References

- Ptolemy project home page:
  http://ptolemy.eecs.berkeley.edu/

- Tutorial: Building Ptolemy II Models Graphically:
  http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-129.html

- Latest release:
  http://ptolemy.eecs.berkeley.edu/ptolemyll/ptlllatest/

- Latest version in the SVN repository:
  http://chess.eecs.berkeley.edu/ptexternal/
Outline

- Simple model building
  - Writing actors
  - Writing directors
Building Models
Outline

- Simple model building
- Writing actors
- Writing directors
public class Ptolemnizer extends TypedAtomicActor {
    public Ptolemnizer(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);
        input = new TypedIOPort(this, "input");
        input.setTypeEquals(BaseType.STRING);
        input.setInput(true);
        output = new TypedIOPort(this, "output");
        output.setTypeEquals(BaseType.STRING);
        output.setOutput(true);
    }
    public TypedIOPort input;
    public TypedIOPort output;
    public void fire() throws IllegalActionException {
        if (input.hasToken(0)) {
            Token token = input.get(0);
            String result = ((StringToken)token).stringValue();
            result = result.replaceAll("t", "pt");
            output.send(0, new StringToken(result));
        }
    }
}
Object Model for Executable Components

- «Interface» Executable
  - +fire()
  - +initialize()
  - +postFire() : boolean
  - +preinitialize()
  - +stopFire()
  - +terminate()
  - +wrapup()

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

- ComponentEntity

- CompositeEntity

- Director

- AtomicActor

- CompositeActor
Outline

- Simple model building
- Writing actors
- Writing directors
Object Model (Simplified) for Communication Infrastructure

- **IOPort**
  - 0..1 relationship with NoRoomException
  - 0..n relationship with Receiver

- **Receiver**
  - +get() : Token
  - +getContainer() : IOPort
  - +hasRoom() : boolean
  - +hasToken() : boolean
  - +put(t : Token)
  - +setContainer(port : IOPort)

- **Mailbox**
- **ProcessReceiver**
- **QueueReceiver**
- **DEReceiver**
- **SDFReceiver**

- **CTRReceiver**
- **CSPReceiver**
- **PNReceiver**
- **FIFOQueue**
- **ArrayFIFOQueue**
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.
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.
package doc.tutorial;
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;
    }
}

Implementation of the NondogmaticPNDirector
Using It

Model of a sensor sensing a sinusoidal signal with the specified frequency and phase at the specified sampling frequency. This composite actor simulates real-time behavior by sleeping the amount of time given by the samplingPeriod (in seconds) before producing an output.
Extension Exercise 2

Build a director that subclasses Director and allows different receiver classes to be used on different connections. This is a form of what we call “amorphous heterogeneity.”
Implementation of the AmorphousDirector

```java
package doc.tutorial;
import ...

public class AmorphousDirector extends Director {
    public AmorphousDirector(CompositeEntity container, String name)
        throws IllegalActionException, NameDuplicationException {
        super(container, name);
    }
    public Receiver newReceiver() {
        return new DelegatingReceiver();
    }

    public class DelegatingReceiver extends AbstractReceiver {
        private Receiver _receiver;
        public DelegatingReceiver() {
            super();
            _receiver = new SDFReceiver();
        }
        public DelegatingReceiver(IOPort container) throws IllegalActionException {
            super(container);
            _receiver = new SDFReceiver(container);
        }

        public void clear() throws IllegalActionException {
            IOPort container = getContainer();
            if (container != null) {
                StringParameter receiverClass = (StringParameter)
                    container.getAttribute("receiverClass", StringParameter.class);
                if (receiverClass != null) {
                    String className = ((StringToken)receiverClass.getToken()).stringValue();
                    try {
                        Class desiredClass = Class.forName(className);
                        _receiver = (Receiver)desiredClass.newInstance();
                    } catch (Exception e) {
                        throw new IllegalActionException(container, e,
                            "Invalid class for receiver: " + className);
                    }
                }
            }
            _receiver.clear();
        }

    }

    public Token get() throws NoTokenException {
        return _receiver.get();
    }
    ...
```
Using It

AmorphousDirector

Ramp → Repeat → SequencePlotter

Edit parameters for input

receiverClass: ptolemy.actor.QueueReceiver

Edit parameters for input

receiverClass: ptolemy.actor.MailboxReceiver

Exception

Invalid class for receiver: ptolemy.actor.MailboxReceiver
in AmorphousExample.SequencePlotter.input
Because:
ptolemy.actor.MailboxReceiver

Dismiss Display Stack Trace
Extension Exercise 3

Build a director that fires actors in left-to-right order, as they are laid out on the screen.
Implementation of the LeftRightDirector
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 Extensions
Python Actors, Cal Actors, MATLAB 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.
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 Extensions
Using Models to Control Models

This model illustrates the use of a "run composite actor" component. That component contains another Ptolemy II model. Each time it fires, it performs a complete execution of that other Ptolemy II model, rather than just one firing as would be typical of a composite actor.

Look inside this actor to see the model that is repeatedly executed.

This model generates Lissajous figures, which are plots of one sinusoid vs. another. On each execution, it generates one figure.

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

Lee, Berkeley 23
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.
VisualSense extends the Ptolemy II discrete-event domain with communication between actors representing sensor nodes being mediated by a channel, which is another actor.

The example at the left shows a grid of nodes that relay messages from an initiator (center) via a channel that models a low (but non-zero) probability of long range links being viable.
Viptos: Extension of VisualSense with Programming of TinyOS nodes

Viptos demo: Multihop routing (Surge)

Physical environment Simulation (with visualization of routing tree)

Hardware


Software Code generation: Models to nesC.
Another Extension: HyVisual – Hybrid System Modeling Tool Based on Ptolemy II

HyVisual was first released in January 2003.
Another Extension: Kepler: Aimed at Scientific Workflows

Key capabilities added by Kepler:
- Database interfaces
- Data and actor ontologies
- Web service wrappers
- Grid service wrappers
- Semantic types
- Provenance tracking
- Authentication framework

This example shows the use of data ontologies and database wrappers.
Kepler as an Interface to the Grid

CPES Fusion Simulation Workflow

- **Fusion Simulation Codes**: (a) GTC; (b) XGC with M3D
  - e.g. (a) currently 4,800 (soon: 9,600) nodes Cray XT3; 9.6TB RAM; 1.5TB simulation data/run
- **GOAL**:
  - automate remote simulation job submission
  - continuous file movement to analysis cluster for dynamic visualization & simulation control
  - … with runtime-configurable observables

Overall architect (& prototypical user): Scott Klasky (ORNL)
WF design & implementation: Norbert Podhorszki (UC Davis)
Leverage: Kepler is a Team Effort

Resurgence

Cipres

NLADR

LOOKING

Griddles

SKIDL

SRB

Other contributors:
- Chesire (UK Text Mining Center)
- DART (Great Barrier Reef, Australia)
- National Digital Archives + UCSD-TV (US)
- ...

Contributor names and funding info are at the Kepler website: http://kepler-project.org

Lee, Berkeley 30
Getting More Information: Documentation

PTOLEMY II
HETEROGENEOUS CONCURRENT MODELING AND DESIGN IN JAVA

Volume 1: User-Oriented
Volume 2: Developer-Oriented
Volume 3: Researcher-Oriented

Tutorial information: http://ptolemy/conferences/07/tutorial.htm