Chapter 1. An Overview of Ptolemy

1.1 Introduction

The core of Ptolemy is a compact software infrastructure upon which specialized design environments (called *domains*) can be built. The software infrastructure, called *the Ptolemy kernel*, is made up of a family of C++ class definitions. Domains are defined by creating new C++ classes derived from the base classes in the kernel.

Domains can operate in either of two modes:

- Simulation — A scheduler invokes code segments in an order appropriate to the model of computation.
- Code generation — Code segments in an arbitrary language are stitched together to produce one or more programs that implement the specified function.

The use of an object-oriented software technology permits a domain to interact with one another without knowledge of the features or semantics of the other domain. Thus, using a variety of domains, a team of designers can model each subsystem of a complex, heterogeneous system in a natural and efficient manner. These different subsystems can be nested to form a tree of subsystems. This hierarchical composition is key in specifying, simulating, and synthesizing complex, heterogeneous systems.

By supporting heterogeneity, Ptolemy provides a research laboratory to test and explore design methodologies that support multiple design styles and implementation technologies. A simple example is simulating the effects of transmitting compressed video and audio over an asynchronous transfer mode (ATM) network. The network will delay, drop, and reorder packets based on the congestion. Compression and decompression, however, work on the video and audio data, and the time associated with the data is not relevant to the signal processing. The simulation in this case is heterogeneous: the network processes discrete events (packets) with a notion of time, whereas the signal processing processes data independent of time. Other examples of heterogeneous systems include integrated control and signal processing architectures, mixed analog/digital simulation, and hardware/software codesign.

In short, Ptolemy is a flexible foundation upon which to build prototyping environments. The Ptolemy 0.7 release contains, for example, dataflow-oriented graphical programming for signal processing [Lee87a,b][Buc91][Buc93a,b,c], a multi-threaded process networks modeling environment [Par95], a synchronous/reactive programming framework [Edw97], discrete-event modeling of communication networks [Wal92][Hal93][Cha97], and synthesis environments for embedded software [Bha93a,b,c][Bha94a,b][Pin95]. We have also developed prototyping environments that are not released with Ptolemy 0.7, such as design assistants for hardware/software codesign [Kal93]. The Ptolemy system is fundamentally extensible, as we release all of the source code. Users can create new component models, new design process managers, and even entirely new programming environments.

1.2 History

Ptolemy is a third-generation software environment that started in January of 1990. It is an outgrowth of two previous generations of design environments, Blosim [Mes84a,b] and Gabriel [Lee89][Bie90], that were aimed at digital signal processing (DSP). 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. 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.

Since 1990, we have had seven major releases of Ptolemy, numbered 0.1 through 0.7. The zero indicates that Ptolemy is research software and not a commercial product. Between annual major releases, we put out one or two incremental releases. Our goal is to test our algorithms and methodologies in Ptolemy and to transfer them as quickly as possible to the public through freely distributable releases. Because of the critical mass of users of Ptolemy worldwide, a news group called `comp.soft-sys.ptolemy` was formed in 1994. The Ptolemy Web site `http://ptolemy.eecs.berkeley.edu/` went on-line in May of 1994.

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 Advanced Research Projects Agency (ARPA) and the United States Air Force. The Ptolemy part of the RASSP project was to research system-level design methodology for embedded signal processors. Our project aimed to develop formal models for such heterogeneous systems, a software environment for the design of such systems, and synthesis technologies for implementation of such systems. In the latter category, we have been concentrating on problems not already well addressed elsewhere, such as the synthesis of embedded software and the partitioning and scheduling of heterogeneous parallel systems. In 1997 the project became part of the DARPA Composite CAD program, and has shifted its focus towards more aggressively heterogeneous systems, including for example microelectromechanical components, and distributed adaptive signal processing systems.

We have transferred many of our research ideas to computer-aided design tool vendors such as Cadence, Hewlett Packard, and Synopsys. Cadence's Signal Processing Workshop (SPW) includes their version of our synchronous dataflow (SDF) domain and multirate dataflow schedulers. Cadence's Convergence environment (released in October, 1995) is Cadence's implementation of our ideas for heterogeneous simulation. Cadence has used Convergence to allow SPW and Bones (a discrete-event simulator) to cooperate in a simulation, just as the Ptolemy kernel has allowed the SDF domain and the Discrete-Event (DE) domain since 1990. Berkeley Design Technology has a similar cosimulation environment for SPW and Bones, but their implementation is based on the Ptolemy kernel. In June of 1997, Hewlett Packard announced plans to release a Ptolemy-based dataflow modeling environment that is integrated with their highly regarded analog, RF, and microwave circuit simulation software.

1.3 Ptolemy Kernel

The overall organization of the latest release of the Ptolemy system is shown in figure 1-1. A typical use of Ptolemy involves starting two Unix^{TM} processes, as shown in figure 1-1(a), by running `pigi` (Ptolemy interactive graphical interface). The first process contains the `vem` user interface and the `oct` design database [Har86], and the other process contains the Ptolemy kernel. An alternative is to run Ptolemy without the graphical user interface, as a single process, as shown in figure 1-1(b). In this case, the textual interpreter is based on the Tool Command Language, Tcl [Ous90][Ous94], and is called call `ptcl` for Ptolemy Tcl. It is possible to design other user interfaces for the system. We are releasing a preliminary version of a third interface called Tycho. In its current form, Tycho is best suited for language-sensitive editing and consoles for tools such as Matlab and Mathematica.

The executable programs `pigiRpc` or `ptcl` can be configured to include any subset of the available domains. The most recent picture of the domains that Berkeley has developed is shown in figure 1-2. Many different styles of design are represented by these domains. More are constantly being developed both at U.C. Berkeley and elsewhere, to experiment with or support alternative styles.

The Ptolemy kernel provides the most extensive support for domains where a design is represented as a network of blocks, as shown in figure 1-3. A base class in the kernel, called `Block`, represents an object in this network. Base classes are also provided for interconnecting blocks (`PortHole`) as well as for carrying data between blocks (`Geodesic`) and managing garbage collection efficiently (`Plasma`). Not all domains use these classes, but most current ones do, and hence can very effectively use this infrastructure.

Figure 1-3 shows some of the representative methods defined in these base classes. For example, note the *`initialize`*, *`run`*, and *`wrapup`* methods in the class `Block`. These provide an interface to whatever functionality the block provides, representing for example functions performed before, during, and after (respectively) the execution of the system.

Blocks can be hierarchical, as shown in figure 1-4. The lowest level of the hierarchy, as far as Ptolemy is concerned, is derived from a kernel base class called `Star`. A hierarchical block is a `Galaxy`, and a top-level system representation is a `Universe`.

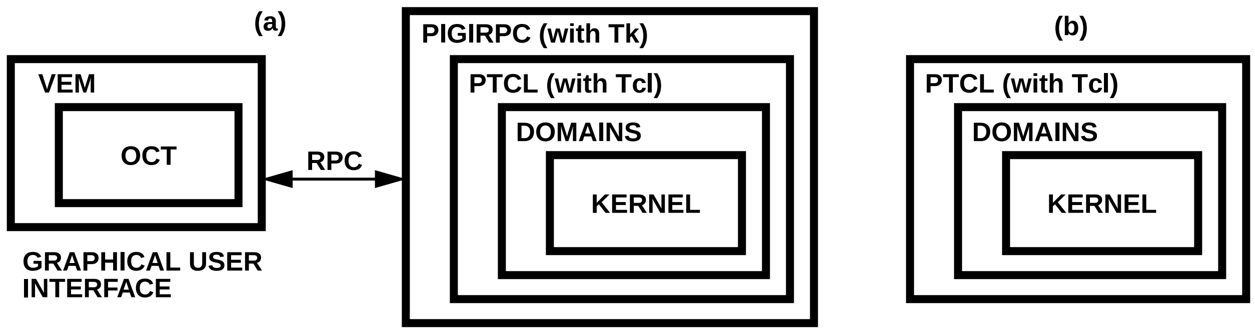


FIGURE 1-1: The overall organization of Ptolemy version 0.7, showing two possible execution styles: (a) graphical interface and (b) textual interface.

1.4 Models of Computation

The Ptolemy kernel does not define any model of computation. In particular, although the Berkeley team has done quite a bit of work with dataflow domains in Ptolemy, every effort has been made to keep dataflow semantics out of the kernel. Thus, for example, a network of blocks could just as easily represent a finite-state machine, where each block represents a state. It is up to a particular domain to define the semantics of a computational model.

Suppose we wish to define a new domain, called XXX. We would define a set of C++

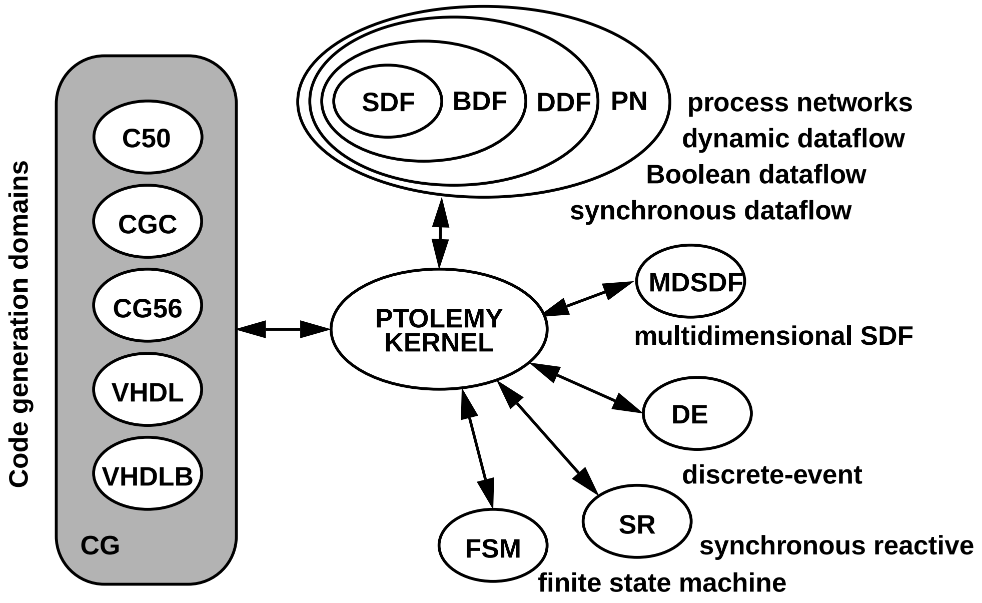


FIGURE 1-2: Domains available with Ptolemy 0.7

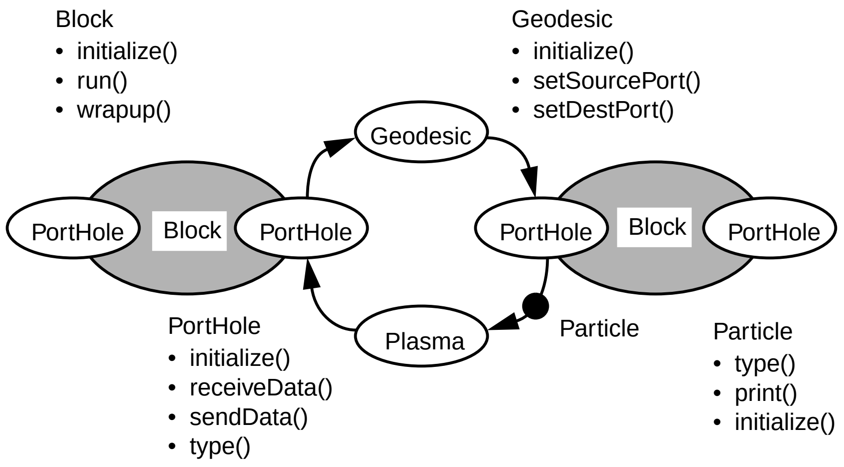
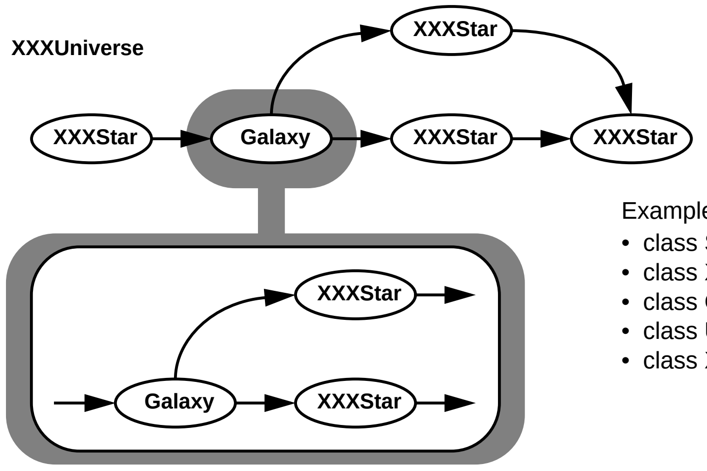


FIGURE 1-3: Block objects in Ptolemy can send and receive data encapsulated in Particles through Portholes. Buffering and transport is handled by the Geodesic and garbage collection by the Plasma. Some methods are shown.

classes derived from kernel base classes to support this domain. These classes might be called `XXXStar`, `XXXUniverse`, etc., as shown in figure 1-4.

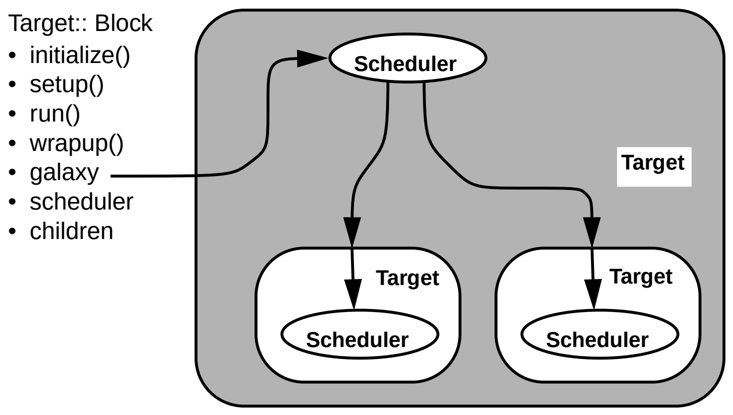
The semantics of a domain are defined by classes that manage the execution of a specification. These classes could invoke a simulator, or could generate code, or could invoke a sophisticated compiler. The base class mechanisms to support this are shown in figure 1-5. A `Target` is the top-level manager of the execution. Similar to a `Block`, it has methods called `setup`, `run`, and `wrapup`. To define a simulation domain called `XXX`, for example, one would define at least one object derived from Target that runs the simulation. As suggested by figure 1-5, a Target can be quite sophisticated. It can, for example, partition a simulation for parallel execution, handing off the partitions to other Targets compatible with the domain.

A Target will typically perform its function via a Scheduler. The Scheduler defines the operational semantics of a domain by controlling the order of execution of functional modules. Sometimes, schedulers can be specialized. For instance, a subset of the dataflow model of computation called synchronous dataflow (SDF) allows all scheduling to be done at com-



Examples of Derived Classes
- class Star:: Block
- class XXXStar:: Star
- class Galaxy:: Block
- class Universe:: Galaxy, Runnable
- class XXXUniverse:: Universe

FIGURE 1-4: A complete Ptolemy application (a Universe) consists of a network of Blocks. Blocks may be Stars (atomic) or Galaxies (composite). The "XXX" prefix symbolizes a particular domain (or model of computation).



Target:: Block
- initialize()
- setup()
- run()
- wrapup()
- galaxy
- scheduler
- children

FIGURE 1-5: A Target, derived from Block, manages a simulation or synthesis execution. It ca invoke it's own Scheduler on a Galaxy, which can in turn invoke Schedulers sub-Targets.

pile time. The Ptolemy kernel supports such specialization by allowing nested domains, as shown in figure 1-6. For example, the SDF domain (see figure 1-2) is a subdomain of the BDF domain. Thus, a scheduler in the BDF domain can handle all stars in the SDF domain, but a scheduler in the SDF domain may not be able to handle stars in the BDF domain. A domain may have more than one scheduler and more than one target.

1.5 Dataflow Models of Computation

One of the most mature domains included in the current system is the synchronous dataflow (SDF) domain [Lee87a,b], which is similar to that used in Gabriel. This domain is used for signal processing and communications algorithm development, and has particularly good support for multirate algorithms [Buc91]. It has been used at Berkeley for instruction, at both the graduate and undergraduate level [Lee92]. A dynamic dataflow (DDF) domain extends SDF by allowing data-dependent flow of control, as in Blosim. Boolean dataflow (BDF) [Buc93a,b,c] has 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 [Lee91b][Sri93].

A generalization of dataflow, called Kahn process networks [Kah74], has been realized by Tom Parks in the PN domain [Par95].

1.6 Discrete-Event Models of Computation

A number of simulation domains with discrete-event semantics has been developed for Ptolemy, but only the DE domain is released with Ptolemy 0.7. The DE domain is a generic discrete-event modeling environment, useful for simulating queueing systems, communication networks, and hardware systems. The discrete-event domains no longer released with

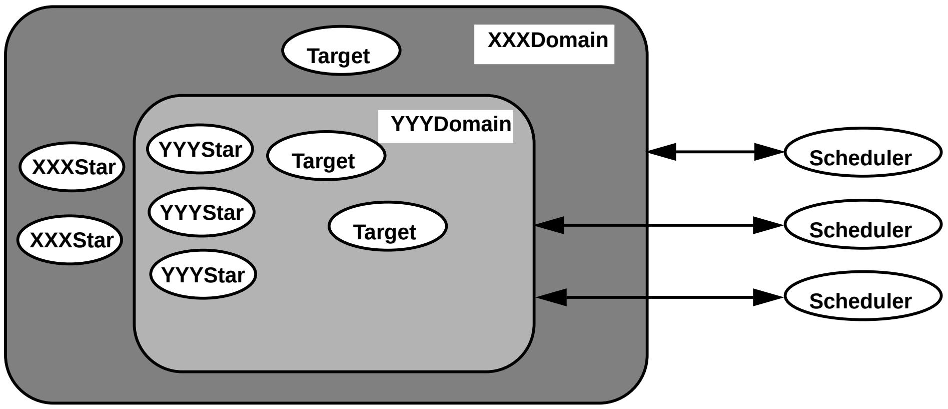


FIGURE 1-6: A Domain (XXX) consists of a set of Stars, Targets and Schedulers that support a particular model of computation. A sub-Domain (YYY) may support a more specialized model of computation.

Ptolemy 0.7 are Thor [Tho88] for modeling circuits at the register-transfer level [Kal93], communicating processes (CP) for modeling large-scale systems at a high level of abstraction, and message queue (MQ) for modeling a centralized network controller in a large-scale cell-relay network simulations [Lao94].

1.7 Synchronous Reactive Modeling

The software analogy of synchronous digital circuits has been realized by Stephen Edwards in the SR domain [Edw97]. This model of computation is better suited than dataflow to control-intensive applications, and is more efficient than DE.

1.8 Finite State Machines

Another approach to designing control-intensive applications is to mix the new FSM domain with dataflow, DE, or (in future releases) SR. The FSM domain is still very new and has many limitations, but we believe that for the long term, it provides one of the most exciting developments in the Ptolemy software.

1.9 Mixing Models of Computation

Large systems often mix hardware, software, and communication subsystems. The hardware subsystems may include pre-fabricated components, such as custom logic, processors with varying degrees of programmability, systolic arrays, and multiprocessor subsystems. Tools supporting each of these components are different, possibly using dataflow principles, regular iterative algorithms, communicating sequential processes, control/dataflow hybrids, functional languages, finite-state machines, and discrete-event system theory and simulation.

In Ptolemy, domains can be mixed and even nested. Thus, a system-level description can contain multiple subsystems that are designed or specified using different styles. The kernel support for this is shown in figure 1-7. An object called `XXXWormhole` in the `XXX` domain is derived from `XXXStar`, so that from the outside it looks just like a primitive in the `XXX`

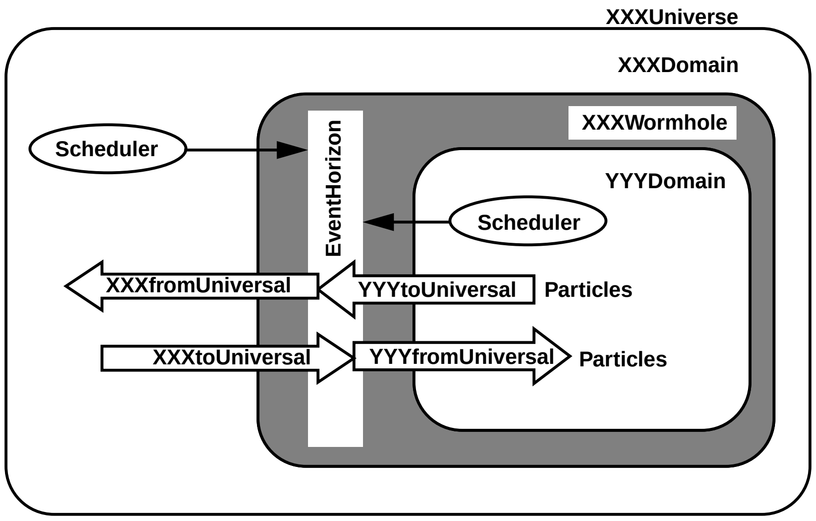


FIGURE 1-7: The universal EventHorizon provides an interface between the external and internal domains.

domain. Thus, the schedulers and targets of the `XXX` domain can handle it just as they would any other primitive block. However, inside, hidden from the `XXX` domain, is another complete subsystem defined in another domain, say `YYY`. That domain gets invoked through the `setup`, `run`, and `wrapup` methods of `XXXWormhole`. Thus, in a broad sense, the wormhole is polymorphic. The wormhole mechanism allows domains to be nested many levels deep, e.g. one could have a DE domain within an SDF domain within a BDF domain. The FSM domain is designed to always be used in combination with other domains.

1.10 Code Generation

Domains in figure 1-2 are divided into two classes: simulation and code generation. In simulation domains, a scheduler invokes the run methods of the blocks in a system specification, and those methods perform a function associated with the design. In code generation domains, the scheduler also invokes the run methods of the blocks, but these run methods synthesize code in some language. That is, they generate code to perform some function, rather than performing the function directly. The Target then is responsible for generating the connecting code between blocks (if any is needed). This mechanism is very simple, and language independent. We have released code generators for C, Motorola 56000 assembly, and VHDL languages, as show in figure 1-2.

An alternative mechanism that is supported but less exploited in current Ptolemy domains is for the target to analyze the network of blocks in a system specification and generate a single monolithic implementation. This is what we call compilation. In this case, the primitive blocks (`Stars`) must have functionality that is recognized by the target. In the previous code generation mechanisms, the functionality of the blocks is arbitrary and can be defined by the end user.

1.11 Conclusion

In summary, the key idea in the Ptolemy project is to mix models of computation, implementation languages, and design styles, rather than trying to develop one, all-encompassing technique. The rationale is that specialized design techniques are (1) more useful to the system-level designer, and (2) more amenable to high-quality high-level synthesis of hardware and software. The Ptolemy kernel demonstrates one way to mix tools that have fundamentally different semantics, and provides a laboratory for experimenting with such mixtures.

1.12 Current Directions

Since early 1995, a significant part of the Ptolemy project personnel have been pursuing models of computations for control-intensive computation, particularly in combination with compute-intensive subsystems, and mapping computation onto distributed architectures. The two primary models for control-intensive computation are finite state machines and a synchronous/reactive systems.

In late 1996, we shifted the focus of the project towards the design of distributed, network-aware, adaptive applications. We expect that future releases of our software will be network-savvy, including transparent HTTP support and mutable and migratable computations. Fundamental work in the semantics of models of computation will of course continue to fuel experiments with new domains and code generation techniques.

1.13 Organization of the documentation

The Ptolemy documentation is divided into three 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.

The second 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 third 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.

1.14 Acknowledgments

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

1.14.1 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, Kahn, Sangiovanni, and Gray. Professional staff support has included Brian Evans, Alan Kamas, Christopher Hylands, John Reekie, Mary Stewart, Kirk Thege, and Kevin Zimmerman. Software organization and project management has been handled by Joseph Buck, Brian Evans, Alan Kamas, Christopher Hylands, Phil Lapsley, José Pino, John Reekie, 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 wrote 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. Special thanks to Joe for his work on the 0.6 release.

Other key contributors to the kernel include Soonhoi Ha and Ichiro Kuroda. Soonhoi Ha also wrote 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). Mike Chen also developed the MDSDF domain. 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. Brian Evans developed the interfaces to MATLAB and Mathematica, with help from Steve Eddins at The MathWorks and Steve Gu, respectively.

All code generation domains are based on a secondary kernel implemented as the CG domain. Its principal creators 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, Christopher Hylands, Praveen Murthy, S. Sriram, and Kennard White. Chih-Tsung Huang, with help from José Pino, ported many of the assembly code generation stars from Gabriel. Many people had contributed to the Gabriel stars, including Jeff Bier, Martha Fratt, Wai Ho, Steve How, Phil Lapsley, Maureen O'Reilly, and Anthony Wong. Brian Evans and Luis Gutierrez have enhanced the Motorola 56000 stars, demonstrations, and targets, and S. Sriram has done the same for the Motorola 96000 stars. Patrick Warner wrote the C code generation target for the Network Of Workstations distributed operating system by Prof. Patterson's group at U.C. Berkeley.

Shuvra Bhattacharyya and Joe Buck wrote the loop scheduling mechanism, and Bhattacharyya contributed the Gantt chart display tool. The parallel schedulers were written by Gilbert Sih and Soonhoi Ha, with significant contributions from Joe Buck, Tom Parks, José Pino, and Kennard White. Praveen Murthy wrote the Sproc domain, used to generate parallel 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. Mike Williamson wrote the VHDL domains. Ichiro Kuroda from NEC contributed to the state handling mechanism.

The graphical user 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 Cassotto and Bill Bush have provided modifications and improvements to `vem`.

Christopher Hylands, Edward Lee, and John Reekie are the primary architects of the Tycho interface [Hyl97]. Tycho, named after the astronomer Tycho Brahe, is written in [Incr Tcl], an object-oriented extension of Tcl by Michael J. McLennan at AT&T Bell Labs. Significant development of Tycho has been contributed by Kevin Chang, Joel King, and Cliff Cordiero. Wan-Teh Chang and Bilung Lee have developed graphical editors for finite state machines. Code by Joseph Buck, Alan Kamas, and Douglas Niehaus originally written for pigi have been reused in the Tycho kernel. Some contributions to Tycho were made by Brian Evans.

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 will be acknowledged when their work is included in a release.

1.14.2 Support

The Ptolemy project is currently supported by the Defense Advanced Research Projects Agency (DARPA), the State of California MICRO program, and the following companies: The Alta Group of Cadence Design Systems, Dolby Laboratories, Hewlett Packard, Hitachi, Hughes Space and Communications, LG Electronics, Lockheed Martin ATL, NEC, Philips, Rockwell, and the Semiconductor Research Corporation.

Funding at earlier stages of the project was also provided by the National Science Foundation (NSF), the Office of Naval Technology (ONT), via the Naval Research Labs (NRL), AT&T, Bell Northern Research (BNR), Hughes Network Systems, Hughes Research Laboratories, Mentor Graphics, Mitsubishi, Motorola, Sony, and Star Semiconductor.

In-kind contributions have been made by Ariel, Berkeley Camera Engineering, Philips, Spectrum Signal Processing, Synopsys, Signal Technology Inc. (STI), Texas Instruments, Wolfram Research, Inc., and Xilinx.

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

1.14.3 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` [Har86] and the graphical editor `vem` [Har86] 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 unexpected by the authors.
- Tcl/Tk, architected by Prof. John Ousterhout of U.C. Berkeley, has improved the user interface in Ptolemy. The textual command-line Tcl interface [Ous90][Ous94] is the basis for the Ptolemy interpreter `ptcl`, and the graphics toolkit Tk [Ous94] is the basis for interactive graphics. Tcl serves a scripting language to control Ptolemy runs, and as an interpreter to compute parameters. Since Tcl is a scripting language, casual users have been able to extend Ptolemy's interface. Tcl is robust and lightweight.
- [Incr Tcl], an object-oriented extension of Tcl written by Michael J. McLennan at AT&T Bell Labs is used in Tycho.
- The Gnu tools, from the Free Software Foundation, have been instrumental in Ptolemy's development. 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.