics
  - removal of excess exception handling

Code transformations (on AST)

```
// Original actor source
Token t1 = in.get(0);
Token t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

specialization of Token declarations

```
// With specialized types
IntMatrixToken t1 = in.get(0);
IntMatrixToken t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

The Ptolemy II type system supports polymorphic actors with propagating type constraints and static type resolution. The resolved types can be used in optimized generated code.

See Jeff Tsay, *A Code Generation Framework for Ptolemy II*

Code transformations (on AST)

```
// With specialized types
IntMatrixToken t1 = in.get(0);
IntMatrixToken t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

Domain-polymorphic code is replaced with specialized code. Extended Java (from Titanium project) treats arrays as primitive types.

transformation using domain semantics

```
// Extended Java with specialized communication
int[][] t1 = _inbuf[0][_inOffset = (_inOffset+1)%5];
int[][] t2 = _inbuf[1][_inOffset = (_inOffset+1)%5];
_outbuf[_outOffset = (_outOffset+1)%8] = t1 + t2;
```

See Jeff Tsay, *A Code Generation Framework for Ptolemy II*

Code transformations (on AST)

```
// Extended Java with specialized communication
int[][] t1 = _inbuf[0][_inOffset = (_inOffset+1)%5];
int[][] t2 = _inbuf[1][_inOffset = (_inOffset+1)%5];
_outbuf[_outOffset = (_outOffset+1)%8] = t1 + t2;
```

convert extended Java to ordinary Java

```
// Specialized, ordinary Java
int[][] t1 = _inbuf[0][_inOffset = (_inOffset+1)%5];
int[][] t2 = _inbuf[1][_inOffset = (_inOffset+1)%5];
_outbuf[_outOffset = (_outOffset+1)%8] =
    IntegerMatrixMath.multiply(t1, t2);
```

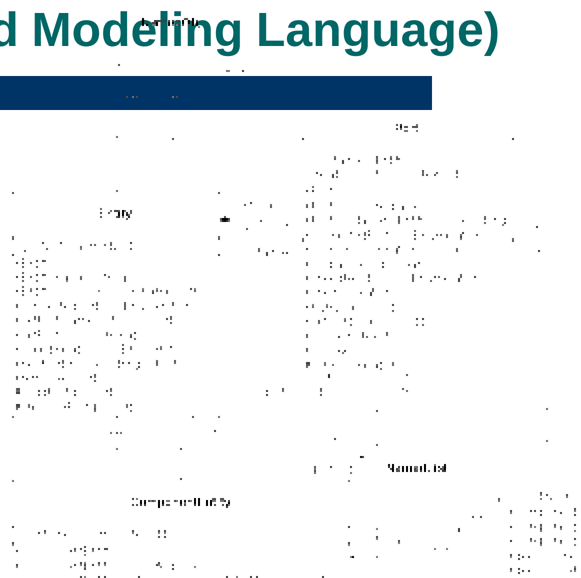
See Jeff Tsay, *A Code Generation Framework for Ptolemy II*

Software Practice

- Object models in UML
- Design patterns
- Layered software architecture
- Design and code reviews
- Design document
- Nightly build
- Regression tests
- Sandbox experimentation
- Code rating

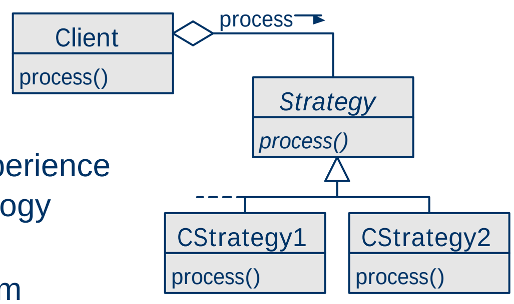
UML (Unified Modeling Language)

We make extensive use of static structure diagrams, and much less use of other UML languages.



Design patterns

- A high-level vocabulary for describing recurring patterns:
  - Strategy
  - Composite
  - Factory
  - Template method
- A way of factoring experience into concrete terminology
- We studied the most important patterns from Gamma *et al*



Design and Code Reviews

- Objective is “publishable software”
- Defined roles for participants
  - Author has the last word
- Mechanism for new group members to learn to differentiate good from bad software.

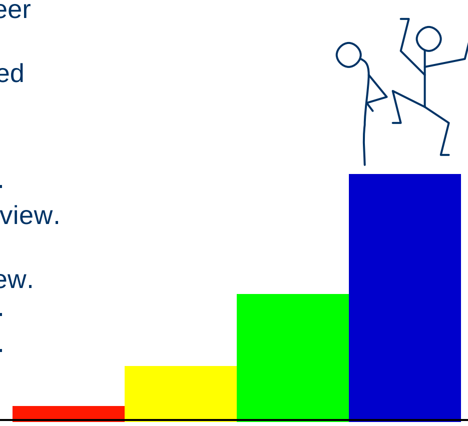
All technical reviews are based on the idea that developers are blind to some of the trouble spots in their work...

Steve McConnell

Code rating

- What is this about really?
  - *Confidence* in quality
  - *Commitment* to stability

- A simple framework for
  - quality improvement by peer review
  - change control by improved visibility
- Four confidence levels
  - Red. No confidence at all.
  - Yellow. Passed design review. Soundness of the APIs.
  - Green. Passed code review. Quality of implementation.
  - Blue. Passed final review. Backwards-compatibility assurance.



How we do a review

- Top level
  - The author announces that the package is ready for review
  - The moderator organizes and moderates the review
  - The author responds to the issues raised in the review, redesigning or reworking as necessary
  - The author announces the new rating.
- In the review
  - The *moderator* runs the meeting and keeps the discussion on track; and acts as *reader* (in our process).
  - The *reviewers* raise issues and defects
  - The *author* answers questions
  - The *scribe* notes raised issues and defects
  - *Nobody* attempts to find solutions!

Roles define and clarify responsibility

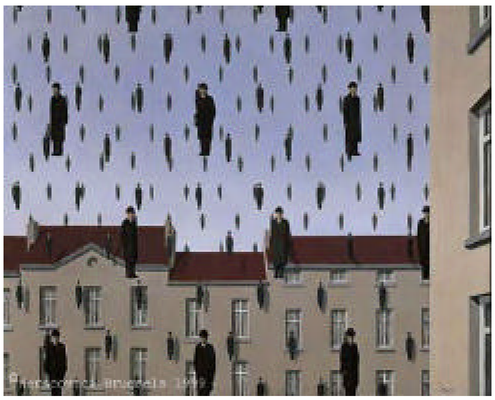
What were the review benefits?

- Students
  - better design and more confidence.
  - good feedback about documentation and naming issues
  - revealed quite a few flaws
  - an affirmation that your architecture is sound
  - encourage other people in the group to reuse code
  - forcing function to get documentation in order
  - my coding style changed
- Staff
  - exposed quite a few design flaws
  - caught lots of minor errors, and quite a few insidious errors

Design in an Abstract Universe

When choosing syntax and semantics, we can invent the "laws of physics" that govern the interaction of components.

As with any such laws, their utility depends on our ability to understand models governed by the laws.



Magritte, Gelconde