Concurrent Models of Computation for Embedded Software

Edward A. Lee
Professor, UC Berkeley
EECS 290n – Advanced Topics in Systems Theory
Fall, 2004

Copyright © 2004, Edward A. Lee, All rights reserved

Lecture 5: Extending Ptolemy II

Background for Ptolemy II

Gabriel (1986-1991)
- Written in Lisp
- Aimed at signal processing
- Synchronous dataflow (SDF) block diagrams
- Parallel schedulers
- Code generators for DSPs
- Hardware/software co-simulators

Ptolemy Classic (1990-1997)
- Written in C++
- Multiple models of computation
- Hierarchical heterogeneity
- Dataflow variants: BDF, DDF, PN
- C/VHDL/DSP code generators
- Optimizing SDF schedulers
- Higher-order components

Ptolemy II (1996-2022)
- Written in Java
- Domain polymorphism
- Multithreaded
- Network integrated
- Modal models
- Sophisticated type system
- CT, HDF, CI, GR, etc.

Each of these served us, first-and-foremost, as a laboratory for investigating design.

PPlot (1997-??)
- Java plotting package

Tycho (1996-1998)
- Itcl/Tk GUI framework

Diva (1998-2000)
- Java GUI framework

All open source.
All truly free software (cf. FSF).
Framework Infrastructure that Supports Diverse Experiments with Models of Computation

Concurrency management supporting dynamic model structure.

Director from a library defines component interaction semantics.

Large, domain-polymorphic component library.

Visual editor supporting an abstract syntax.

The Basic Abstract Syntax

- Actors
- Attributes on actors (parameters)
- Ports in actors
- Links between ports
- Width on links (channels)
- Hierarchy

Concrete syntaxes:
- XML
- Visual pictures
- Actor languages (Cal, StreamIT, …)
MoML
XML Schema for this Abstract Syntax

Ptolemy II designs are represented in XML:

```xml
...<entity name="FFT" class="ptolemy.domains.sdf.lib.FFT">
  <property name="order" class="ptolemy.data.expr.Parameter" value="order">
    ...
  </property>
  <port name="input" class="ptolemy.domains.sdf.kernel.SDFIOPort">
    ...
  </port>
...
</entity>
...<link port="FFT.input" relation="relation"/>
<link port="AbsoluteValue2.output" relation="relation"/>
...```

Hierarchy - Composite Components
Kernel Classes
Support the Abstract Syntax

Concurrency Management Supporting Dynamic Model Structure

Changes to a model while the model is executing:
- Change parameter values
- Change model structure

How can this be made safe?
- Workspace class
- ChangeRequest class
- stopFire() method

Can dynamically modify the model while it executes... safely.
try {
    _workspace.getReadAccess();
    _actions depending on model structure
} finally {
    _workspace.doneReading();
}

Many threads can have read access at the same time. Only one thread can have write access, and only if no other thread has read access.

Specialized wait(Object) method releases the locks during the wait().
When to Execute Change Requests

In many models of computation, there is a natural time: between iterations.

In PN, this is not a trivial question...

- All threads must be stopped (blocked)
  - On reads
  - On writes to full buffers
  - Or block themselves with a wait()

- What happens when the model structure changes during a call to `get()`?

ProcessThread with Pauses for Mutations

```java
while(iterate) {
    if (_director.isStopFireRequested()) {
        synchronized (_director) {
            _director._actorHasStopped();
            while (_director.isStopFireRequested()) {
                try {
                    workspace.wait(_director);
                } catch (InterruptedException ex) {
                    break;
                }
            }
            _director._actorHasRestarted();
        }
    }
    boolean iterate = true;
    while (iterate) {
        if (_actor.prefire()) {
            _actor.fire();
            iterate = _actor.postfire();
        }
    }
}
```

Specialized `wait()` method releases workspace locks while the thread is suspended.
Abstract Semantics of *Actor-Oriented* Models of Computation

Actor-Oriented Models of Computation that we have implemented:

- dataflow (several variants)
- process networks
- distributed process networks
- Click (push/pull)
- continuous-time
- CSP (rendezvous)
- discrete events
- distributed discrete events
- synchronous/reactive
- time-driven (several variants)
- ...

Object Model for Executable Components

```java
// Interface: Executable
+fire()
+initialize()
+postfire() : boolean
+prefire() : boolean
+preinitialize()
+stopFire()
+terminate()
+wrapup()

// Interface: Actor
+getDirector() : Director
+getExecutiveDirector() : Director
+getManager() : Manager
+inputPortList() : List
+newReceiver() : Receiver
+outputPortList() : List

ComponentEntity
  0..1
  0..n

CompositeEntity

CompositeActor

Director

AtomicActor
```
Object Model (Simplified) for Communication Infrastructure

Object-Oriented Approach to Achieving Behavioral Polymorphism

These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is supplied by the director, not by the component.

Recall: Behavioral polymorphism is the idea that components can be defined to operate with multiple models of computation and multiple middleware frameworks.
Extension Exercise

Build a director that subclasses PNDirector to allow ports to alter the “blocking read” behavior. In particular, if a port has a parameter named “tellTheTruth” then the receivers that your director creates should “tell the truth” when hasToken() is called. That is, instead of always returning true, they should return true only if there is a token in the receiver.

Parameterizing the behavior of a receiver is a simple form of communication refinement, a key principle in, for example, Metropolis.

Implementation of the New Model of Computation

package experiment;
import ...

public class NondogmaticPNDirector extends PNDirector {
    public NondogmaticPNDirector(CompositeEntity container, String name)
        throws IllegalActionException, NameDuplicationException {
        super(container, name);
    }
    public Receiver newReceiver() {
        return new FlexibleReceiver();
    }

    public class FlexibleReceiver extends PNQueueReceiver {
        public boolean hasToken() {
            IOPort port = getContainer();
            Attribute attribute = port.getAttribute("tellTheTruth");
            if (attribute == null) {
                return super.hasToken();
            }
            // Tell the truth...
            return _queue.size() > 0;
        }
    }
}
Ptolemy II Software Architecture
Built for Extensibility

Ptolemy II packages have carefully constructed dependencies and interfaces.

Hierarchical Heterogeneity

Directors are domain-specific. A composite actor with a director becomes opaque. The Manager is domain-independent.
Polymorphic Components - Component Library Works Across Data Types and Domains

**Data polymorphism:**
- Add numbers (int, float, double, Complex)
- Add strings (concatenation)
- Add composite types (arrays, records, matrices)
- Add user-defined types

**Behavioral polymorphism:**
- In dataflow, add when all connected inputs have data
- In a time-triggered model, add when the clock ticks
- In discrete-event, add when any connected input has data, and add in zero time
- In process networks, execute an infinite loop in a thread that blocks when reading empty inputs
- In CSP, execute an infinite loop that performs rendezvous on input or output
- In push/pull, ports are push or pull (declared or inferred) and behave accordingly
- In real-time CORBA, priorities are associated with ports and a dispatcher determines when to add

By not choosing among these when defining the component, we get a huge increment in component re-usability. But how do we ensure that the component will work in all these circumstances?
Shared Infrastructure Modularity Mechanisms

This model illustrates the mechanisms in Ptolemy II for defining classes and subclasses with inheritance.

This actor is a class definition, indicated by the blue halo. It is ignored by the director and serves as a declaration. To create an instance of this class, right click on the class definition and select "Create Instance" (or type Ctrl-N). To see the class definition, look inside.

This is an instance of the above class definition. Look inside to see the subclass definition.

This is an instance of the base class for the above class definition.

More Shared Infrastructure: Hierarchical Heterogeneity and Modal Models

The objects highlighted in pink are defined in the superclass. Such objects cannot be removed in the derived class. Their parameters can be changed, however. This implies that they can be reused and can be assigned custom icons. To exercise the superclasses, right click on the background and select "Open Base Class".

The example Ptolemy II model: hybrid control system.
Branding

Ptolemy II configurations are Ptolemy II models that specify
- welcome window
- help menu contents
- library contents
- File->New menu contents
- default model structure
- etc.

A configuration can identify its own “brand” independent of the “Ptolemy II” name and can have more targeted objectives.

An example is HyVisual, a tool for hybrid system modeling. VisualSense is another tool for wireless sensor network modeling.

Ptolemy II Extension Points

- Define actors
- Interface to foreign tools (e.g. Python, MATLAB)
- Interface to verification tools (e.g. Chic)
- Define actor definition languages
- Define directors (and models of computation)
- Define visual editors
- Define textual syntaxes and editors
- Packaged, branded configurations

All of our “domains” are extensions built on a core infrastructure.
Example Extension: VisualSense

- Branded
- Customized visualization
- Customized model of computation (an extension of DE)
- Customized actor library
-Motivated some extensions to the core (e.g. classes, icon editor).

Example Extensions: Self-Repairing Models

Concept demonstration built together with Boeing to show how to write actors that adaptively reconstruct connections when the model structure changes.

DE Director

This model is a simple example of a self-repairing model. The SmartSender actor, if not connected, will search for an unused input port to connect to and will establish a connection.

SmartSender
SmartSender2
SmartSender3
SmartSender4
MonitorValue
MonitorValue2
MonitorValue3
MonitorValue4

If you have source code installed, look inside the SmartSender to see how this is realized. Or get documentation for a more detailed explanation.

Author: Edward A. Lee

Try running the model, and then deleting the connections while the model is running. The actors will reconnect. Then try creating new instances of SmartSender and MonitorValue by clicking and pasting. You can do this while the model is running. If you drag in new actors from the library, they can also supply input ports.
Example Extensions

Python Actors and Cal Actors

Cal is an experimental language for defining actors that is analyzable for key behavioral properties.

This model demonstrates the use of function closures inside a CAL actor.

The PrimeSieve actor uses nested function closures to realize the Sieve of Eratosthenes, a method for finding prime numbers. Its state variable, "filter," contains the current filter function. If it is "false" a new prime number has been found, and a new filter function will be generated.

The PrimeSieve actor expects an ascending sequence of natural numbers, starting from 2, as input.

Example Extensions

Using Models to Control Models

This is an example of a "higher-order component," or an actor that references one or more other actors.
Examples of Extensions
Mobile Models

Model-based distributed task management:

PushConsumer actor receives pushed data provided via CORBA, where the data is an XML model of a signal analysis algorithm.

MobileModel actor accepts a StringToken containing an XML description of a model. It then executes that model on a stream of input data.

Authors:
Yang Zhao
Steve Neuendorffer
Xiaojun Liu

Examples of Extensions
Hooks to Verification Tools

New component interfaces to Chic verification tool

Authors:
Arindam Chakrabarti
Eleftherios Matsikoudis
Examples of Extensions
Hooks to Verification Tools

Synchronous assume/guarantee interface specification for Block1

Examples of Extensions
Hooks to Verification Tools
Summary

Ptolemy II provides considerable infrastructure for experimenting with models of computation.