Representing Ptolemy II Models in Two-Dimensional Space

Ismael M. Sarmiento
Faculty Mentor: Dr. Edward Lee
Graduate Mentor: Stephen A. Neuendorffer
ismael.sarmiento@fiu.edu

Abstract

The Ptolemy II software is capable of modeling complex problems through a simple, actor-based user interface. Previously, the lack of a two-dimensional framework limited the results of Ptolemy models involving any sort of two-dimensional motion to simple position plots. The need to generate intuitive two-dimensional representations of models led us to develop such a framework, in which the results of models involving two-dimensional motion are now displayed as animations. These animations give the author a more concrete, real-world representation of the results, while the ability to simultaneously display position plots allows the author to maintain a more quantitative view of the data. Because the two-dimensional animation actors receive nearly the same input as the existing plotting actors, adding animation functionality to existing models poses minimal burden on the author.

1. Introduction

Ptolemy II is an extensive, open source software package in development at the University of California, Berkeley, which allows the simulation of concurrent systems based on user-created models [1]. The software is written entirely in Java, but the creation of models requires little or no programming knowledge, as Ptolemy II provides a robust graphical user interface for constructing models. The user need only drag the components required for the model into the editor window and specify how they interact with each other.

1.1. Representations of output

When Ptolemy II models are executed, the results can be displayed in a variety of ways. Strings of values can be printed to the screen, various plots of the results can be generated, and for models involving three-dimensional motion, three-dimensional animations can be created. While there are many options for displaying output beyond the ones described above, Ptolemy II lacks a framework for generating two-dimensional animations for models involving 2D motion. The goal of the project described herein is to create such framework, while imposing only a minimal burden on the authors of existing models who wish to add this functionality to their models.

1.2. The structure of Ptolemy II models

Ptolemy II models contain two essential types of components: directors and actors (Figure 1.1). Directors are components based on certain models of computation which drive Ptolemy II models. They determine when and in what order the individual actors are to execute, or "fire," as it is known in Ptolemy II. While Ptolemy II supports at least nine different directors, this project is concerned only with the GR director, which is responsible for driving models that display graphics.

Actors are components which represent objects or behaviors and interact with each other. The interactions between actors are defined by the connections between them, which are visually represented as lines routed from the output of one actor to the input of another. Ptolemy II currently contains hundreds of actors, ranging in function from simple mathematical operations to network communication, with more actors being developed every day. This project is essentially a collection of actors which allow the user to display the output of models as a two dimensional animation.

1.3. Diva

The actors in the two-dimensional graphics framework make use of another University of California, Berkeley software project, Diva. Diva wraps the standard Java2D graphics classes into a package that makes much of the functionality of Java2D more easily accessible to the programmer. All of the figures used by Ptolemy in the two-dimensional animations are based on Diva figures.

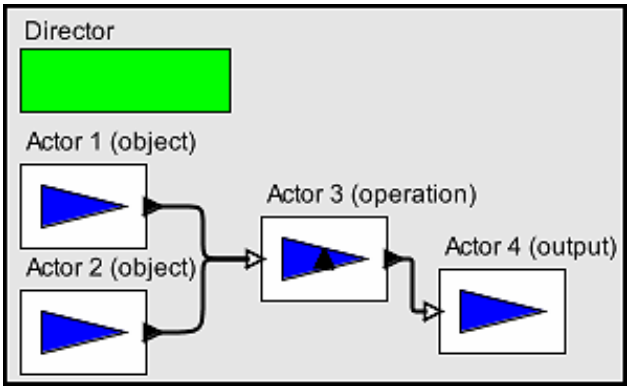

Figure 1.1 The structure of a Ptolemy II model.

3. Design principles

In designing the two-dimensional framework for Ptolemy II, it was important to insure that the package would be intuitive to use for those with experience in Ptolemy II. To this end, significant consideration was given to preserving many of the conventions used in the existing three-dimensional graphics framework at both the developer and user levels.

At the developer level, a class hierarchy for the 2D framework parallel to that used by the 3D framework was desirable. The parallel hierarchy between the 2D and 3D frameworks would insure that extensions to one framework could easily be implemented, if appropriate, in the other framework. The similar hierarchy would also allow authors familiar with the 3D framework to contribute functionality to the 2D framework without spending considerable time becoming familiar with the newer package.

At the user level, generating output with the 2D framework should be nearly identical to doing so with the 3D framework. There should be, where possible, a one-to-one correspondence between the number and type of actors and connections necessary for graphical output in the 2D and 3D frameworks.

A final design principle important in the development of the two-dimensional framework was that adding 2D animation functionality to existing models be as little a burden to the author as possible. Ideally, displaying the results of an existing model as a two-dimensional animation should be a matter of rerouting an existing connection to a new 2D animation sink (Figure 3.1), or simply adding another connection to maintain the existing output format while adding a 2D animation sink.

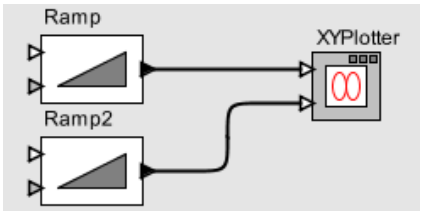

Figure 3.1a A model which generates output as a plot.

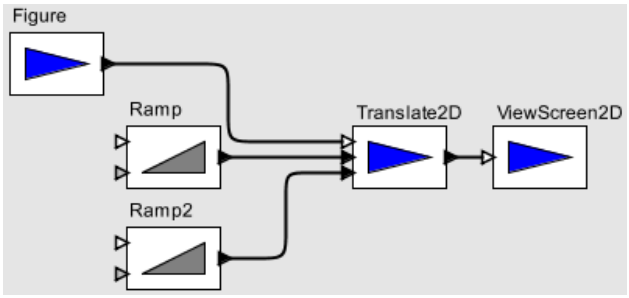

Figure 3.1b The model in Figure 3.1a converted to use animation for output.

4. Implementation of the 2D framework

As set forth in the previous section, the two-dimensional graphics framework developed in this project closely resembles the three-dimensional framework in structure. Both frameworks follow the same hierarchy down to the abstract GRActor class which defines the basic behavior of all graphical actors. Beneath the GRActor class, the two frameworks divide, and the 2D framework continues with the GRActor2D class on which all two-dimensional actors are based. Figure 4.1 shows the basic class hierarchy for the actors in the two-dimensional framework. While there are more classes backing the 2D framework than those shown in the figure, the displayed classes define all of the actors accessible to the user in the 2D framework (Figure 4.2). Their functions are described in the following sections.

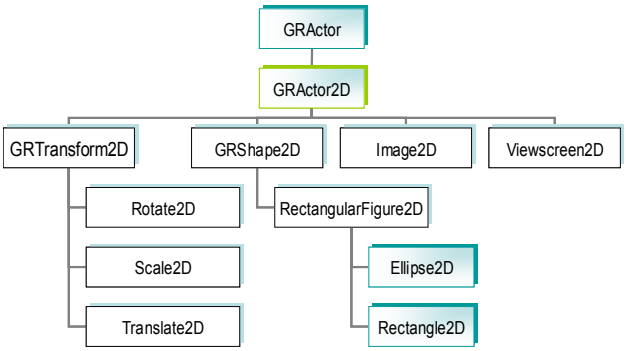

Figure 4.1 The 2D framework class hierarchy

4.1. GRActor and GRActor2D

GRActor2D and its super class, GRActor, define the way in which all 2D actors are created. They initialize and reset the scene graph connections of the actors to the view screen in which they are displayed. Like GRActor, GRActor2D is an abstract class which must be extended by actors in the two-dimensional framework.

4.2. Viewscreen2D

Viewscreen2D is a specialized class within the two-dimensional graphics framework used to display the animations on the screen. It is essentially a Java JFrame with a Diva JCanvas and multiple Diva layers. All figures which are to be displayed on the screen are first added to the view screen, where they are associated with an interactor for handling mouse and keyboard events which occur on the figure. All user events are first captured by the frame of the view screen, which has a listener associated with it. The listener determines what figure, if any, the event was intended for, and forwards the event to the interactor of the figure, which defines the action to take when a particular event is encountered. If, for example, the user clicks on a figure on the view screen, a message will make its way down to the interactor of the figure informing it the figure has been clicked, and the figure will be selected. Any mouse drags or arrow key presses would then translate the figure around the view screen. If the message was not intended for any figure—for example, the user wishes to translate or scale the view screen—then the interactor of the view screen handles the user event.

4.3. GRShape2D

The GRShape2D class is an abstract class which defines the behaviors and properties common to all basic, geometric shapes. These include the stroke and fill colors of the figure, the stroke width, and the connection of the figure to the view screen.

Currently, the only actor derived directly from GRShape2D is RectangularFigure2D, which is an abstract class serving as the parent of geometric figure actors based on rectangles, i.e. rectangles, rounded rectangles, and ellipses. RectangularFigure2D specifies that all derived figures have a rectangular shape with a position and size, and handles the updating of the figure on the view screen when any of these properties change.

4.4. Image2D

Image2D is derived directly from GRActor2D, and allows the user to display and manipulate figures based on image files in GIF, JPEG, or PNG format. The image files can be obtained from the local computer or from a remote host if a URL is provided. Image2D objects are identical to RectangularFigure2D objects in functionality, but replace the color and stroke properties of rectangular figures with a file path or URL.

4.5. Transform2D

Transform2D is the parent class of all affine transformations in the 2D framework. The subclasses of Transform2D include Rotate2D for rotating figures, Scale2D for scaling figures and Translate2D for translating figures.

4.5. Non-actor classes

The two-dimensional framework is supported by various classes which do not create actors. These non-actor classes include Viewscreen2DListener and FigureInteractor, and are used primarily for handling interaction between the user and the output of the model as described in section 4.2.

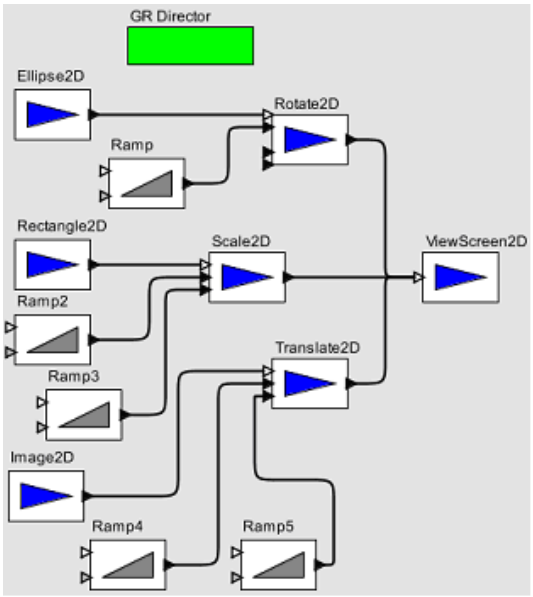

Figure 4.2a A Ptolemy II model using all of the actors of the two-dimensional framework.

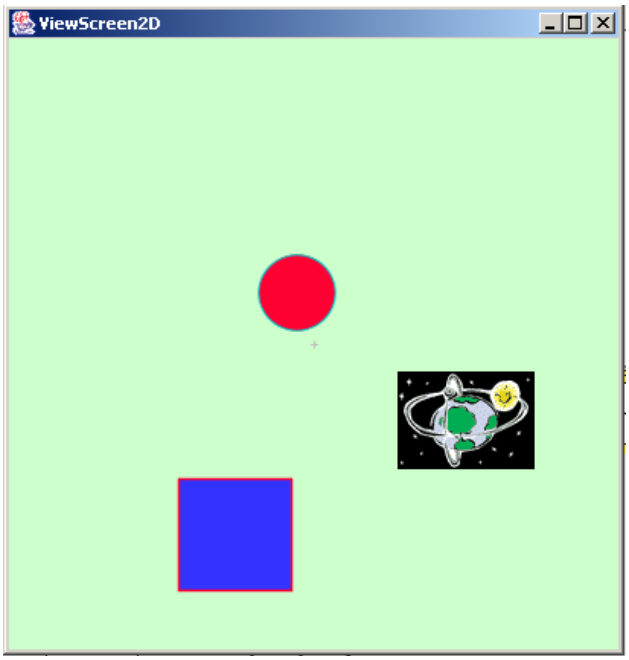

Figure 4.2b A static image of the output of the model in Figure 4.2a.

6. Improvements to the 2D framework

In its current state, the two-dimensional framework could benefit significantly from improvements in two major areas: speed and interaction. While the framework can handle basic animation models with no difficulty, it begins to struggle with more complex models involving multiple forms of output—for example, plots and view screens—rendering the contents of the view screen sluggishly. The problem appears to be inherent in the repainting mechanism of the 2D framework, and some additional attention to it could result in a much more useful framework.

The second problem is in the implementation of the interactive parts of the 2D framework. Currently, all interaction between the user and the view screen (see section 4.2) is essentially hard-coded into the Viewscreen2D, Viewscreen2DListener, and FigureInteractor classes. This strays from the general design of Ptolemy II, as any changes the user wishes to make to the actions taken when a particular event occurs requires additions and modifications to the underlying Java code. Ideally, an actor should be designed around the listener and interactor which processes user events and forwards the result to another actor which will handle the event according to the user's preference.

7. Conclusion

Extending the Ptolemy II software to include a two-dimensional framework for displaying the result of models as animations allows the creators of models to see their work running in a more intuitive fashion than simple position plots. Displaying the results of a model as a position plot and animation simultaneously provides the user with a general overview of the results, as a well quantitative data for analysis. While the 2D framework is a useful tool for researchers, in its current state, the 2D framework struggles to render animations at a reasonable speed when more complex models are used, and limits the interaction between the user and the view screen to hard-coded actions. Once these two issues are resolved, nearly every model involving some form of two dimensional motion will benefit from using the 2D framework to the display the results.

8. References

[1] Shuvra S. Bhattacharyya, Elaine Cheong, John Davis II, Mudit Goel, Christopher Hylands, Bart Kienhuis, Edward A. Lee, Jie Liu, Xiaojun Liu, Lukito Muliadi , Steve Neuendorffer, John Reekie, Neil Smyth, Jeff Tsay, Brian Vogel, Winthrop Williams, Yuhong Xiong, Yang Zhao, Haiyang Zheng, "Heterogeneous Concurrent Modeling and Design in Java, (Volume 1:Introduction to Ptolemy II) ", Technical Memorandum UCB/ERL M03/27, University of California, Berkeley, CA USA 94720, July 16, 2003.