2.4.3 Prior software

At every opportunity, we have built upon prior software, much of which we have been permitted to redistribute together with our Ptolemy distribution. We wish to gratefully acknowledge the following contributions:

- The **oct** tools, written by the CAD group at U. C. Berkeley, under the direction of Prof. Richard Newton, provide both the design database (**oct**), and the underpinnings of the graphical user interface (**vem**) [Har86]. **Vem** is an interactive graphical editor for **oct**. The flexibility of **oct**, which makes minimal assumptions about the data stored in the database, and the extensibility of **vem**, through its **rpc** interface, have allowed us to use this software in ways quite unexpected by the authors.
- Tcl/Tk, written primarily by John Ousterhout of U. C. Berkeley, has dramatically improved the user interface in Ptolemy. Both the textual, command-line interface and the graphical interface use Tcl, an embeddable control language [Ous90][Ous94]. Tcl is impressively robust and lightweight, making its integration relatively painless. The associated X window toolkit, Tk [Ous91], has been embedded in the graphical user interface. Since it is built upon Tcl, an interpreted language, we were able to make the user interface extensible even by casual users. We believe that this will dramatically enhance the usability and flexibility of the Ptolemy system.
- Although they do not form an integral part of Ptolemy, the Gnu tools, from the Free Software Foundation, have been instrumental in its development. Moreover, the ability to distribute the compiler used in the development of Ptolemy has been critical to the success of our dynamic linking mechanism. It enables us to distribute a compiled executable together with the compiler that generated it. Thus, users lacking the skill or patience to recompile the Ptolemy system can nonetheless take advantage of dynamic linking of new functional blocks. They can use the same version of the compiler used to generate the executable, even if that version of the compiler is not the one installed by default on their own system.
- Xgraph, written by David Harrison, of the CAD group at U. C. Berkeley, has provided the principal data display and presentation mechanism. Joe Buck modified this program only slightly, to accept binary input in addition to ASCII. Its flexibility and well conceived design have permitted us to use it for almost all data display. Only recently have we augmented it with Tk-based animated displays.
- Thor, which was written by the CAD group at Stanford University based on a kernel written at the University of Colorado, provided the starting point for our register-transfer-level circuit simulation domain. Seungjun Lee, of U. C. Berkeley, based the Ptolemy Thor domain on the original code. Some parts of the code, notably the logic analyzer display, are still used unmodified.

- The National Science Foundation (NSF) — (contract MIP-9201605) Microelectronics and Information Processing Systems Division (MIPS) and Computer and Computation Research Division.
- Office of Naval Technology (ONT), via the Naval Research Labs (NRL) N00014-92-K-2018
- State of California MICRO program.

Corporate sponsors

- AT&T
- Bell Northern Research (BNR)
- Comdisco Systems
- Dolby Laboratories
- Hitachi America
- Hitachi Ltd.
- Hughes Network Systems
- Hughes Research Laboratories
- Mentor Graphics
- Mitsubishi
- Motorola
- NEC
- Philips
- Rockwell
- Semiconductor Research Corporation (SRC)
- Sony
- Star Semiconductor

In-kind contributions have been received from

- Ariel
- Berkeley Camera Engineering
- Texas Instruments
- Spectrum Signal Processing
- Synopsys
- Signal Technology Inc. (STI)
- Xilinx

Other sponsors have contributed indirectly by supporting Gabriel, the predecessor.

lel assembly code for the Sproc multiprocessor DSP, and Kennard White wrote the CM-5 target, used to generate parallel code for the connection machine from Thinking Machines, Inc.

Seungjun Lee and Tom Parks wrote the CP domain. The Thor domain was written primarily by Seungjun Lee, with contributions from Asawaree Kalavade, Tom Parks, and Anders Wass. Asawaree Kalavade wrote the Silage domain, and Mike Williamson wrote the VHDL domains. Allen Lao developed the MQ domain. Ichiro Kuroda from NEC contributed to the development of the state handling mechanism.

The graphical interface was written by Edwin Goei, based on the **vem** program, written by David Harrison and Rick Spickelmier. It has been extensively modified by Alan Kamas who has been responsible for the incorporation of Tcl/Tk into Ptolemy. The GUI has been enhanced by Wan-Teh Chang, Wei-Jen Huang, Mario Silva and Kennard White. Andrea Casotto and Bill Bush have provided modifications and improvements to **vem**.

Several people have had a major impact on the development of Ptolemy through their major efforts on its predecessor, Gabriel. Phil Lapsley has had incalculable impact on the directory structure, project management, documentation, and code generation efforts in Ptolemy. The first version of the graphical interface was written by Holly Heine.

Many people have had an impact on the current release by contributing stars and/or demo programs. These include, in addition to all the people mentioned above, Egbert Ammicht (from AT&T Bell Labs), Rachel Bowers, Stefan DeTroch (from IMEC), Rolando Diesta, Erick Hamilton, Wei-Yi Li, John Loh, and Gregory Walter. Others had an indirect impact by contributing stars or demo programs to the predecessor program, Gabriel. These include Jeff Bier, Martha Fratt, Eric Guntvedt, Mike Grimwood, Wai-Hung Ho, Steve How, Jonathan Lee, Brian Mountford, Maureen O'Reilly, Andria Wong, and Anthony Wong.

Ptolemy is very much an ongoing project, with current efforts expected to be included in future releases. Participants in those efforts will be acknowledged when their work is included in a release.

2.4.2 Support

The Ptolemy project started in late 1989. A variety of sponsors have directly or indirectly supported the effort. The largest current sponsors are ARPA, together with the United States Air Force, under the technology base portion of the RASSP project (rapid prototyping of application-specific signal processors). The second largest sponsor, supporting much of the work on embedded system design, is the Semiconductor Research Corporation (SRC). Other government funding has been provided by the National Science Foundation (NSF), the State of California MICRO Program, and the Office of Naval Technology (ONT), via the Naval Research Labs (NRL). A number of companies have also sponsored various aspects of the project over the years. The complete list of sponsors (arranged alphabetically) is given below.

Government sponsors

- Advanced Research Projects Agencies (ARPA)
  Arlington, Virginia
- Solid State Electronics Directorate, Wright Laboratory
  Wright-Patterson AFB, Ohio 45433-5000

2.4 Acknowledgments

Ptolemy is a team effort in every sense. Here we acknowledge the key contributions, and apologize for inadvertent omissions.

2.4.1 Personnel

The overall coordinators are Prof. Edward A. Lee and Prof. David G. Messerschmitt of the EECS department at U. C. Berkeley, although there has also been involvement by the groups of Profs. Rabaey, Brodersen, Linnartz, and Kahn. Professional staff support has included Brian Evans, Alan Kamas, Christopher Hylands, Mary Stewart, Kirk Thege, and Kevin Zimmerman. Software organization and project management has been handled by Joseph Buck, Alan Kamas, Christopher Hylands, Phil Lapsley, and Kennard White.

Joseph Buck has been responsible for key management of the development of the kernel, and hence has impacted every aspect of Ptolemy. He also coordinated many of the contributions, and authored the BDF domain, the interpreter (**`ptcl`**), and the original **`ptlang`** preprocessor. He also designed the memory allocation system used by assembly language code generation domains. Special thanks to Synopsys for allowing Joe to work on the 0.5 release after joining the company.

Other key contributors to the kernel include Soonhoi Ha and Ichiro Kuroda. Soonhoi Ha also authored the DDF, DE, and CGC domains, including many of the basic stars and the basic domain interface, and also made extensive contributions to the CG domain, the kernel, and parallel schedulers of all types. Anindo Banerjea and Ed Knightly wrote the DE Scheduler that is based on the "calendar queue" mechanism developed by Randy Brown. This was based on code written by Hui Zhang. Other significant contributions to the kernel have been made by Wan-Teh Chang, Mike Chen, Paul Haskell, Asawaree Kalavade, Alireza Khazeni, Tom Parks, José Pino, and Kennard White. Mike Chen wrote the matrix classes and the matrix particles, based in part on a prototype supplied by Chris Yu (from the Naval Research Laboratories). Joe Buck, Asawaree Kalavade, Alan Kamas, and Alireza Khazeni wrote the fixed-point particle class. Paul Haskell created the image particle classes and developed many of the image and video signal processing demos. Philip Bitar had impact on the design of the DE domain and on the visual style used in the graphical interface.

Code generation domains are all based on a secondary kernel implemented as the CG domain. The principal creators of this domain are Joe Buck, Soonhoi Ha, Tom Parks, and José Pino. Kennard White made major extensions to the ptlang preprocessor to support code generation domains.

José Pino has been primarily responsible for assembly code generation domains, and Tom Parks for the C code generation domain, although extensive contributions have been made by Joe Buck, Soonhoi Ha, Praveen Murthy, S. Sriram, and Kennard White. Chih-Tsung Huang, with help from José Pino, ported all the assembly code generation stars from Gabriel. Many people had contributed to these, including Jeff Bier, Martha Fratt, Wai Ho, Steve How, Phil Lapsley, Maureen O'Reilly, and Anthony Wong.

Shuvra Bhattacharyya and Joe Buck wrote the loop scheduling mechanism, and Bhattacharyya contributed the Gantt chart display tool. The parallelizing schedulers were written by Gilbert Sih and Soonhoi Ha, with significant contributions from Joe Buck, Tom Parks, José Pino, and Kennard White. Praveen Murthy authored the Sproc domain, used to generate paral-

a “Run” command to invoke a simulation, it is the target that selects the scheduler to use, invokes the **`setup`** method of the scheduler to perform static analysis, and then invokes the **`run`** method of the scheduler to run the simulation.

A **`Domain`** is a collection of stars, schedulers, and targets, as shown in figure 2-4. Domain A is said to be a subdomain of B if its stars can be used within B. For example, the SDF domain is a subdomain of the DDF domain. Synchronous dataflow stars may be used within dynamic dataflow systems.

To get started, you can use the Ptolemy interactive graphical interface (**`pigi`**) or the Ptolemy interpreter (**`ptcl`**). The interpreter has a textual command language based on Tcl, a language developed at Berkeley [Ous90]. Both interfaces are extensible using Tcl, and the graphical interface is extensible using the associated X-window toolkit Tk [Ous91]. 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.

2.3 Organization of the documentation

The Ptolemy documentation is divided into four volumes. This volume, the first, is a user’s manual. It is sufficient for users who do not plan to extend the system by adding code. It includes brief documentation of the most commonly used domains, and brief summaries of stars, galaxies, and demonstration programs that are distributed with the system. More complete documentation of these stars, demos and galaxies is given in volume 2, the star atlas.

The third volume is a programmer’s manual. It includes chapters on writing new stars, writing targets, defining customized user interfaces by writing new Tcl/Tk code, and defining new domains. The fourth volume is the kernel manual. It details every C++ class defined in the Ptolemy kernel. It also gives full documentation for the classes supporting code generation. These classes provide the utilities used to build application-specific environments.

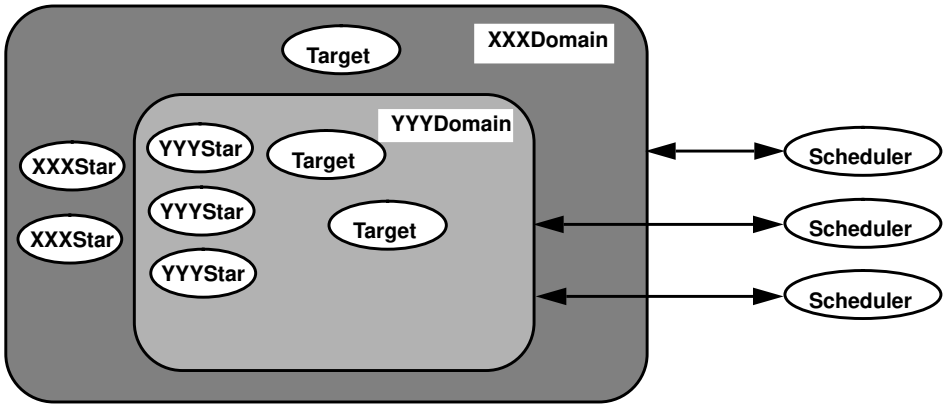

FIGURE 2-4: A Domain consists of a collection of stars, schedulers, and targets. The schedulers determine the operational semantics (e.g. dataflow) of the domain. Domain A is a subdomain of B if it’s stars can be used within B.

verse 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 galaxy in a different domain. 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, as shown in figure 2-2. For example, a particle in a signal processing system is often a signal sample, usually a floating-point value. But it can also be a matrix, or 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 (see figure 2-3). User-defined particles can pass through this interface as well.

A **Target** in Ptolemy manages a simulation or synthesis process. It is analogous, therefore, to the design flow managers found in some CAD frameworks. When the user issues

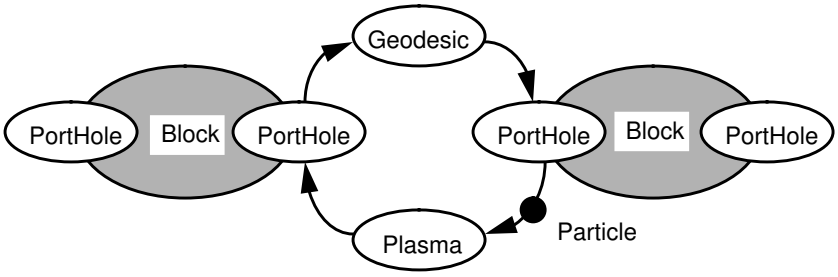

FIGURE 2-2: Blocks in Ptolemy exchange data in the form of Particles. Particles can be simple of complicated C++ objects. PortHoles define the interface to Blocks.

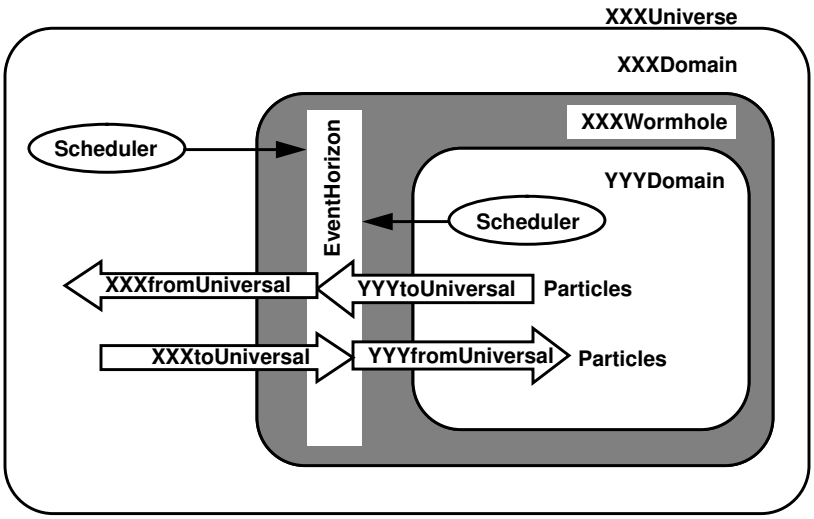

FIGURE 2-3: The EventHorizon provides an interface between the external and internal domains

interface") is based on **vem** [Har86], a graphical editor for the **oct** design database [Har86], both developed by the CAD group at Berkeley. A textual, command-line interface (call **ptcl**) is based on Tcl [Ous90][Ous94]. The textual interface is included as a subset of the graphical interface, augmented by the X-window toolkit Tk [Ous91][Ous94].

2.2 Terminology

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 or Unix[1] command lines, are printed in the special font just used for "class." However, since the documentation is written by many people, we cannot promise perfect consistency.

In Ptolemy, an overall description of a system is decomposed into software modules called **Block**s. These blocks are invoked at run time in an order determined by a **Scheduler**, and exchange data among themselves as they execute. The scheduler determines the operational semantics of a network of blocks, so schedulers are different in each domain. The Ptolemy kernel itself contains no usable scheduler, and thus imposes no operational semantics on the rest of the system.

From the user's perspective, there are two types of blocks: the **Star** and the **Galaxy**, as shown in figure 2-1. The **Star** is elemental, in the sense that it is implemented by 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. Volume 3 of the Almagest provides a detailed programmer's manual. A **Galaxy** is a block which internally contains **Star**s as well as possibly other **Galaxies**. The **Galaxy** is thus a construct for producing a hierarchical description of the simulation. A **Uni-**

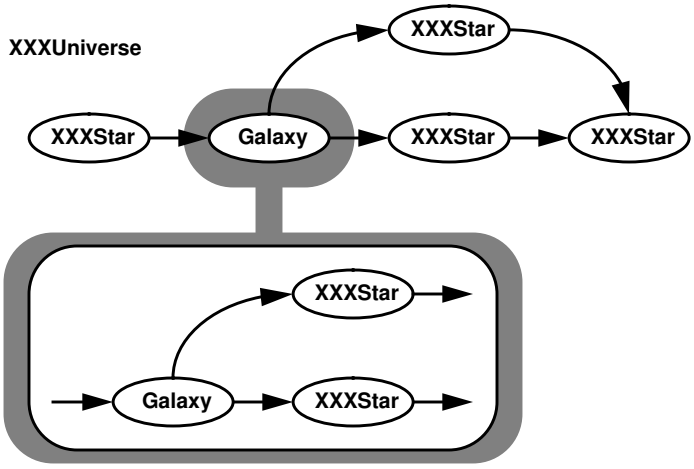

FIGURE 2-1: A hierarchical application in Ptolemy has Stars as leaf cells, Galaxies as composite blocks, and a Universe as a complete application. If the Domain is different inside a Galaxy than outside, then the Galaxy is implemented as Wormhole. A Wormhole hides the internal semantics from the outside Scheduler. In this figure, "XXX" represents some domain (such as SDF).

1. Unix is a trademark of Unix Systems Laboratories.

Ptolemy uses an object-oriented programming methodology to support heterogeneity, and it 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 to each other. Our goal has been 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 [Buc94]. We are in the process of expanding Ptolemy to cover higher levels of system specification as well as lower levels of system description. Higher levels of system specification will include graphical "makefile" dependencies, graphical specification of system parameters (e.g., in filter design [Whi93] and in two-dimensional decimator design [Eva93, ch. 8]), and mathematical models of systems [Eva93]. Lower-level descriptions will be able specify implementations of heterogeneous embedded systems [Bha93a,b,c] [Buc93b,c] [Bha94a,b] [Pin94]. These future goals will extend Ptolemy's capabilities as a rapid prototyping tool for embedded signal processors.

Each model of computation in Ptolemy is called a **`Domain`**. One of the most mature domains included in the current system is the synchronous dataflow (SDF) domain [Lee87a] [Lee87b], which is similar to that used in Gabriel. This domain is used for signal processing algorithm development, and has particularly good support for multirate algorithms [Buc91]. It has been used at Berkeley for instruction, both at the graduate and undergraduate level [Lee92]. A dynamic dataflow (DDF) domain extends SDF by allowing data-dependent flow of control, as in Blosim. A new domain called boolean dataflow (BDF) [Buc93a,b,c] implements a compile-time scheduler for dynamic dataflow graphs [Lee91a].

Several code-generation domains use dataflow semantics [Pin92][Mur93]. These domains are capable of synthesis of C code, assembly code for certain programmable DSPs [Won92], VHDL, and Silage [Kal93]. A significant part of the research that led to the development of these domains has been concerned with synthesizing code that is efficient enough for embedded systems [Bha93a,b,c][Bha94a,b][Buc93b,c]. A large amount of effort has also been put into the automatic parallelization of the code [Ha91] [Ha92][Sih93a,b], and on parallel architectures that take advantage of it [Sri93][Lee91b].

Ptolemy contains a number of simulation domains with discrete-event semantics. The DE domain is a generic discrete-event modeling environment, useful for simulating queueing systems, communication networks, and hardware systems. The Thor domain [Tho86] models circuits at the register-transfer level. Although it has not been developed sufficiently to compete with commercial simulators of this type, it is included in order to demonstrate hardware/software codesign methodologies that are being researched by our group [Kal93]. The communicating processes (CP) domain has related semantics, but it is especially useful for modeling large-scale systems at a high level of abstraction.

The system is capable, within certain limitations, of mixing domains within one design. Our work on hardware/software codesign illustrates this by combining a variety of synthesis domains and simulation domains [Kal93]. The system can also simulate combinations of signal processing and communication networks (such as in packet speech and packet video) [Wal92] [Has93]. A message queue (MQ) domain has been developed to model a centralized network controller in a large-scale cell-relay network simulation that mixes the SDF, DE, and MQ domains [Lao94].

The graphical interface (called **`pigi`**, an acronym for "Ptolemy interactive graphical

Chapter 2. An Overview of Ptolemy

2.1 Introduction

Ptolemy is a flexible foundation upon which to build prototyping environments. Several such environments have been built, including dataflow-oriented graphical programming for signal processing, discrete-event modeling of communication networks, register-transfer-level circuit design, synthesis environments for embedded software, and design assistants for hardware/software codesign. The Ptolemy system is fundamentally extensible. Users can create new component models, new design process managers, and even entirely new programming environments.

Although many users primarily work in only one environment, Ptolemy combines several environments for multi-paradigm simulation and synthesis. To use existing simulation and synthesis tools and methodologies effectively in large designs, different models of computation must be supported for different parts of the overall design. Large systems often mix hardware, software, and communication subsystems. The hardware subsystems may include pre-fabricated target hardware such as 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, finite-state machines, and discrete-event system theory and simulation. For large designs involving more than one of these, the design tools must support heterogeneity.

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 coexist 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 to the rapid prototyping of complex systems.

This work, started in January of 1990, is an outgrowth of two previous generations of design environments, Blosim [Mes84a,b] and Gabriel [Lee89][Bie90], that were aimed at digital signal processing. Both environments use dataflow semantics with block-diagram syntax for the description of algorithms. To broaden the applicability beyond DSP, the Ptolemy kernel does not build in dataflow semantics, but instead provides support for a wide variety of computational models, such as dataflow, discrete-event processing, communicating sequential processes, computational models based on shared data structures, and finite-state machines (under development). For these computational models, the Ptolemy kernel provides a mixture of compile-time and run-time scheduling techniques. Unlike Blosim or Gabriel, then, the Ptolemy kernel provides infrastructure that is extensible to new computational models without re-implementation of the system. The flexibility of Ptolemy is particularly important for enabling research in design methodology. In September of 1993, the Ptolemy project became part of the technology base portion of the RASSP project (rapid prototyping of application-specific signal processors), organized and sponsored by ARPA and the United States Air Force.