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 and Researchers
Jorn Janneck
Sonia Sachs

Grad Students
Chamberlain Fong
Jie Liu
Xiaojun Liu
Steve Neuendorffer

Embedded Systems

- Telephones
- Pagers
- Cars
- Audio equipment
- Aircraft
- Trains
- Appliances
- Toys
- Security systems
- Games
- PDAs
- Medical diagnostics
- Weapons
- Pacemakers
- Televisions
- 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.
Example: An Engine Control System

Component-Based Design
- Location transparency
- Hierarchy
- Modularity
- Reusability
- Scalability

Component & Composition

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
- Globally Asynchronous communication in Polis

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

What is a domain?

- Ontology: What is a component?
  - Constraints? Objects (data + methods)?
- Epistemology: What knowledge do components have?
  - Time? Name spaces? Signals? State?
- Protocols: How do components communicate?
  - Rendezvous? Message passing? Continuous-time signals?
  - Streams? Method calls?
- Lexicon: What do components communicate?
  - Objects? Transfer of control? Data structures? ASCII text?

Ptolemy II – Our Software Laboratory

- 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

Ptolemy II Provides

- Component Architecture
- Execution Interface
- Infrastructure Support
  - Expression language
  - Type system
  - Math package
  - Graph package
  - Plot package
  - GUI package
  - Actor library
Example: Pendulum

Ptolemy II Infrastructure

Ptolemy II Packages
- kernel (clustered graphs)
- actor (executable models)
- data (tokens, expressions)
- moml (persistent model)
- graph (graph algorithms)
- math (math algorithms)
- plot (plotting utilities)

Basic Kernel Classes
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.

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

The kernel.event package provides support for:

- Queuing 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.

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

Services
- Message passing
- Executable interface
- Topology caching
- Clustering
- Parameterization
- Typing
- Polymorphism
Executable Interface

Manager and Directors

Hierarchical Heterogeneity:

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

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.

Execution Sequencing

- Managers and Directors
- Preinitialize
- Initialize
- Fire
- Postfire
- Wrapup
- Iteration
Creating a Model

- Choose applet or application
- Choose Vergil, MoML, or Java code
- Pick one or more domains
- Drag and drop actors and make connections
- Design control interface
- Soon: Choose distribution architecture

Ptolemy II uses features in JDK 1.3, 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.

XML Model Specification (MoML)

```xml
<?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>
```

HTML

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="700"
height="300"
codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-win32.cab#Version=1,3,0,0">
<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=1.3"
width="700"
height="300"
code="doc/tutorial/TutorialApplet.class"
codebase="../..">
</COMMENT>
<NOEMBED>
No JDK 1.3 support for applet!
</NOEMBED>
</OBJECT>
```

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/1.3/jinstall-13-win32.cab#Version=1,3,0,0">
<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=1.3"
width="700"
height="300"
code="doc/tutorial/TutorialApplet.class"
codebase="../..">
</comment>
</object>
```
Simple Applet – Directly in Java

```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

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

Example: Sticky Masses

The stickiness is exponentially decaying with respect to time.

Sticky Masses: Block Diagram
Sticky Masses: Simulation

Nascent Generator Infrastructure

Domain semantics defines communication, flow of control

Ptolemy II model

Schedule:
- fire Gaussian0
- fire Ramp1
- fire Sine2
- fire AddSubtract5
- fire SequenceScope10

Parser

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

Experimental Result: OrthogonalCom system

- Interpreted performance increase between 2.5x and 10x, depending on JIT
- Statically compiled version has additional 2x performance gain with GCJ
- Code size reduction around 10x
e-Ptolemy: A Distributed Embedded Software Framework

- Location Transparent Components
- Publish and Subscribe Message Passing
- Federation of Networked Embedded Systems

Location Transparent Components

Publishing and Subscribe Message Passing

Federation of Networked Embedded Systems

Fault detection

Controller

Communication

Sensors

UI

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.