Exploring Models of Computation with Ptolemy II[*]

Christopher Brooks
EECS Department
University of California, Berkeley
Berkeley, CA, USA
cxh@eecs.berkeley.edu

Edward A. Lee
EECS Department
University of California, Berkeley
Berkeley, CA, USA
eal@eecs.berkeley.edu

Stavros Tripakis
EECS Department
University of California, Berkeley
Berkeley, CA, USA
stavros@eecs.berkeley.edu

ABSTRACT

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. A major problem area being addressed is the use of heterogeneous mixtures of models of computation.

Ptolemy II takes a component view of design, in that models are constructed as a set of interacting components. A model of computation governs the semantics of the interaction, and thus imposes an execution-time discipline. Ptolemy II has implementations of many models of computation including Synchronous Data Flow, Kahn Process Networks, Discrete Event, Continuous Time, Synchronous/ Reactive and Modal Models

This hands-on tutorial explores how these models of computation are implemented in Ptolemy II and how to create new models of computation such as a "non-dogmatic" Process Networks example and a left-to-right execution policy example.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: *real-time and embedded systems*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*cyber-physical systems*; I.6.5 [**Simultation and Modeling**]: Model Development[models cyber-physical interactions]; J.7 [**Computers in Other Systems**][computer-based systems]

General Terms

Design, Experimentation

[*]This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #CCR-0225610 (ITR), #0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), #0931843 (Action-Webs) and #0720841 (CSR-CPS)), the U.S. Army Research Office (ARO #W911NF-07-2-0019), the U.S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyc) and the following companies: Agilent, Bosch, National Instruments, Thales, and Toyota.

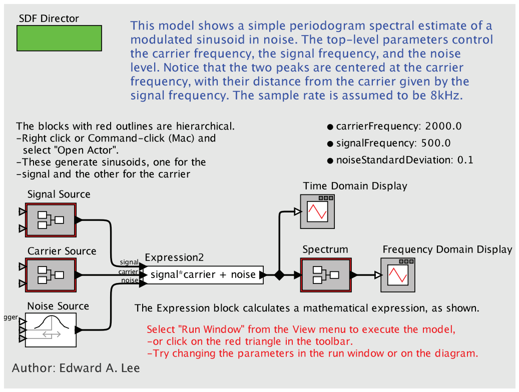
Keywords

Modeling, simulation, concurrency

1. INTRODUCTION

Ptolemy II [1] is an open-source software framework supporting experimentation with actor-oriented design. Actors are software components that execute concurrently and communicate through messages sent via interconnected ports. A model is a hierarchical interconnection of actors. In Ptolemy II, the semantics of a model is not determined by the framework, but rather by a software component in the model called a director, which implements a model of computation. The Ptolemy Project has developed directors supporting Kahn process networks (PN) [3], discrete-events (DE), dataflow (SDF), synchronous/reactive(SR), rendezvous-based models, 3-D visualization, and continuous-time models. Each level of the hierarchy in a model can have its own director, and distinct directors can be composed hierarchically. A major emphasis of the project has been on understanding the heterogeneous combinations of models of computation realized by these directors. Directors can be combined hierarchically with state machines to make modal models [4, 5]. A hierarchical combination of continuous-time models with state machines yields hybrid systems [6]; a combination of synchronous/reactive with state machines yields Statecharts [2] (the Ptolemy II variant [7] is close to SyncCharts).

Ptolemy II has been under development since 1996; it is a successor to Ptolemy Classic, which was developed since 1990. The core of Ptolemy II is a collection of Java classes and packages, layered to provide increasingly specific capabilities. The kernel supports an abstract syntax, a hierarchical structure of entities with ports and interconnections. A graphical editor called Vergil supports visual editing of this abstract syntax. An XML concrete syntax called MoML provides a persistent file format for the models. Various specialized tools have been created from this framework, including HyVisual (for hybrid systems modeling), Kepler (for scientific workflows), VisualSense (for modeling and simulation of wireless networks), Viptos (for sensor network design), and some commercial products. Key parts of the infrastructure include an actor abstract semantics, which enables the interoperability of distinct models of computation with a well-defined semantics; a model of time (specifically, super-dense time, which enables interaction of continuous dynamics and imperative logic); and a sophisticated type system supporting type checking, type inference, and polymorphism. The type system has recently been extended


This model shows a simple periodogram spectral estimate of a modulated sinusoid in noise. The top-level parameters control the carrier frequency, the signal frequency, and the noise level. Notice that the two peaks are centered at the carrier frequency, with their distance from the carrier given by the signal frequency. The sample rate is assumed to be 8kHz.

The blocks with red outlines are hierarchical.
-Right click or Command-click (Mac) and select "Open Actor".
-These generate sinusoids, one for the
-signal and the other for the carrier

- carrierFrequency: 2000.0
- signalFrequency: 500.0
- noiseStandardDeviation: 0.1

The Expression block calculates a mathematical expression, as shown.

Select "Run Window" from the View menu to execute the model,
-or click on the red triangle in the toolbar.
-Try changing the parameters in the run window or on the diagram.

Author: Edward A. Lee

Figure 1: An graph editor from the Vergil application. Vergil is a graphical editor for Ptolemy II.

to support user-defined ontologies [8]. Various experiments with synthesis of implementation code and abstractions for verification are included in the project.

2. SPEAKERS

- **Edward A. Lee**, (eal@eecs.berkeley.edu)
  `http://ptolemy.org/~eal`

  Edward A. Lee is the Robert S. Pepper Distinguished Professor of the Electrical Engineering and Computer Sciences (EECS) department at U.C. Berkeley. His research interests center on design, modeling, and simulation of embedded, real-time computational systems. He is the director of the Ptolemy project.

- **Stavros Tripakis** (stavros@eecs.berkeley.edu)
  `http://www-verimag.imag.fr/~tripakis`

  Stavros Tripakis is Associate Research Scientist at UC Berkeley. His work lies in the areas of embedded, real-time and distributed systems, model-based and component-based design, verification, testing and synthesis.

- **Christopher Brooks** (cxh@eecs.berkeley.edu)
  `http://ptolemy.org/~cxh`

  Christopher Brooks is the Ptolemy Project Software Manager and the Executive Director of the the Center for Hybrid and Embedded Software Systems (CHESS).

3. TOPICS

The tutorial is a hands-on tutorial of 3 to 3.5 hours on Sunday, October 24th, 2010. Attendees should bring their own laptops.

- (13:00-14:00) Setting up Ptolemy II 8.0 and Eclipse: (We will have USB sticks with Windows, Mac and Linux binaries): installing Eclipse; configuring Ptolemy; compiling Ptolemy; running a model within Eclipse.
- (14:00-15:00) Overview of Models of Computation in Ptolemy II: abstract semantics; actors and directors; currently implemented models of computation (Synchronous Data Flow, Dynamic Data Flow, Kahn Process Networks, Discrete Event, Continuous Time, Synchronous/Reactive and Modal Models); combining models of computation.
- (15:00 - 15:30) Break
- (15:30 - 17:00) Extending Ptolemy Models of Computation: Ptolemy execution semantics object model; using Eclipse to extend Ptolemy; building a non-dogmatic Kahn Process Networks model of computation; building a left to right model of computation.

The tutorial is similar to material presented at the February, 2009 Ptolemy Miniconference tutorial, (35 attendees, http://ptolemy.org/conferences/09/tutorial.htm) and at the February, 2007 Ptolemy Miniconference tutorial, (30 attendees, http://ptolemy.org/conferences/07).

4. REFERENCES

[1] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(2):127–144, 2003.

[2] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[3] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing*, pages 993–998. North-Holland Publishing Co., 1977.

[4] E. A. Lee. Finite State Machines and Modal Models in Ptolemy II. Technical Report UCB/EECS-2009-151, EECS Department, University of California, Berkeley, Nov 2009.

[5] E. A. Lee and S. Tripakis. Modal Models in Ptolemy. In *EOOLT 2010 – 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Linköping University Electronic Press, Oct. 2010. To appear.

[6] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume LNCS 3414, pages pp. 25–53, Zurich, Switzerland, 2005. Springer-Verlag.

[7] E. A. Lee and H. Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, Salzburg, Austria, 2007. ACM.

[8] J. M.-K. Leung, T. Mandl, E. A. Lee, E. Latronico, C. Shelton, S. Tripakis, and B. Lickly. Scalable semantic annotation using lattice-based ontologies. In *12th International Conference on Model Driven Engineering Languages and Systems*, pages 393–407. ACM/IEEE, October 2009.

Getting Started with Ptolemy II and Eclipse

Christopher Brooks
EECS Department
University of California, Berkeley
Berkeley, CA, USA
cxh@eecs.berkeley.edu

1. LOADING THE PTOLEMY, ECLIPSE AND (POSSIBLY) JDK DISTRIBUTIONS

1. There are two types of USB drives for loan. Most have Mac and Windows binaries on them. Drives that have an `L` written on them contain Linux binaries.

2. Insert the USB drive into your machine.

3. The Mac/Windows drives contain these directories

   ```
   doc mac32 mac64 macSrc win32 win64 winSrc
   ```

   Depending on whether you are running Mac OS X or Windows, *copy* the contents of `winSrc` *or* `macSrc`. These files contain the Ptolemy source files, we will use them if the installer fails.

   The Linux drives contain these directories:

   ```
   linux32 linux64 ptII8.0.1.src.tar.gz
   ```

   Linux users should *copy* the `ptII8.0.1.src.tar.gz` to the local directory and later untar it with:

   ```
   tar -zxf ptII8.0.1.src.tar.gz
   ```

4. See the "Setting up Ptolemy and Eclipse" document for whether to download the 32-bit or 64-bit versions of the Java Development Kit (JDK) and Eclipse.

   Depending on whether you are running Mac OS X or Windows, *copy* the contents of the `mac32`, `mac64`, `win32` or `win64`. When in doubt, try the 32-bit version.

5. Linux and Windows users, install the 1.6 Java Development Environment. Mac OS X users, there is no standalone Java installer, so you need take no action.

6. Linux, Mac OS X and Windows users, install the Eclipse Integrated Development Environment.

Setting up Ptolemy II and Eclipse

These are simplified instructions for setting up Eclipse using the source tree. See `eclipse.htm` for instructions about how to set up Eclipse using the Subversion version control system. Subversion allows you to get updates from the Ptolemy development tree. The instructions here provide access to a static source code tree, such as what is shipped with the Ptolemy II release.

These instructions assume you are using Eclipse Helios under Windows or Mac OS X. Other ways of setting up and building Ptolemy II are described on the Ptolemy II install page.

Contents of this page:

Install Eclipse

1. Download the latest version of Eclipse from http://www.eclipse.org.
   In October, 2010, we chose **Eclipse for RCP and RAP Developers**, which is 190Mb. The **Eclipse for RCP and RAP Developers** version includes the plug-in development environment (PDE), which is needed by the backtrack facility, which is an optional part of Ptolemy II that allows models to restore their old state. If the version of Eclipse that you install does not have the PDE, then there will be build errors, which can be fixed by excluding `ptolemy/backtrack/` from the build.

   Eclipse is available as both 32-bit and 64-bit binaries. Deciding which one to download can be complex.

   The basic idea is that to use a 64-bit version of Eclipse, you must have a 64-bit JVM and an operating system that will support 64-bit applications. When in doubt, choose the 32-bit version of Eclipse, it is more likely to work than the 64-bit version.

   See below for details.

   32-bit or 64-bit?

   - (*Linux*): Run `uname -a`. If the result contains `x64`, you are running a 64-bit version of Linux and would select the 64-bit version of Eclipse.. Also, try `java -version`. Below is a run on a 64-bit Linux machine:

     ```
     -bash-3.2$ uname -a
     Linux sisyphus.eecs.berkeley.edu 2.6.18-164.11.1.el5xen #1 SMP \
      Wed Jan  6 13:43:33 EST 2010 x86_64 x86_64 x86_64 GNU/Linux
     -bash-3.2$ java -version
     java version "1.5.0_22"
     Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_22-b03)
     Java HotSpot(TM) 64-Bit Server VM (build 1.5.0_22-b03, mixed mode)
     ```

- (*Mac OS X*): Under Mac OS X 10.5 and 10.6, the default JVM is 1.6, which is 64-bit. However, Java 1.6.0_20 does not work well. The issue is that the following stack trace appears:

  ```
  2010-10-10 11:50:02.263 java[7823:613] java.lang.NullPointerExceptior
  at apple.awt.CToolkit.postEvent(CToolkit.java:1086)
  at apple.awt.EventFactoryProxy.forwardEvent(EventFactoryProxy.java:53
  ```

  The bug happens when running from Eclipse or from the command line. The bug requires UI interactions such as mouse movement or clicking. The bug is probably a threading bug and may only show up on multi-core machines. The downside is that when the bug occurs, the debugging session in Eclipse may become non-responsive.

  For further information, see: `http://lists.apple.com/archives/java-dev/2010/May/msg00140.html`.
  So, at this time, we are recommending the 32-bit version with Java 1.5.
  Download `eclipse-rcp-helios-SR1-macosx-cocoa.tar.gz` uncompress it and move `eclipse/` into `/Applications`, which will create `/Applications/eclipse/Eclipse.app`.
  Firefox Mac OS X Details:

  1. Firefox will download the file into a directory, which may be `/Downloads`
  2. Double click on `eclipse-rcp-helios-SR1-macosx-cocoa.tar.gz` which will create the `eclipse` directory.
  3. Drag the `eclipse` directory to your `Applications` directory.

  The next step on the Mac is to force Eclipse to use Java 1.5 by editing `eclipse.ini`.
  1. In the Finder, go to `/Applications/eclipse`, find the `Eclipse` executable and right click (Control + click) and choose "Show Package Contents".
  2. Browse to `Contents/MacOS` and double click on `eclipse.ini`, which will open up the file in TextEdit
  3. Edit the file and insert `-vm /System/Library/Frameworks/JavaVM.framework/Versions/1.5/Home/bin/java`
     before the `-vmargs` line.
  4. Save the file and exit TextEdit.

  For details about `eclipse.ini`, see http://wiki.eclipse.org/Eclipse.ini
  When you start up Eclipse, if you want to verify which version of Java you are running
  - (*Mac OS X*): `Eclipse` | `About Eclipse` | `Installation Details` | `Configuration`.
    For Java 1.5, the `java.class.version` property should be `49.0`.
    For a 64-bit JVM, the `java.vm.name` property *might* have the string `64` in it.
- (*Windows*): Determine whether you are running a 32-bit or 64-bit version of the OS.
  - *Windows 7 or Vista*: "Start", then right click on "Computer", then click on "Properties". Under System, you should see the system type. The 64-bit version of Windows 7 will say "64-bit Operating System". See `http://windows.microsoft.com/en-US/windows7/32-bit-and-64-bit-Windows-frequently-asked-questions`. for details.
  - *Windows XP*: "Start", then right click on "My Computer", and then click on "Properties". Under "System", if you see "x64 Edition", then you are running a 64-bit version of Windows. If "x64 Edition" is **not** listed, then you are running a 32-bit version of Windows.

After determining whether to download the 32-bit or 64-bit version of Eclipse, download the file and unzip the download file into an appropriate place, such as `C:\Program Files`, which will create `C:\Program Files\eclipse\eclipse.exe`.
If `eclipse\eclipse.exe` is not created, then it could be that the security policy on your machine is preventing the creation of `.exe` files. If this is the case, then try right clicking on the zip file, select "Properties" and then at the bottom, hit "Unblock" and then "Apply". Or try running `unzip` from the command line.

2. (*All Platforms*): Finish the installation by double clicking on `/Applications/eclipse/Eclipse.app` or `eclipse/eclipse.exe`.
   The first time this is run it will complete the installation process.
   - (*Windows*): If Eclipse fails to start with the message "Windows cannot access the specified device, path or file. You may not have the appropriate permissions to access the item", then it may be necessary to make `eclipse.exe` and a dll executable:

     ```
     chmod a+x eclipse.exe
     ```

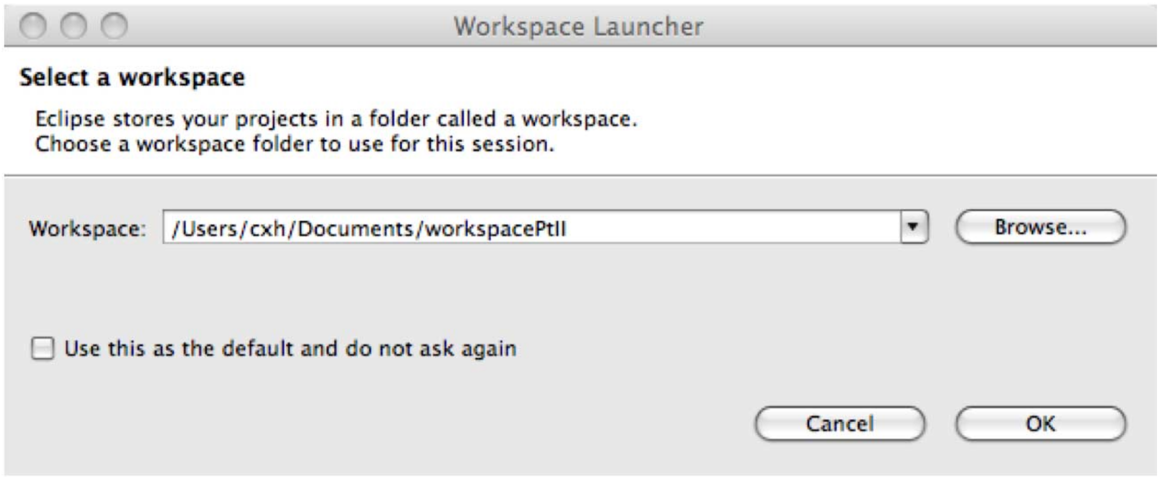
     If you get the message "The Eclipse executable launcher was unable to locate its companion shared library", then run the following command in the `eclipse` directory.

     ```
     find . -name *.dll -exec chmod a+x {} \;
     ```

     After the first run, normal start-up will occur whenever it is started.

Eclipse Preferences for Ptolemy II

Eclipse has the notion of a workspace, which is a collection of one or more projects. Ptolemy II is a project.
When the "Workspace Launcher" window comes up, either stick with the default or choose a new directory. Note that the projects can be located outside of the workspace directory. One caveat is that Eclipse will not permit the project directory to be a parent of the workspace directory, so placing the workspace in a subdirectory such as the default location is best.
Click "OK" to create the workspace and open Eclipse.
Below is the Workspace Launcher Window:


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /Users/cxh/Documents/workspacePtII

☐ Use this as the default and do not ask again

The default configuration of Eclipse has some difficulties with Ptolemy II, so a few changes are necessary.

1. (*Optional*): Shortcuts and Memory Size
    - (*Mac OS X*): In `Applications/eclipse`, drag the `Eclipse.app` icon to the Finder Dock on the edge of your screen. This will give you fast access to starting Eclipse.
      There is usually no need to set the memory size for Eclipse under Mac OS X. By default, it is set to 512 megabytes. For details about setting the memory size, see: Eclipse Workbench User Guide/Tasks/Running Eclipse.
    - (*Windows*): In Windows, create a shortcut to `eclipse.exe` by going to the directory where Eclipse is installed, right clicking on eclipse.exe and selecting `Create Shortcut`.
      To add Eclipse to the start menu, right click on the shortcut and select `Pin to Start Menu`
      There is usually no need to set the memory size for Eclipse. By default, it is set to 512 megabytes in the `eclipse.ini` file.
2. Eclipse requires some customization to build Ptolemy II and to keep the Ptolemy II coding style.
   In the steps below, we outline changes to be made in the Eclipse Preferences window.
   Under *Windows* the Eclipse Preferences window is invoked via `Window` | `Preferences`.
   Under *Mac OS X* the Eclipse Preferences window is invoked via `Eclipse` | `Preferences`.
   For each of the changes, hit `Apply`. When all the changes are done, hit `OK`, which will close the Eclipse Preferences Window. in the preferences window
   (*Mac OS X*: `Eclipse` | `Preferences`)
   (*Windows*: `Window` | `Preferences`)
3. Go to `Java` | `Code Style` | `Formatter` and click New
4. In the "New Profile" Window, for the Profile name, enter "Ptolemy II" Under "Initialize settings with the following profile", select "Java Conventions [built-in]" (We use Java Conventions over the "Eclipse" setting because the Eclipse style uses tabs.) Then hit OK
5. Under the "Indentation" tab, change the Tab policy to "Spaces only".
6. Under the "New Lines" tab, select "at end of file"
7. Under the "Comments" tab,
   unselect "Enable Javadoc comment formatting"
   unselect "Enable block comment formatting", otherwise the block comments that have <pre> ... <pre> get changed.
   unselect "Enable line comment formatting", otherwise the headers for public methods etc. get changed.
8. Click OK.
9. In the Preferences Window, click Apply.

- Ptolemy II 8.0 uses some features of Java 1.5, which is also known as Java 5.0. In particular, `ptolemy/actor/ptalon` uses generics, which require Java 1.5 (aka Java 5.0) or later.

However, the Ptolemy II development trunk that includes changes after Ptolemy II 8.0 uses some features of Java 1.6 (aka Java 6.0). In particular, `ptdb` to uses javax.swing.GroupLayout. which requires Java 1.6 (also known as 6.0) or later.

- (*Mac OS X*): It is best to set Eclipse to use Java 1.5 or 5.0 or later source code compliance.
- (*Windows*): It is best to set Eclipse to use Java 1.6 or 6.0 or later source code compliance.

1. While still in the preferences window
   (*Mac OS X*: `Eclipse` | `Preferences`)
   (*Windows*: `Window` | `Preferences`)
   expand `Java` | `Compiler`
2. Make sure that Set "Compiler compliance level":
    - (*Mac OS X*): "1.5" or "5.0"
    - (*Windows*): "1.6" or "6.0"

   Note that Java 1.5 is the same as Java 5.0 and Java 1.6 is the same as Java 6.0. What we don't want is "1.4" or "4.0". If you plan on using the the Ptolemy II development tree after Ptolemy II 8.0. then you don't want "1.5" or "5.0".
3. Click Apply.

- Eclipse has very good compiler error/warning. One of the warnings complains if a Serializable class does not have serialVersionUID declared. Since this warning is only useful if you are tightly managing serialization, we turn it off:

1. While still in the preferences window
   (*Mac OS X*: `Eclipse` | `Preferences`)
   (*Windows*: `Window` | `Preferences`)
   Expand `Java` | `Compiler` | `Errors/Warning`
2. Under "Potential programming problems", change "Serializable class without serialVersionUID" to "Ignore"
3. Under "Generic Types", change "Unchecked generic type operation" to "Ignore".
4. Under "Generic Types", change "Usage of a raw type" to "Ignore".
5. Click Apply. If you are prompted for a full rebuild, click Yes.

- (*Windows*:) The PtDoc Doclet in `$PTII/doc/doclets/PtDoclet.java` requires `tools.jar`, which is only in the Java Development Kit (JDK), not the Java Runtime Environment (JRE). So, be sure that a JDK is selected for building, not a JRE.
While still in the preferences window:
(*Mac OS X*: `Eclipse` | `Preferences`)
(*Windows*: `Window` | `Preferences`)

1. Select `Java` | `Installed JREs`
   (*Mac OS X*): select `JVM 1.5.0`.
   (*Windows*): select `JVM 1.6.0`.
2. Verify that the checked line corresponds with a JDK, not a JRE.

- (*Windows*): While in the "Installed JREs" window, under Java 1.6, `tools.jar` must be added the external jars:

1. Select the default JRE | `Edit` | `Add External Jars`.
2. Browse to the path of your JDK for example `c:\Program Files\Java JDK1.6.0_22\lib`
3. Hit `Open`, select `tools.jar` and then finish.

- Click OK in the Preferences window to apply all of the above changes.

Setting up Eclipse to manage your Ptolemy II development environment

Eclipse will manage your ptII code tree as a *project* called the ptII project.

1. Download a ptII source tree from http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIlatest
   The Windows and Mac OS X installers include the source, or you may install a separate source-only download.
2. Eclipse uses the `.classpath` file to set paths and exclude files. There are two choices:
   1. In the Ptolemy II source tree, copy `.classpath.default` to `.classpath`. This choice is the simpler choice, but might exclude optional third party packages.
   2. Run `./configure`. This choice is more complex, especially for Windows users. However, this choice will include optional third party software that is already installed on your

machine.

Choose **one** of the options below

A. Copy .classpath to .classpath.default

- (*Mac OS X*): Unfortunately, by default the Mac OS X Finder hides files whose names begin with a period, such as `.classpath`. One solution is to follow the instructions from: http://www.macworld.com/article/51830/2006/07/showallfinder.html:
  1. Start up the Terminal application, which may be found as `/Utilities/Terminal.app`. Enter:

     ```
     defaults write com.apple.Finder AppleShowAllFiles YES
     ```

  2. Restart the Finder by holding down the Option key, then clicking and holding on the Finder icon in the Dock. Then, select "Relaunch". Or, log out and log back in again.
  3. Start up the Terminal application, which may be found as `/Utilities/Terminal.app`
  4. In the Terminal, change to the directory where the Ptolemy II source code was installed. For example:

     ```
     cd /Applications/Ptolemy/ptII8.0.1
     ```

  5. Copy the files:

     ```
     cp .classpath.default .classpath
     ```

- (*Windows*):
  1. Find the Ptolemy II source distribution, which by default would be `c:\Ptolemy\ptII8.0.1`.
  2. Right click on `.classpath` and select delete.
  3. Right click on `.classpath.default`, select copy.
  4. Select paste and paste a copy of the file.
  5. Right click on the copy, select rename and rename the file to `.classpath.default`.

OR

B. Run ./configure

Ptolemy II includes a number of packages that rely on software that you may or may not have installed, such as MATLAB, the rxtx package (for serial port connections), joystick support, Java Advanced Imaging (JAI), the Java Media Framework (JMF), and Java 3D. If you wish to use or extend these features, you will need to perform a few extra steps. These steps require execution of a script called `configure`

The procedure below will modify the `.classpath` file that is provided in the version control repository to customize it for the software that you have installed.

- (*Mac OS X*): Using the Terminal to run configure
  1. Start up the Terminal application, which may be found as `/Utilities/Terminal.app`
  2. In the Terminal, change to the directory where the Ptolemy II source code was

installed. For example:

```
cd /Applications/Ptolemy/ptII8.0.1
```

3. Set `$PTII` to the location of the source tree:

```
export PTII=`pwd`
```

4. Make `configure` executable.

```
chmod a+x configure
```

```
5.  Run ./configure
```

```
./configure
```

- (*Windows*): Install Cygwin and run configure Unfortunately Windows does not ship with Unix shell commands like `make`. The workaround is to install Cygwin, see http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIlatest/cygwin.htm
  1. Start up the Cygwin Bash shell: `Start` | `Run` | `Cygwin`
  2. Change to the directory where the Ptolemy II source code was installed. For example:

     ```
     cd c:/Ptolemy/ptII8.0.1
     ```

     Note that if there are **spaces** in the path name to the directory where your Ptolemy installation is located, then you should use double quotes around the path: `cd "c:/Documents and Settings/`*`yourLogin`*`/workspace/ptII"`
  3. Set `$PTII` to the location of the Ptolemy II source tree:

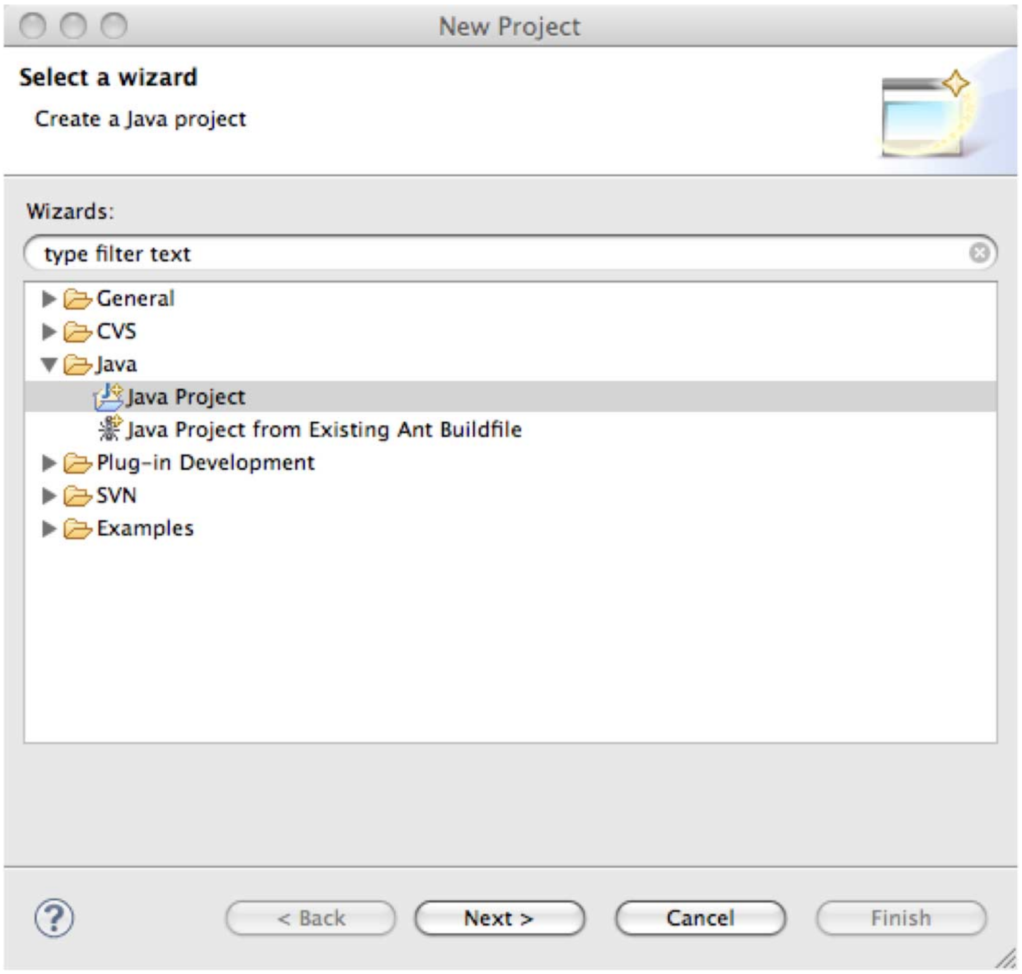
     ```
     export PTII=c:/Ptolemy/ptII8.0.1
     ```

     Again, if there are spaces in the path, use double quotes: `export PTII="c:/Documents and Settings/`*`yourLogin`*`/workspace/ptII"`
  4. Run:

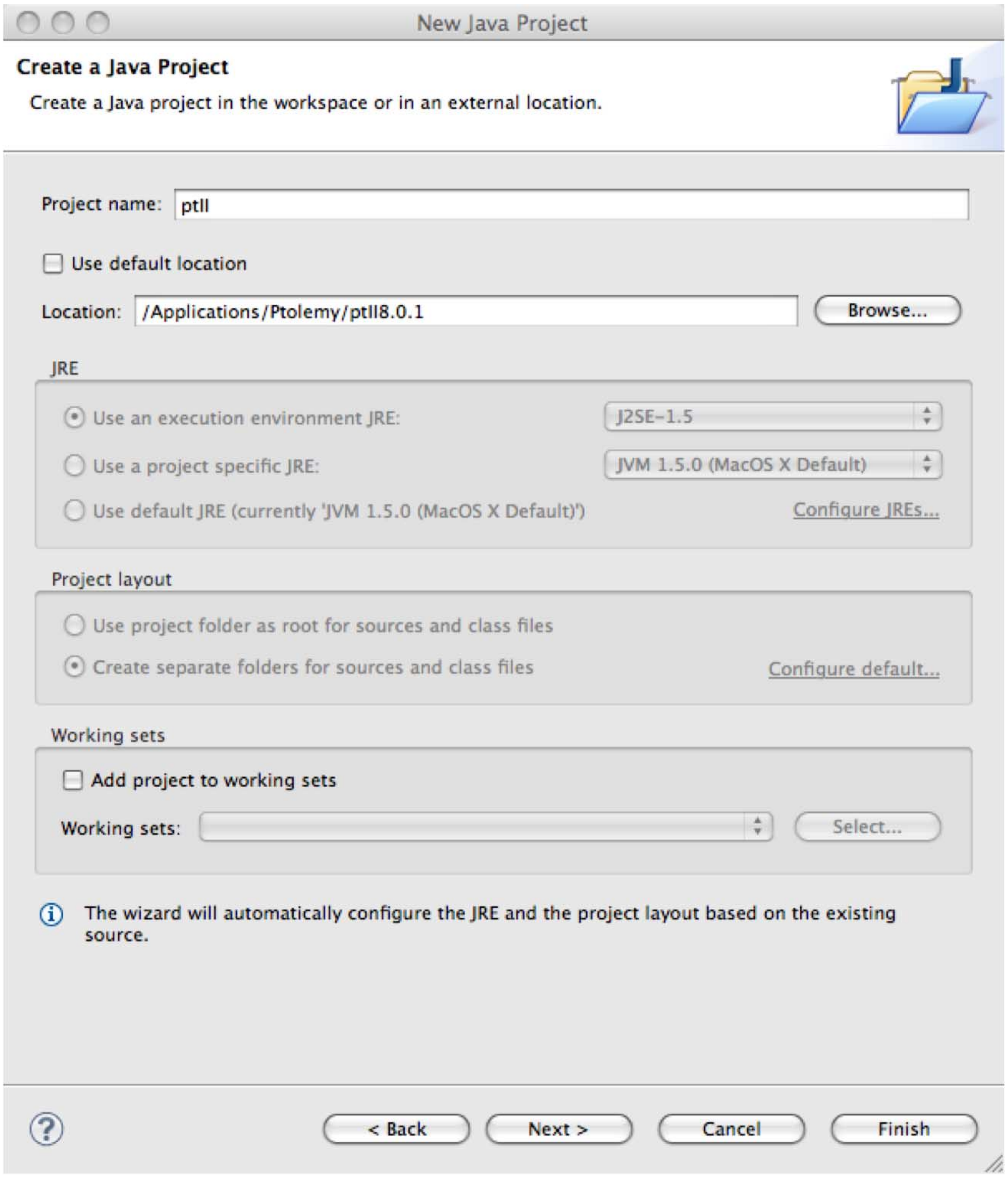
     ```
     ./configure
     ```

     This will create `$PTII/.classpath`

3. If Eclipse is not running, start up Eclipse
4. In Eclipse, do `File` | `New` | `Project`.
5. In the "New Project" window, select "Java" > "Java project", Click Next.



6. In the "New Java Project" window:
   In "Project Name", enter any project name, `ptII` is a common choice.
   Uncheck "Use default location" and browse to the location of the source tree.
   (*Mac OS X*): For example, browse to `/Applications/Ptolemy/ptII8.0.1`
   Press "Next". There might be a delay as Eclipse scans your ptII tree.
   Below is the New Java Project window:


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: ptII

☐ Use default location

Location: /Applications/Ptolemy/ptII8.0.1

JRE

- [✓ Use an execution environment JRE:] J2SE-1.5
- Use a project specific JRE: JVM 1.5.0 (MacOS X Default)
- Use default JRE (currently 'JVM 1.5.0 (MacOS X Default)')

Project layout

- Use project folder as root for sources and class files
- [✓ Create separate folders for sources and class files]

Working sets

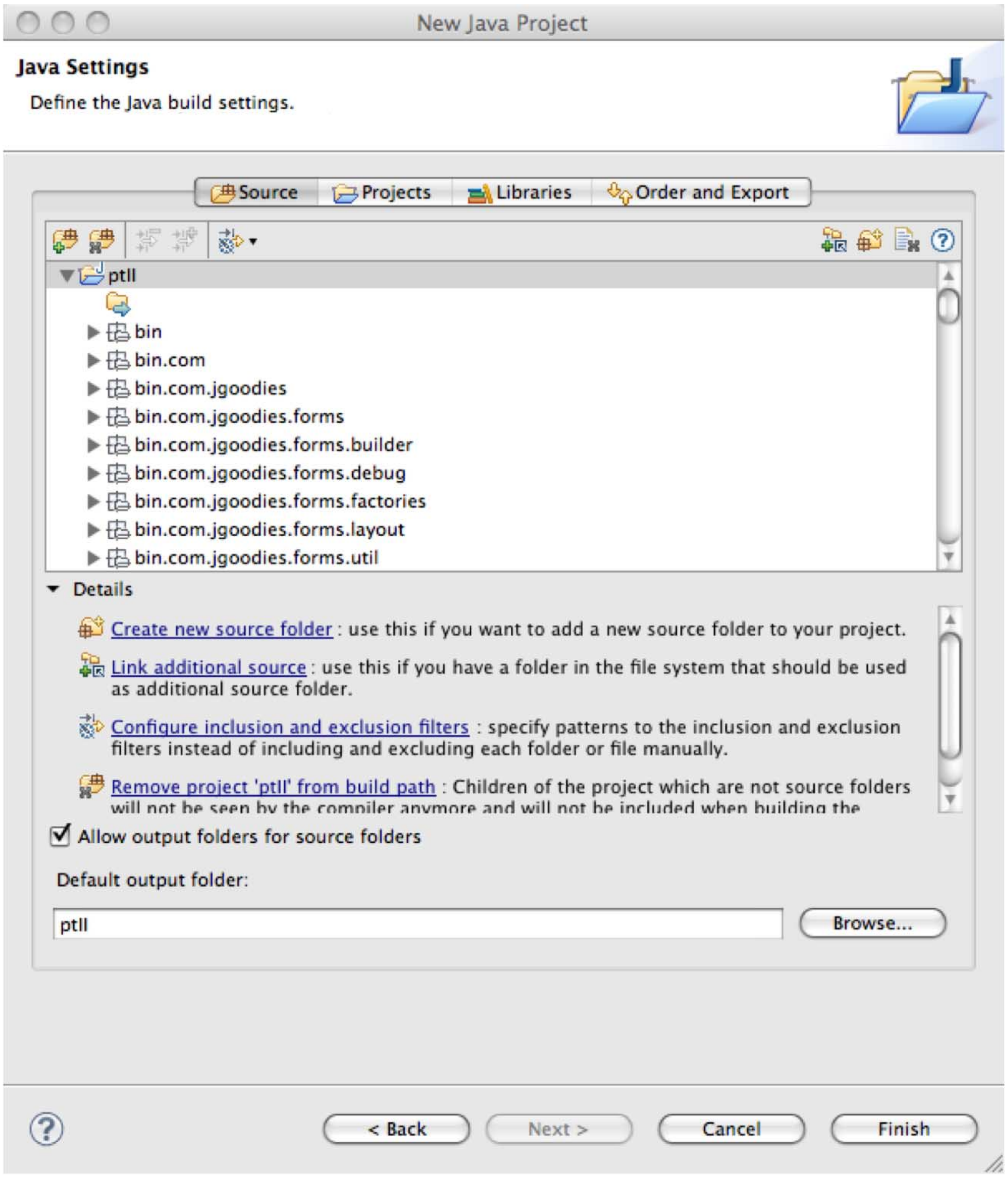
☐ Add project to working sets

Working sets:

The wizard will automatically configure the JRE and the project layout based on the existing source.

Warning: If the "Setting Build Paths" window comes up with a message "The output folder has changed. Do you want to remove all generated resources from the old location ptII/bin", then the problem is that the `.classpath` file was not found. It is best if you exit Eclipse, create the `.classpath` file and redo the above steps.

7. In the "New Java Project" window, select "Allow output folders for source folder", then click "Finish"
   Below is the second New Java Project window:


New Java Project

Java Settings

Define the Java build settings.

ptII
- bin
- bin.com
- bin.com.jgoodies
- bin.com.jgoodies.forms
- bin.com.jgoodies.forms.builder
- bin.com.jgoodies.forms.debug
- bin.com.jgoodies.forms.factories
- bin.com.jgoodies.forms.layout
- bin.com.jgoodies.forms.util

Details

Create new source folder : use this if you want to add a new source folder to your project.

Link additional source : use this if you have a folder in the file system that should be used as additional source folder.

Configure inclusion and exclusion filters : specify patterns to the inclusion and exclusion filters instead of including and excluding each folder or file manually.

Remove project 'ptII' from build path : Children of the project which are not source folders will not be seen by the compiler anymore and will not be included when building the

☑ Allow output folders for source folders

Default output folder: ptII

8. When asked if you want to shift to the Java perspective, click on Yes.
9. The workspace will take a few minutes to build.

Running Ptolemy II

1. In the Run menu, select "Run Configurations...".

2. In the resulting dialog, select "Java Application" and click "New".

3. In the dialog, fill in the boxes as follows:
   - Name: `Vergil`
   - Project: `ptII`
   - Main class: `ptolemy.vergil.VergilApplication`

4. Press the Run button.

The Ptolemy II welcome window should appear.

You may now wish to read the Using Vergil tutorial.

Simple Debugging Session

1. Locate ptolemy/vergil/VergilApplication in the Explorer and double click. Place a breakpoint on the first line of main() by using `Run` | `Add/Remove Breakpoint`
2. Tell Eclipse which class to run with `Run` | `Run`. On the Main tab, select the Ptolemy II package and enter ptolemy.vergil.VergilApplication as Main class.
3. Press the Run button
4. To debug, quit Vergil, and place a breakpoint in, say, the fire() method of ptolemy.domains.ct.kernel.CTBaseIntegrator. Then `Run` | `Debug`, and as above. Open the Lorenz CT demo from the Quick Tour and run it.

Troubleshooting

Preferences

If you have already used Eclipse and you would like to start over with new projects and preferences, remove the `workspace` directory in the Eclipse directory. The `workspace` directory will only appear if you have already run Eclipse. **Note that removing the `workspace` directory will cause Eclipse to 'forget' about any projects that you may have set up**

Build Error: Assert cannot be resolved

If, when building, the Problem tab shows "Assert cannot be resolved", then the problem is that Junit is not being found.

Rebuilding Briefly flashes a window

If you have problems where clicking on build briefly flashes up a window, look in `$PTII/.classpath` for and empty exclusion that looks like ||

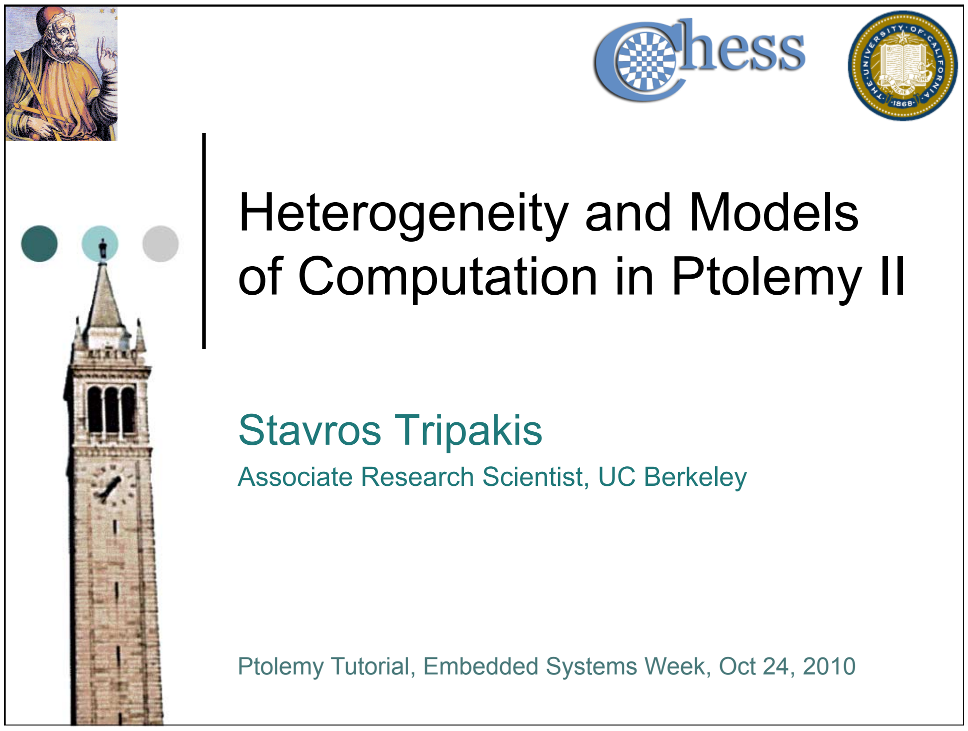
If you have problems with the classpath, look in the `workspace/.metadata/log` file that is in the directory where eclipse is installed. For more information about the `.metadata` directory, see below.

Eclipse takes a long time to start up

If Eclipse takes a long time to start up, then the problem could be a problem in your .metadata file.

Basically, when eclipse starts up, it might try to update `H:/workspace/.metadata` . The solution is covered in `http://www.eclipse.org/documentation/html/plugins/org.eclipse.platform.doc.user/doc/tasks/running_eclipse.htm`: The way I figured this out was by running Norton Antivirus and doing View -> File System Realtime Scan Statistics and then I noticed that my machine was updating H:/workspace/.metadata

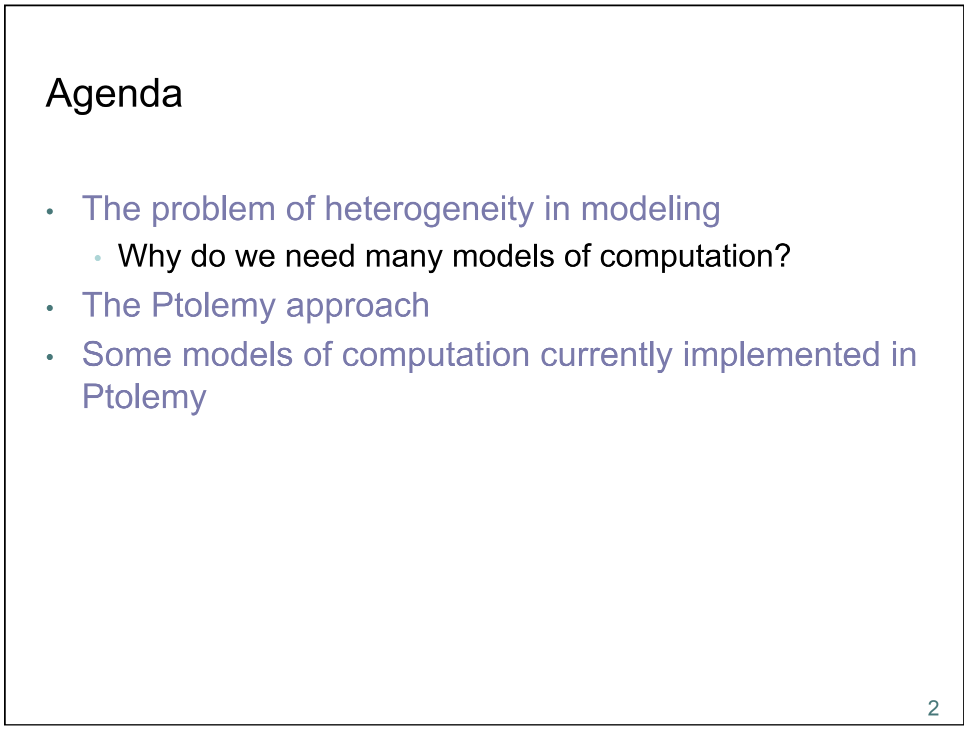
I think I introduced the problem by clicking on the Eclipse.exe binary and selecting Pin to Start Menu. My solution was to remove the Eclipse bogus entry in the start menu and then create a shortcut, change Start in property and then pin that shortcut to my start menu.



Heterogeneity and Models of Computation in Ptolemy II

Stavros Tripakis
Associate Research Scientist, UC Berkeley

Ptolemy Tutorial, Embedded Systems Week, Oct 24, 2010



Agenda

- The problem of heterogeneity in modeling
  - Why do we need many models of computation?
- The Ptolemy approach
- Some models of computation currently implemented in Ptolemy

Agenda

- The problem of heterogeneity in modeling
  - Why do we need many models of computation?
- The Ptolemy approach
- Some models of computation currently implemented in Ptolemy

Heterogeneity in modeling

One system, many models!

Different parts of the same system typically modeled using different modeling languages.

How to combine these different models?

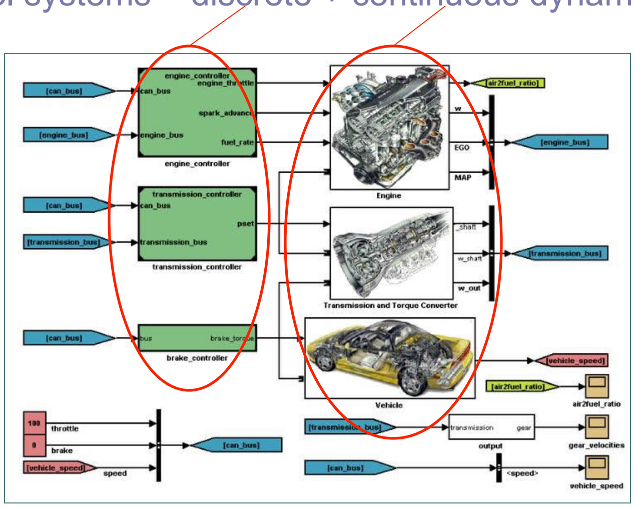
How to reason about the system as a whole?

Examples of heterogeneous models

Control systems = discrete + continuous dynamics

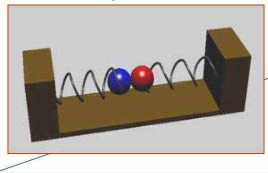
Simulink

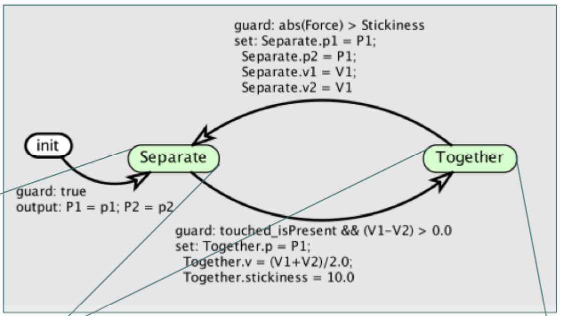
Copyright
The Mathworks

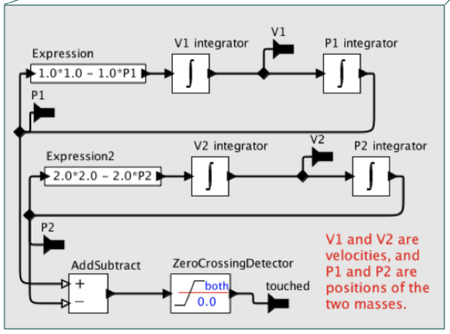


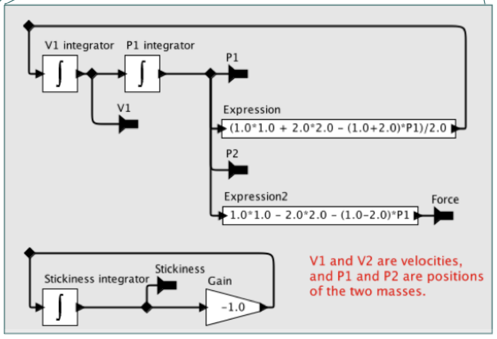
Hybrid automata = state machines + differential equations

sticky masses

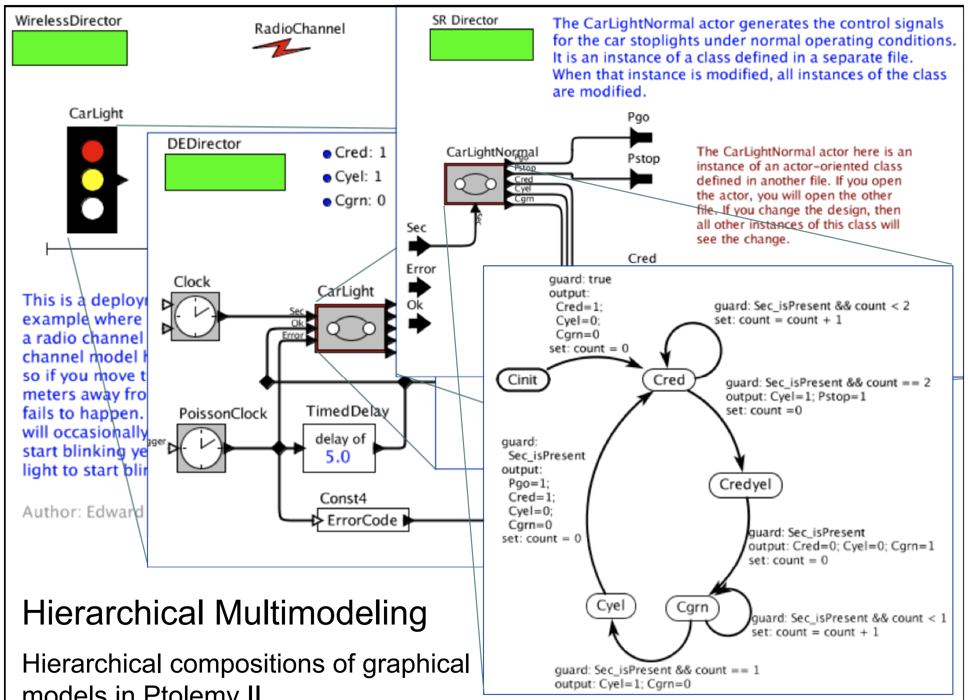








V1 integrator
P1 integrator
P1
V1
Expression
(1.0*1.0 + 2.0*2.0 - (1.0+2.0)*P1)/2.0
P2
Expression2
1.0*1.0 - 2.0*2.0 - (1.0-2.0)*P1
Force
Stickiness integrator
Stickiness
Gain
-1.0
V1 and V2 are velocities, and P1 and P2 are positions of the two masses.



The CarLightNormal actor generates the control signals for the car stoplights under normal operating conditions. It is an instance of a class defined in a separate file. When that instance is modified, all instances of the class are modified.

The CarLightNormal actor here is an instance of an actor-oriented class defined in another file. If you open the actor, you will open the other file. If you change the design, then all other instances of this class will see the change.

This is a deploy example where a radio channel channel model so if you move t meters away fro fails to happen. will occasionally start blinking ye light to start blir

guard: true
output:
Cred=1;
Cyel=0;
Cgrn=0
set: count = 0

guard: Sec_isPresent && count < 2
set: count = count + 1

guard: Sec_isPresent && count == 2
output: Cyel=1; Pstop=1
set: count =0

guard:
Sec_isPresent
output:
Pgo=1;
Cred=1;
Cyel=0;
Cgrn=0
set: count = 0

guard: Sec_isPresent
output: Cred=0; Cyel=0; Cgrn=1
set: count = 0

guard: Sec_isPresent && count < 1
set: count = count + 1

guard: Sec_isPresent && count == 1
output: Cyel=1; Cgrn=0

Hierarchical Multimodeling

Hierarchical compositions of graphical models in Ptolemy II.

Agenda

- The problem of heterogeneity in modeling
- The Ptolemy approach
  - Abstract semantics
  - Actors
  - Hierarchy
  - Directors
  - Models of computation (MOCs)
- Some MOCs currently implemented in Ptolemy

Abstract semantics

Views every actor as an Abstract State Machine:

Actor = Inputs + Outputs + States + Initial state + Fire + Postfire

Fire = output function: produces outputs given current inputs + state $F : S \times I \rightarrow O$

Postfire = transition function: updates state given current inputs + state $P : S \times I \rightarrow S$

Why separate fire and postfire?

9

Behaviors

Set of traces:

$$s_0 \xrightarrow{x_0, y_0} s_1 \xrightarrow{x_1, y_1} s_2 \xrightarrow{x_2, y_2} \cdots$$

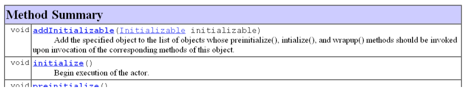
such that for all i :

$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i, x_i)$$

10

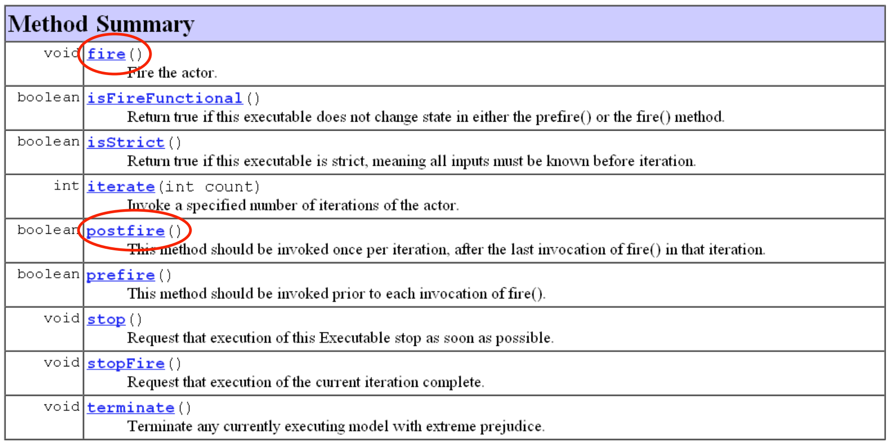
Implemented as Java interfaces

Interface “initializable”



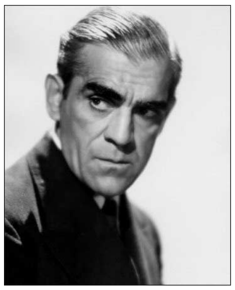
Method Summary	
void	addInitializable(Initializable initializable) Add the specified object to the list of objects whose preinitialize(), intialize(), and wrapup() methods should be invoked upon invocation of the corresponding methods of this object.
void	initialize() Begin execution of the actor.
void	preinitialize()

Interface “executable”



Method Summary	
void	fire() Fire the actor.
boolean	isFireFunctional() Return true if this executable does not change state in either the prefire() or the fire() method.
boolean	isStrict() Return true if this executable is strict, meaning all inputs must be known before iteration.
int	iterate(int count) Invoke a specified number of iterations of the actor.
boolean	postfire() This method should be invoked once per iteration, after the last invocation of fire() in that iteration.
boolean	prefire() This method should be invoked prior to each invocation of fire().
void	stop() Request that execution of this Executable stop as soon as possible.
void	stopFire() Request that execution of the current iteration complete.
void	terminate() Terminate any currently executing model with extreme prejudice.

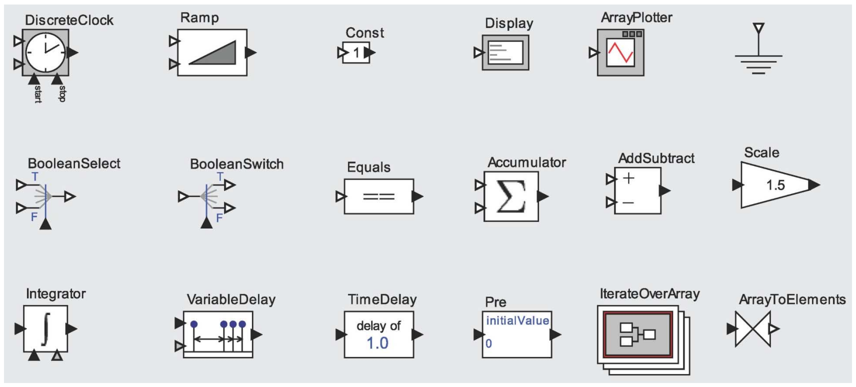
Actors



Actors

On-line documentation

- Ptolemy has a library of predefined actors
- They are all Java classes that implement the "executable" interface



13

Abstract vs. concrete semantics

Different actors instantiate these sets differently: concrete semantics differ

Fire = output function: produces outputs given current inputs + state

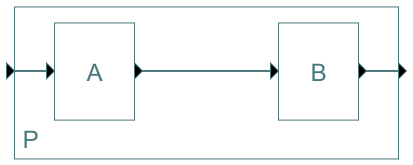
$$F : S \times I \rightarrow O$$

Postfire = transition function: updates state given current inputs + state

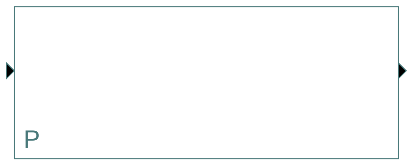
$$P : S \times I \rightarrow S$$

14

Hierarchy

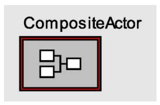


Hierarchy



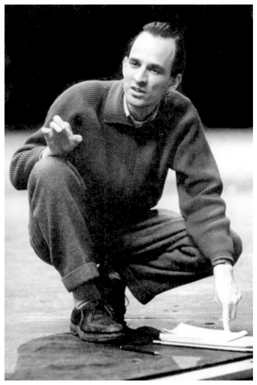
Fundamental modularization mechanism:
hide details, master complexity

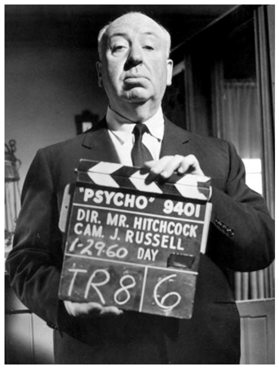
In Ptolemy: **composite** actors



How to give semantics to a hierarchical model?

Directors





Directors = composition operators

Given a diagram of actors *A1, A2, …,* with fire & postfire functions *F1/P1, F2/P2, …*

… a director defines a new pair of fire & postfire functions *F* and *P.*

F and *P* define a new, composite actor *A.*

A can be used like an atomic actor (black-box).

Caveat: Java code implements this simplified abstract view: it is more complex, but also more flexible, optimized, …

Models of computation

In general:

MOC = a paradigm for concurrent computation and communication between a set of actors

In Ptolemy:

Each director defines a MOC = “domain”

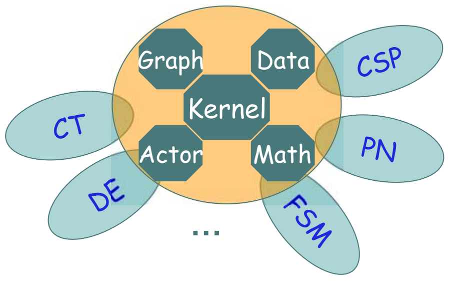
Agenda

- The problem of heterogeneity in modeling
- The Ptolemy approach
- Some models of computation currently implemented in Ptolemy
  - Synchronous Reactive (SR)
  - Synchronous Data Flow (SDF)
  - Modal Models
  - Discrete Event (DE)
  - Continuous Time (CT)
  - Process Networks (PN)
  - Rendez-vous

21

Models of Computation Implemented in Ptolemy II

1. CI – Push/pull component interaction
2. Click – Push/pull with method invocation
3. CSP – concurrent threads with rendezvous
4. Continuous – continuous-time modeling with fixed-point semantics
5. CT – continuous-time modeling
6. DDF – Dynamic dataflow
7. DE – discrete-event systems
8. DDE – distributed discrete events
9. DPN – distributed process networks
10. FSM – finite state machines
11. DT – discrete time (cycle driven)
12. Giotto – synchronous periodic
13. GR – 3-D graphics
14. Pthales – multidimensional data flow
15. PN – process networks
16. Rendezvous – extension of CSP
17. SDF – synchronous dataflow
18. SR – synchronous/reactive
19. TM – timed multitasking

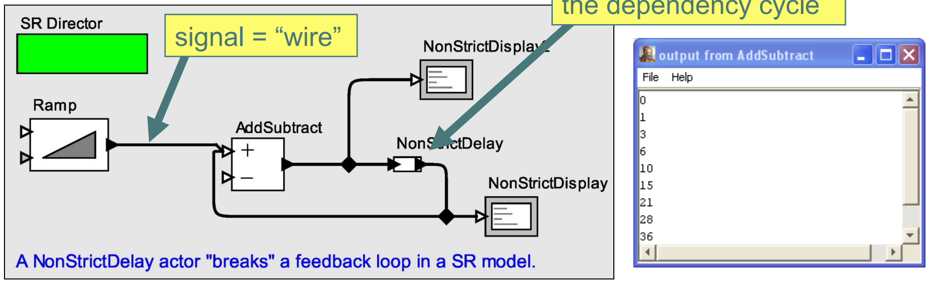


22

[Synchronous languages: Esterel, Lustre, Signal, …]

Synchronous/Reactive (SR)

Moore machine breaks the dependency cycle



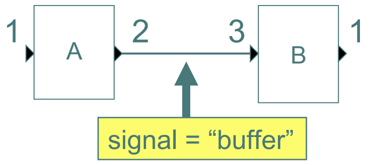
A NonStrictDelay actor "breaks" a feedback loop in a SR model.

- Operation proceeds in global "clock ticks" (synchronous rounds).
- At each tick, every signal has a value or is absent: $I = V \cup \{\varepsilon\}$
- SR Director fires actors in topological order
  - Well-defined when delays ("latches") break cyclic dependencies
- In case of causality cycles, fixpoint computation starting from unknown values for all signals ("constructive semantics"):
  - Iterate **fire** of all actors until fixpoint is reached; then **postfire** each actor just once.

c.f. separation of fire and postfire

[Lee-Messerschmitt '87]

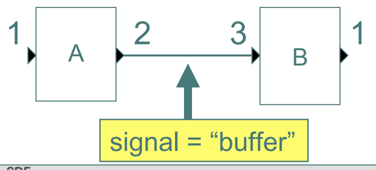
Synchronous Data Flow (SDF)



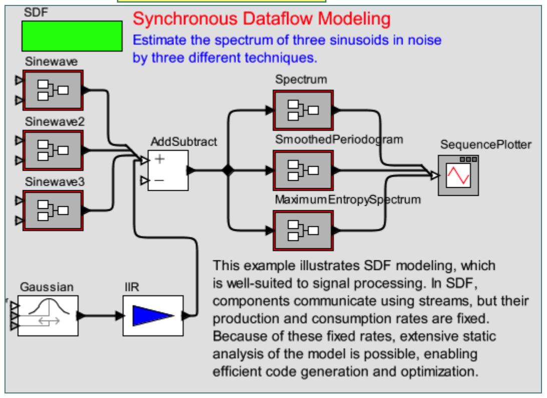
In each firing, actors consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.

[Lee-Messerschmitt '87]

Synchronous Data Flow (SDF)

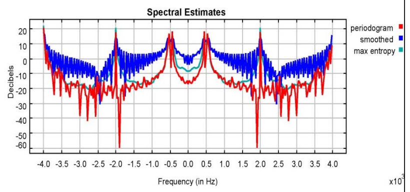


In each firing, actors consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.



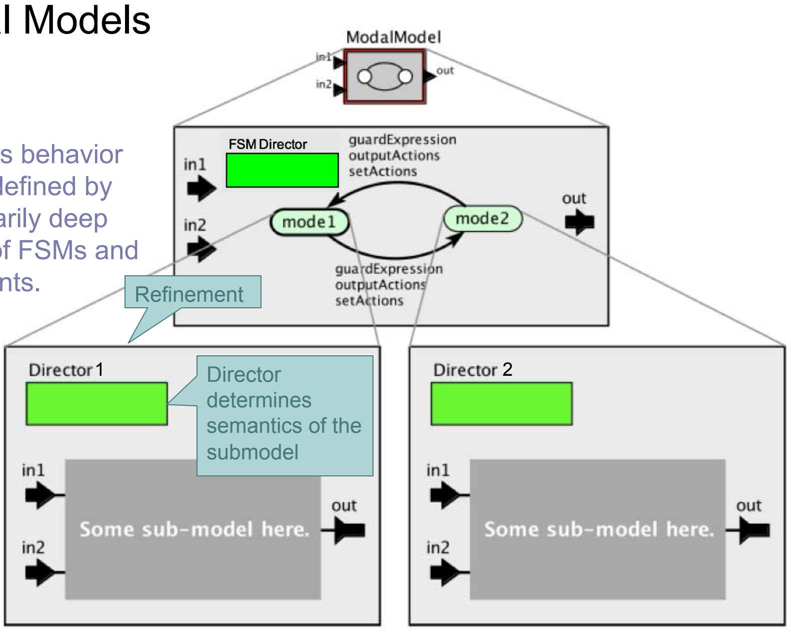
This example illustrates SDF modeling, which is well-suited to signal processing. In SDF, components communicate using streams, but their production and consumption rates are fixed. Because of these fixed rates, extensive static analysis of the model is possible, enabling efficient code generation and optimization.

- SDF Director computes periodic schedule, e.g., A,A,A,B,B
- Composite fire() fires all internal actors according to schedule



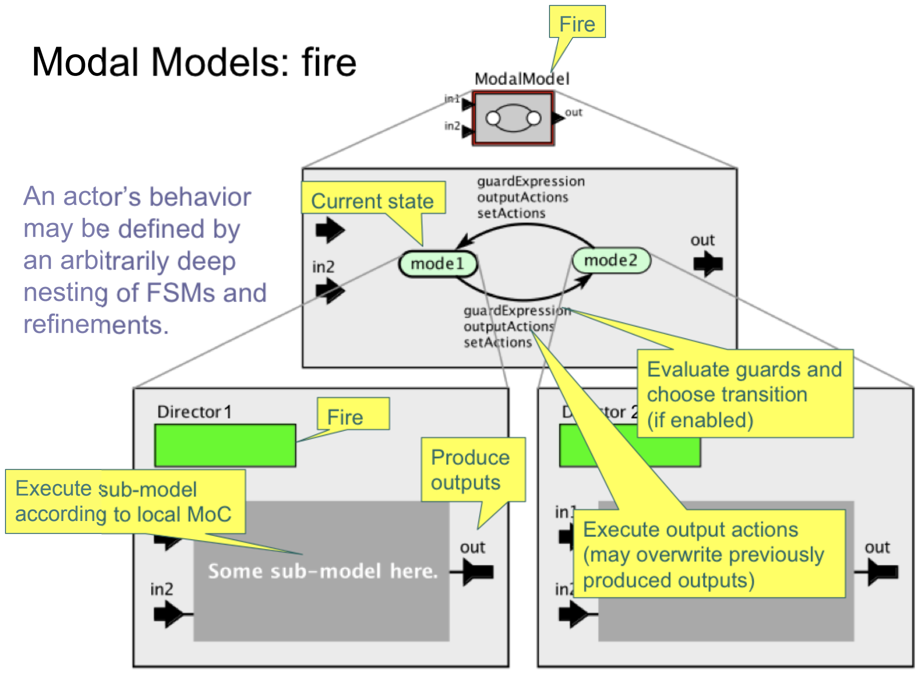
Modal Models

An actor's behavior may be defined by an arbitrarily deep nesting of FSMs and refinements.



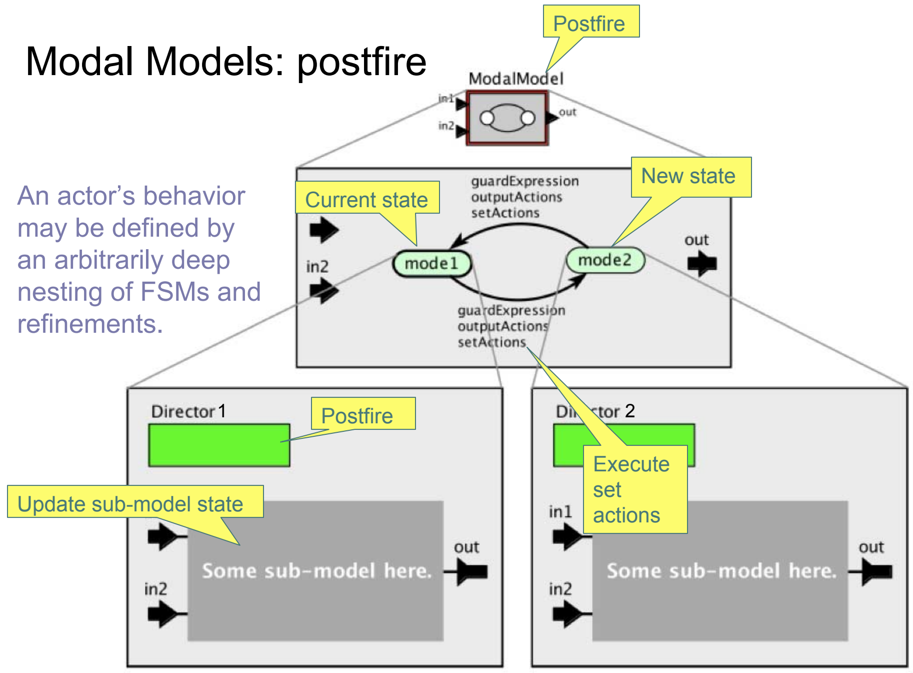
Modal Models: fire

An actor's behavior may be defined by an arbitrarily deep nesting of FSMs and refinements.



Modal Models: postfire

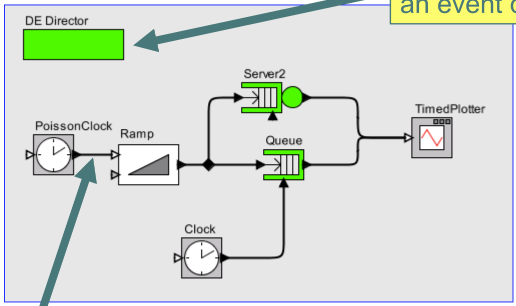
An actor's behavior may be defined by an arbitrarily deep nesting of FSMs and refinements.



[Discrete-event simulation: Verilog, VHDL, …]

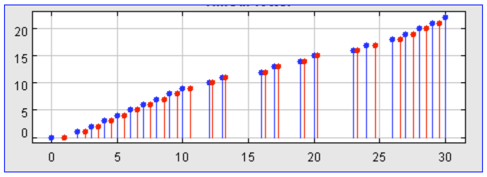
Discrete Event (DE)

Timed semantics using an event queue



In DE, actors send time-stamped events to one another, and events are processed in chronological order.

signal = sequence of **timed events**
timed event = value + time-stamp



29

Behaviors – untimed

$$F : S \times I \to O$$
$$P : S \times I \to S$$

Set of untimed traces:

$$s_0 \xrightarrow{x_0, y_0} s_1 \xrightarrow{x_1, y_1} s_2 \xrightarrow{x_2, y_2} \cdots$$

such that for all i :

$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i, x_i)$$

30

Behaviors – timed

$$F : S \times I \rightarrow O$$
$$P : S \times I \rightarrow S$$

States include special ***timer*** variables:

- Set to some positive value, "expire" when they reach 0

Set of timed traces:

Delays

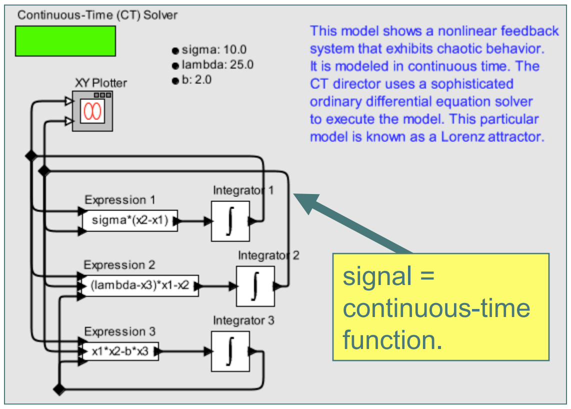
$$\begin{tikzcd} s_0 \arrow[r, "x_0, y_0, d_0"] & s_1 \arrow[r, "x_1, y_1, d_1"] & s_2 \arrow[r, "x_2, y_2, d_2"] & \cdots \end{tikzcd}$$

such that for all i :

$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i - d_i, x_i)$$
$$d_i \leq \min\{v \mid v \text{ is the value of a timer in } s_i\}$$

Ptolemy implementation is optimized with a single global time instead of separate timers

Continuous Time (CT)



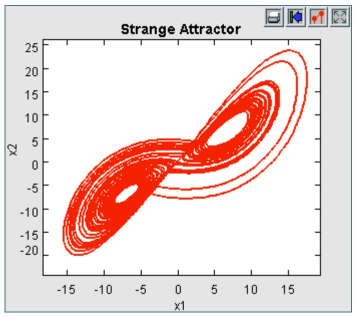
- sigma: 10.0
- lambda: 25.0
- b: 2.0

This model shows a nonlinear feedback system that exhibits chaotic behavior. It is modeled in continuous time. The CT director uses a sophisticated ordinary differential equation solver to execute the model. This particular model is known as a Lorenz attractor.

signal = continuous-time function.

Director includes an ODE solver.

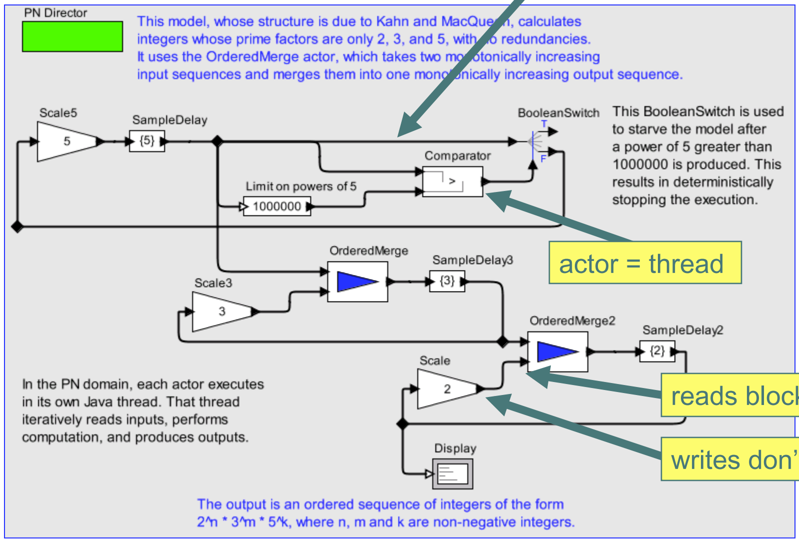
In CT, actors operate on continuous-time and/or discrete-event signals. An ODE solver governs the execution.



[Kahn '74, Kahn-MacQueen '77]

Kahn Process Networks (PN)

signal = stream = unbounded buffer



This model, whose structure is due to Kahn and MacQueen, calculates integers whose prime factors are only 2, 3, and 5, with no redundancies. It uses the OrderedMerge actor, which takes two monotonically increasing input sequences and merges them into one monotonically increasing output sequence.

This BooleanSwitch is used to starve the model after a power of 5 greater than 1000000 is produced. This results in deterministically stopping the execution.

In the PN domain, each actor executes in its own Java thread. That thread iteratively reads inputs, performs computation, and produces outputs.

The output is an ordered sequence of integers of the form 2^n * 3^m * 5^k, where n, m and k are non-negative integers.

actor = thread

reads block

writes don't

In PN, every actor runs in a thread, with blocking reads of input ports and non-blocking writes to outputs.

33

[CSP (Hoare), CCS (Milner), Reo (Arbab), …]

Rendezvous

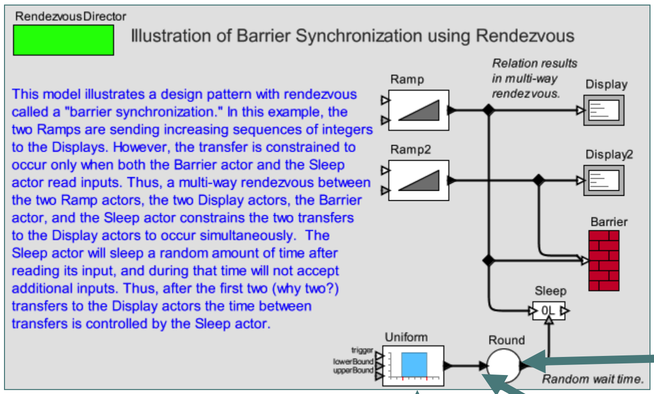


Illustration of Barrier Synchronization using Rendezvous

This model illustrates a design pattern with rendezvous called a "barrier synchronization." In this example, the two Ramps are sending increasing sequences of integers to the Displays. However, the transfer is constrained to occur only when both the Barrier actor and the Sleep actor read inputs. Thus, a multi-way rendezvous between the two Ramp actors, the two Display actors, the Barrier actor, and the Sleep actor constrains the two transfers to the Display actors to occur simultaneously. The Sleep actor will sleep a random amount of time after reading its input, and during that time will not accept additional inputs. Thus, after the first two (why two?) transfers to the Display actors the time between transfers is controlled by the Sleep actor.

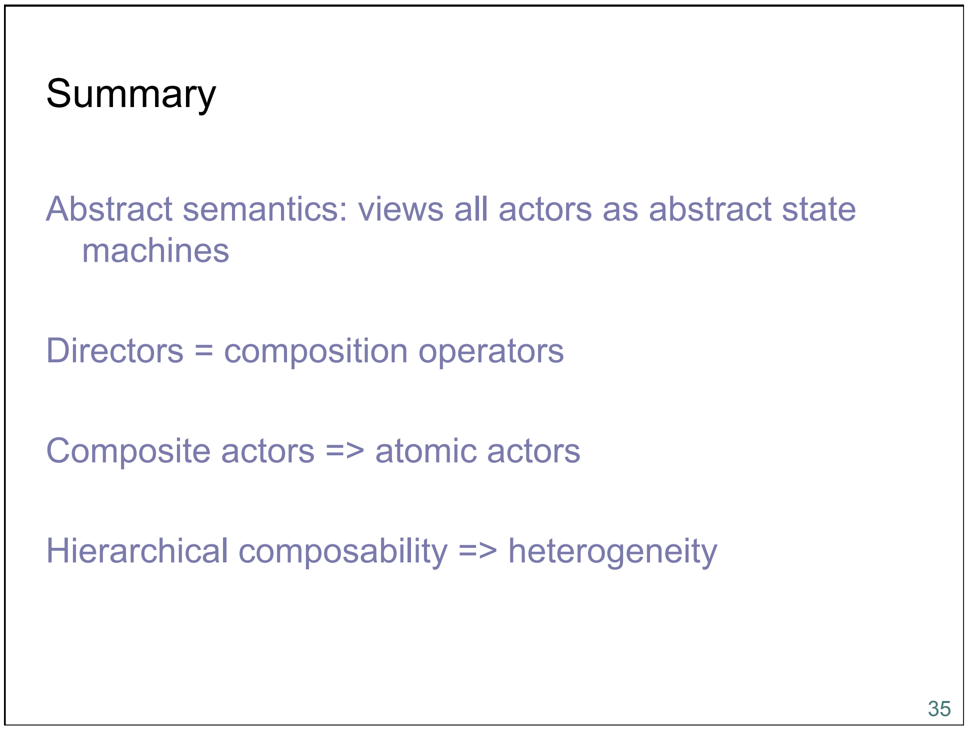
In Rendezvous, every actor runs in a thread, with blocking reads of input ports and blocking writes to outputs. Every communication is a (possibly multi-way) rendezvous.

writes block

actor = thread

reads block

34



Summary

Abstract semantics: views all actors as abstract state machines

Directors = composition operators

Composite actors => atomic actors

Hierarchical composability => heterogeneity

35

diagrams do not include deprecated methods or methods that are present in parent classes.

Arguments to a method or constructor are shown in parentheses, with the types after a colon, so for example, Attribute shows a single constructor that takes two arguments, one of type NamedObj and the other of type String.

A.2.2 Inheritance

Subclasses are indicated by lines with white triangles (or outlined arrow heads). The class on the side of the arrow head is the *superclass* or *base class*. The class on the other end is the *subclass* or *derived class*. The derived class is said to *specialize* the base class, or conversely, the base class to *generalize* the derived class. The derived class *inherits* all the methods shown in the base class and may *override* or some of them. In our object models, we do not explicitly show methods that override those defined in a base class or are inherited from a base class. For example, in figure 1.21, Entity has all the methods NamedObj, and may override some of those methods, but only shows the methods it adds. Thus, the complete set of methods of a class is cumulative. Every class has its own methods plus those of all its superclasses.

An exception to this is constructors. In Java, constructors are not inherited. Thus, in our class dia-

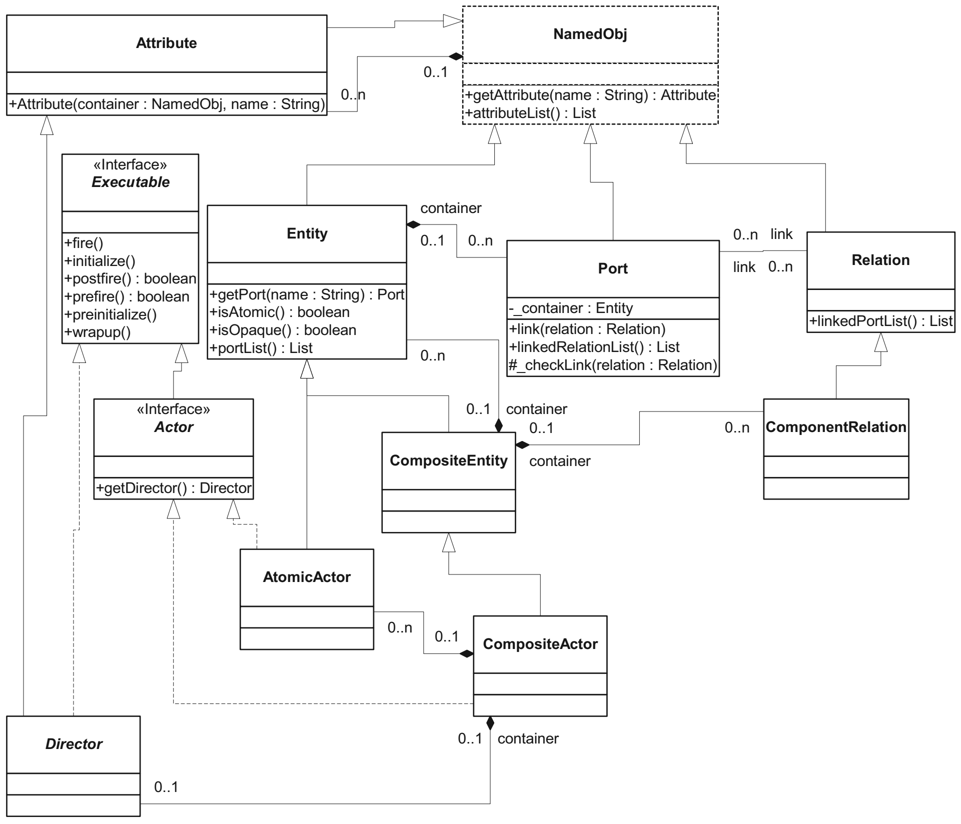


FIGURE 1.21. Simplified static structure diagram for some Ptolemy II classes. This diagram illustrates features of UML syntax that we use.

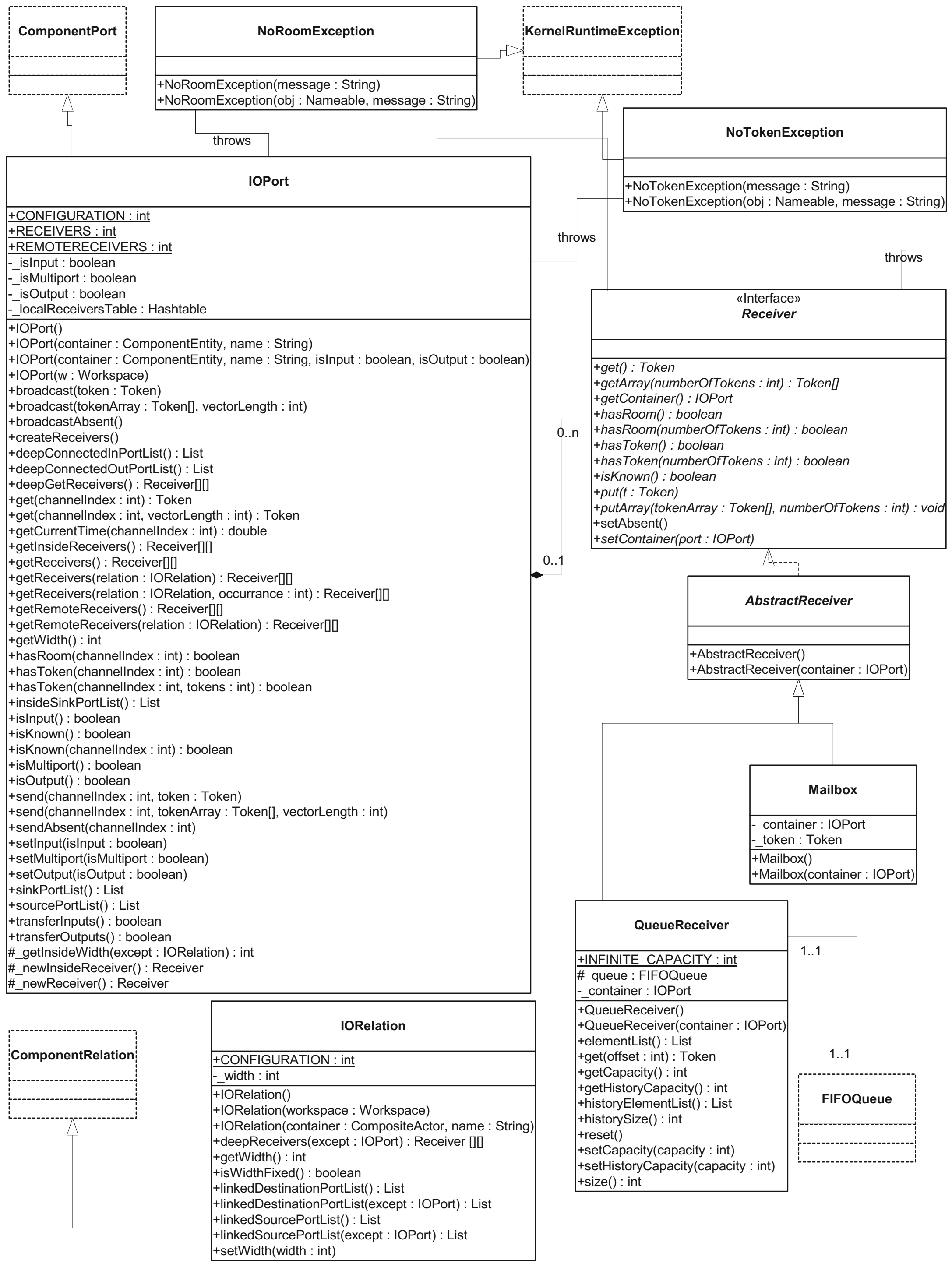

ComponentPort
NoRoomException
+NoRoomException(message : String)
+NoRoomException(obj : Nameable, message : String)
KernelRuntimeException
NoTokenException
+NoTokenException(message : String)
+NoTokenException(obj : Nameable, message : String)
throws
throws
throws
IOPort
+CONFIGURATION : int
+RECEIVERS : int
+REMOTERECEIVERS : int
-_isInput : boolean
-_isMultiport : boolean
-_isOutput : boolean
-_localReceiversTable : Hashtable
+IOPort()
+IOPort(container : ComponentEntity, name : String)
+IOPort(container : ComponentEntity, name : String, isInput : boolean, isOutput : boolean)
+IOPort(w : Workspace)
+broadcast(token : Token)
+broadcast(tokenArray : Token[], vectorLength : int)
+broadcastAbsent()
+createReceivers()
+deepConnectedInPortList() : List
+deepConnectedOutPortList() : List
+deepGetReceivers() : Receiver[][]
+get(channelIndex : int) : Token
+get(channelIndex : int, vectorLength : int) : Token
+getCurrentTime(channelIndex : int) : double
+getInsideReceivers() : Receiver[][]
+getReceivers() : Receiver[][]
+getReceivers(relation : IORelation) : Receiver[][]
+getReceivers(relation : IORelation, occurrance : int) : Receiver[][]
+getRemoteReceivers() : Receiver[][]
+getRemoteReceivers(relation : IORelation) : Receiver[][]
+getWidth() : int
+hasRoom(channelIndex : int) : boolean
+hasToken(channelIndex : int) : boolean
+hasToken(channelIndex : int, tokens : int) : boolean
+insideSinkPortList() : List
+isInput() : boolean
+isKnown() : boolean
+isKnown(channelIndex : int) : boolean
+isMultiport() : boolean
+isOutput() : boolean
+send(channelIndex : int, token : Token)
+send(channelIndex : int, tokenArray : Token[], vectorLength : int)
+sendAbsent(channelIndex : int)
+setInput(isInput : boolean)
+setMultiport(isMultiport : boolean)
+setOutput(isOutput : boolean)
+sinkPortList() : List
+sourcePortList() : List
+transferInputs() : boolean
+transferOutputs() : boolean
#_getInsideWidth(except : IORelation) : int
#_newInsideReceiver() : Receiver
#_newReceiver() : Receiver
«Interface»
Receiver
+get() : Token
+getArray(numberOfTokens : int) : Token[]
+getContainer() : IOPort
+hasRoom() : boolean
+hasRoom(numberOfTokens : int) : boolean
+hasToken() : boolean
+hasToken(numberOfTokens : int) : boolean
+isKnown() : boolean
+put(t : Token)
+putArray(tokenArray : Token[], numberOfTokens : int) : void
+setAbsent()
+setContainer(port : IOPort)
0..n
0..1
AbstractReceiver
+AbstractReceiver()
+AbstractReceiver(container : IOPort)
Mailbox
-_container : IOPort
-_token : Token
+Mailbox()
+Mailbox(container : IOPort)
QueueReceiver
+INFINITE_CAPACITY : int
#_queue : FIFOQueue
-_container : IOPort
+QueueReceiver()
+QueueReceiver(container : IOPort)
+elementList() : List
+get(offset : int) : Token
+getCapacity() : int
+getHistoryCapacity() : int
+historyElementList() : List
+historySize() : int
+reset()
+setCapacity(capacity : int)
+setHistoryCapacity(capacity : int)
+size() : int
1..1
1..1
FIFOQueue
ComponentRelation
IORelation
+CONFIGURATION : int
-_width : int
+IORelation()
+IORelation(workspace : Workspace)
+IORelation(container : CompositeActor, name : String)
+deepReceivers(except : IOPort) : Receiver [][]
+getWidth() : int
+isWidthFixed() : boolean
+linkedDestinationPortList() : List
+linkedDestinationPortList(except : IOPort) : List
+linkedSourcePortList() : List
+linkedSourcePortList(except : IOPort) : List
+setWidth(width : int)

FIGURE 2.2. Port and receiver classes for message passing under various communication protocols.

```java
package doc.tutorial.domains;

import java.util.Comparator;
import java.util.List;
import java.util.TreeSet;

import ptolemy.actor.Actor;
import ptolemy.actor.CompositeActor;
import ptolemy.actor.sched.Firing;
import ptolemy.actor.sched.NotSchedulableException;
import ptolemy.actor.sched.Schedule;
import ptolemy.actor.sched.Scheduler;
import ptolemy.actor.sched.StaticSchedulingDirector;
import ptolemy.data.IntToken;
import ptolemy.data.expr.Parameter;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.Entity;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.Locatable;
import ptolemy.kernel.util.NameDuplicationException;
import ptolemy.kernel.util.NamedObj;

/** Fire actors in left-to-right order. This director is a simple
 *  illustration of how to construct schedules for firing. It examines
 *  the location of the actor in a Vergil window, and on each
 *  invocation of the fire() method, fires each actor once,
 *  starting with the leftmost one and ending with the rightmost one.
 *  If two actors have the same horizontal position, then the order
 *  of their firings is arbitrary.
 *  <p>
 *  Note that this director will fire the actors forever. It may
 *  be difficult to stop the model executing.
 *  @author Edward A. Lee
 */
public class LeftRightDirector extends StaticSchedulingDirector {

    /** Constructor. A director is an Attribute.
     *  @param container The container for the director.
     *  @param name The name of the director.
     *  @throws IllegalActionException If the container cannot contain
     *   this director.
     *  @throws NameDuplicationException If the container already contains an
     *   Attribute with this name.
     */
    public LeftRightDirector(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);
        // Set the scheduler.
        setScheduler(new LeftRightScheduler(this, "LeftRightScheduler"));

        iterations = new Parameter(this, "iterations");
        iterations.setExpression("1");
    }

    /** Parameter specifying the number of iterations.
     *  If the value is 0 or less, then the model does not stop
     *  executing on its own.
     *  This is an int that defaults to 1.
     */
    public Parameter iterations;

    /** Override to initialize the iteration count. */
    public void initialize() throws IllegalActionException {
        super.initialize();
```

```java
        _iterationCount = 0;
    }

    /** Override to check the number of iterations. */
    public boolean postfire() throws IllegalActionException {
        boolean result = super.postfire();
        _iterationCount++;
        int iterationsValue = ((IntToken)iterations.getToken()).intValue();
        if (iterationsValue > 0 && _iterationCount >= iterationsValue) {
                return false;
        }
        return result;
    }

    /** Count of the number of iterations. */
    private int _iterationCount;

    /** Inner class defining the scheduler.
     */
    public static class LeftRightScheduler extends Scheduler {

        /** Constructor. A Scheduler is an Attribute,
         *  normally contained by a director.
         *  @param director The director that will use this scheduler.
         *  @param name The name of the scheduler.
         *  @throws IllegalActionException If the director cannot use
         *   this scheduler.
         *  @throws NameDuplicationException If the director already
         *   contains an Attribute with this name.
         */
        public LeftRightScheduler(LeftRightDirector director, String name)
                throws IllegalActionException, NameDuplicationException {
            super(director, name);
        }

        /** Return a left-to-right schedule. */
        protected Schedule _getSchedule() throws IllegalActionException,
                NotSchedulableException {
            // Get the director.
            NamedObj director = getContainer();
            // Get the container of the director.
            CompositeActor compositeActor
                    = (CompositeActor) (director.getContainer());
            // Get the actors to be fired by the director.
            List<Actor> actors = compositeActor.deepEntityList();
            // Create a sorted list of actors, sorted by
            // a specialized comparator.
            TreeSet<Actor> sortedActors
                    = new TreeSet(new LeftRightComparator());
            sortedActors.addAll(actors);
            // Construct a Schedule from the sorted list.
            Schedule schedule = new Schedule();
            for (Actor actor : sortedActors) {
                Firing firing = new Firing(actor);
                schedule.add(firing);
            }
            return schedule;
        }

        /** Inner class that implements a specialized comparator
         *  that compares the horizontal positions of the two
         *  arguments, which are assumed to actors.
         */
        public static class LeftRightComparator implements Comparator {
            public int compare(Object o1, Object o2) {
                // In case there is no location for an actor,
                // provide a default.
                double[] location1 = { Double.NEGATIVE_INFINITY,
                        Double.NEGATIVE_INFINITY };
                double[] location2 = { Double.NEGATIVE_INFINITY,
                        Double.NEGATIVE_INFINITY };
                // The location of the actor in Vergil is stored in an
                // Attribute that implements the Locatable interface.
                // Get a list of all such attributes, and use the first one
                // (normally there will be only one).
                List locations
                        = ((Entity) o1).attributeList(Locatable.class);
                if (locations.size() > 0) {
                    location1
                            = ((Locatable) locations.get(0)).getLocation();
                }
                locations = ((Entity) o2).attributeList(Locatable.class);
                if (locations.size() > 0) {
                    location2 = ((Locatable) locations.get(0)).getLocation();
                }
                if (location1[0] < location2[0]) {
                    return -1;
                } else if (location1[0] > location2[0]) {
                    return 1;
                } else {
                    // NOTE: It is not correct to return 0 if the x
                    // locations are the same because the actors may
                    // not be the same actor. A comparator has to be
                    // consistent with equals. We arbitrarily return -1,
                    // unless they are equal.
                    if (o1.equals(o2)) {
                        return 0;
                    }
                    return -1;
                }
            }
        }
    }
}
```

```java
package doc.tutorial.domains;

import ptolemy.actor.IOPort;
import ptolemy.actor.Receiver;
import ptolemy.domains.pn.kernel.PNDirector;
import ptolemy.domains.pn.kernel.PNQueueReceiver;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.Attribute;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;

/** This director modifies the PNDirector to support
 *  nonblocking reads. Specifically, it modifies the
 *  receiver so that hasToken() returns true only if the
 *  receiver has a token, unlike the original PNReceiver,
 *  where hasToken() always returns true. The price we
 *  pay for this flexibility is that models are no longer
 *  determinate.
 *  @author Edward A. Lee
 */
public class NondogmaticPNDirector extends PNDirector {

    /** Constructor. A director is an Attribute.
     *  @param container The container for the director.
     *  @param name The name of the director.
     *  @throws IllegalActionException If the container cannot
     *   contain this director.
     *  @throws NameDuplicationException If the container already
     *   contains an Attribute with the same name.
     */
    public NondogmaticPNDirector(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);
    }

    /** Return a new instance of the specialized receiver used by
     *  this director.
     */
    public Receiver newReceiver() {
        return new FlexibleReceiver();
    }

    /** Inner class defining the specialized receiver used by
     *  this director. This receiver overrides hasToken() to
     *  "tell the truth" about whether a token is present.
     */
    public static class FlexibleReceiver extends PNQueueReceiver {

        /** Override the base class to return true only if the
         *  receiver actually has a token.
         */
        public boolean hasToken() {
            IOPort port = getContainer();
            Attribute attribute = port.getAttribute("tellTheTruth");
            if (attribute == null) {
                return super.hasToken();
            }
            // Tell the truth...
            return _queue.size() > 0;
        }
    }
}
```