1. An Overview of Ptolemy

Authors:

Joseph Buck
Edwin E. Goei
Soonhoi Ha
Alan Kamas
Ichiro Kuroda
Phil Lapsley
Edward A. Lee
David G. Messerschmitt

1.1. Introduction

Ptolemy is a very flexible foundation upon which to build simulation environments, where a key objective is the ability to combine these environments into multi-paradigm simulations as necessary. To effectively use existing simulation and synthesis tools and methodologies, different models of computation must be used for different parts of the overall design. Large systems often mix hardware, software, and communication subsystems. They may also combine hardware targets, including custom logic, processors with varying degrees of programmability, systolic arrays, and other multiprocessor subsystems. Tools supporting each of these components are different, using for instance dataflow principles, regular iterative algorithms, communicating sequential processes, control/dataflow hybrids, functional languages, and discrete-event system theory and simulation.

Ptolemy is a third-generation software environment that supports heterogeneous system specification, simulation, and design. It is an object-oriented framework within which diverse models of computation can co-exist and interact. In addition to the usual use of hierarchy to manage complexity, **Ptolemy** uses hierarchy to mix heterogeneous models of computation. The result is a unified software environment that extends the philosophy of mixed-mode circuit simulation up to the design and simulation of complex systems.

This work is an outgrowth of two previous generations of design environments, Blosim [Mes84a,b] and Gabriel [Bie90][Lee89b], that were aimed at digital signal processing applications. Both environments used a block-diagram data-flow paradigm for the description of the algorithms. To broaden the applicability beyond DSP, we see the need for other computational models, such as discrete-event scheduling, mixed compile-time and run-time scheduling, or computational models based on shared data structures. These are not supported very gracefully by Blosim or Gabriel. Most importantly, we see the need for a flexible simulation environment which is extensible to new computational models without re-implementation of the system.

Ptolemy uses an object-oriented programming methodology to support heterogeneity, and is programmed in C++. Data abstraction and polymorphism, two tenets of object-oriented programming, allow models of computation to be abstracted so that their differences are not visible from other domains. Our goal is to make the system non-dogmatic, in the sense that the environment itself does not impose any particular computational model, and it is extensible to new models by simply adding to the system and not modifying what is already there. The overall

software architecture is described in [Buc92b]. Further goals are to incorporate features that have been successful in Blosim or Gabriel, such as achieving modularity and reusability of user-programmed software modules, friendly graphical window interfaces, and code generation for targeting concurrent architectures rather than just simulation.

Each model of computation in **Ptolemy** is called a **`Domain`**. The system currently has a synchronous data-flow (SDF) domain [Bha92] [Buc91] [Lee87a] [Lee87b] [Lee91b] (like Gabriel), a dynamic dataflow (DDF) domain [Ha91] [Ha92] (like Blosim), a discrete-event (DE) domain, a Thor domain [Tho86] (for circuit simulation) and various code generation domains using dataflow semantics [Pin92]. We are working on several more domains that have not reached sufficient maturity to be included in this release [Buc92a] [Lee91a] [Lee92a] [Lee93]. The system is, however, already capable of simulating combinations of signal processing and communication networks (such as in packet speech and packet video) and combinations of behavioral and hardware simulation [Kal91] [Kal92], and is capable of synthesizing real-time implementations in assembly code for Motorola DSPs, as well as accelerated simulation code in C [Pin92].

The graphical interface (pigi) is based on VEM [Har86], a graphical editor for the OCT design database [Har86], both developed in the CAD group at Berkeley.

A word about notation: In this and other **Ptolemy** documentation, keywords from C++, or an abstract data type defined as a **`class`** in C++, or fragments of C++ code, are printed in the special font just used for "class". However, since the documentation is written by many people, we cannot promise perfect consistency.

1.2. Terminology

Ptolemy relies on a basic very flexible computational model of a simulation. The overall simulation is decomposed into software modules called *blocks*. These blocks at runtime are invoked in an order determined by a *scheduler*, and exchange data among themselves as they execute. From the user perspective there are two types of blocks: the **`Star`** and the **`Galaxy`**. The **`Star`** is elemental, in the sense that it is implemented by a user-provided code. There are also many pre-coded **`Star`**s in the **Ptolemy** library, but these should be viewed as examples, rather than as a comprehensive set. Adding new blocks is easy, so the system should be viewed as a programming environment, and not just as a monolithic tool to be used unmodified. A **`Galaxy`** is a block which internally contains **`Star`**s as well as possibly other **`Galaxy`**s. The **`Galaxy`** is thus a construct for producing a hierarchical description of the simulation. A **`Universe`** is a complete program, or application.

One of the key innovations in **Ptolemy** is the extension of the hierarchy of stars, galaxies, and universes to include objects called **`Wormhole`**s. A wormhole is named as such because from the outside, it looks monolithic, like a star, but inside, it looks more like a universe. The scheduler on the outside treats it exactly like a star, but internally it contains its own scheduler. The inside and outside schedulers need not abide by the same model of computation. This is the basic mechanism for supporting heterogeneity.

Data passes between blocks in discrete units called **`Particle`**s. For example, a particle in a signal processing system is often a signal sample, usually a floating-point value. But it can also be an image, for example, in a video sequence. In a communication system, it may be a packet consisting of multiple fields. In domains using dataflow principles, a particle may be called a *token*. Particles pass from one domain to another (into or out of a wormhole) through an

`EventHorizon.` The event horizon manages any format translations that may be required to stitch together two models of computation. User-defined particles are supported.

To get started, you can use the **Ptolemy** interactive graphical interface (pigi), described below, or the **Ptolemy** interpreter (ptcl). The interpreter is based on a language called Tcl, invented at Berkeley [Ous90]. Thus, both a graphical and textual specification language are available. It is unlikely that these two user interfaces will be the only ones developed for **Ptolemy**. They are, in fact, quite distinct from the **Ptolemy** kernel, precisely so that other types of user interfaces can be readily accepted. Future versions of the graphical interface will also use Tcl and its associated X11 toolkit, Tk [Ous91], which will lend it much more flexibility.

1.3. Installation

Ptolemy is a large software system that relies on a properly configured software environment. There are many things that can go wrong in getting **Ptolemy** running. For instance, the windowing system may not be the same one we use, or it may be used in a different way. There is also some configuration required by each user in order to use the graphical interface. The information for doing this is given in the Pigi document, section 2 of the Almagest. Here we give the basic information required to get from an FTP archive or distribution tape to being able to run the system.

1.3.1. Basic Ptolemy installation

First note that the approximate disk space requirements are (for the Sun-4 distribution; other distributions are roughly the same size):

Ptolemy: 49 Mbytes
Ptolemy (after you optionally remake): 64 Mbytes
Gnu subset: 16 Mbytes

Create a user ‘‘ptolemy’’, together with a home directory for the ‘‘ptolemy’’ user. Once the "ptolemy" user account has been created, log in or su to user ‘‘ptolemy’’. If you do not wish to create a user called ‘‘ptolemy’’, see below for an alternative.

If you are loading Ptolemy from a tape, do the following:

a. cd /tmp (or any directory where you have write permission and there is at least 22 Mbytes of free disk space).

b. Load the tape into your drive

c. mt -f /dev/nrst8 rewind
(This rewinds the tape -- change the device name if your tape drive has a different name.)

d. tar xf /dev/nrst8

If you have used FTP to down load the files, then cd to the directory that contains the "*tar.Z" files you downloaded via FTP.

Now, whether you’ve used tape or FTP, there should be a number of large "*tar.Z" files in your current directory. Proceed as follows:

1. zcat pt-0.4.doc.tar.Z | (chdir ~ptolemy/..; tar xf -)
(this uncompresses the documentation, changes directory the parent of the "ptolemy" user, and then creates all of the documentation files.)