Chapter 4. Introduction to Domains, Targets, and Foreign Tool Interfaces

Authors: *Joseph T. Buck*
Brian L. Evans
Soonhoi Ha
Asawaree Kalavade
Edward A. Lee
Thomas M. Parks

Michael C. Williamson

Other Contributors: *The entire Ptolemy team*

4.1 Introduction

The Ptolemy software architecture is described in Chapter 1 and shown in Figure 1-2. The Ptolemy kernel provides a basic set of C++ classes and functions used by the rest of Ptolemy, but it does not implement any particular model of computation. Instead, a model of computation is defined by a domain. A domain defines the semantics of the model, but not how the computations are performed. The computations can be performed using one or more implementation languages, such as C++, C, MATLAB, and VHDL. A target coordinates the scheduling and implementation of algorithms described in a particular domain. As part of the coordination, a target may provide an interface to software (compiler, assembler, simulator, etc.) or hardware. A typical domain supports many different types of schedulers and many different implementation technologies, which is possible by having many different targets. Over twenty domains have been developed for Ptolemy, and 14 are released in Ptolemy 0.7; of these, nine support multiple targets.

In Ptolemy, a complex system is specified as a hierarchical composition (nested tree structure) of simpler subsystems. Each subsystem is modeled by a domain. A subsystem (also called a galaxy) can internally use a different domain than that of its parent or children. In mixing domains, the key is to ensure that at the interface, the child galaxy obeys the semantics of the parent domain. This interface is called a wormhole. Ptolemy does not yet make the wormhole mechanism foolproof. Any domain can be used at the top level.

As shown in Figure 1-2, Ptolemy consists of dataflow, discrete-event, and control-oriented families of domains. The Synchronous Dataflow (SDF) and Discrete-Event (DE) domains are the most mature. In terms of semantics, the Discrete-Event domains are the furthest from the dataflow domains in the 0.7 distribution. Other domains with semantics very different from dataflow, are the Finite State Machine (FSM) and Synchronous/Reactive (SR) domains.

Domains perform either simulation or code generation. Simulation domains are interpreters that run an executable specification of a system on the local workstation. Code generation domains translate the specification into some language such as C or VHDL and then optionally manage the execution of that generated code. In 0.6 and later, code generation domains can be mixed with each other and with simulation domains. Thanks to José Pino for developing hierarchical scheduling to support this capability.

The model of computation is the *semantics* of the network of blocks. It defines what is meant by an interconnection of blocks, and how the interconnection will behave when executed. The domain determines the model of computation, but in the case of code generation domains, it also determines the target language. So for example, the CGC (Code Generation in C), C50 (Code Generation for the Texas Instruments TMS320C50) and the CG56 (Code Generation for the Motorola DSP56000) domains all use the synchronous dataflow model of computation (the same as the SDF domain). The CGC domain also supports features of the Boolean dataflow (BDF) domain, which supports a measure of run-time scheduling in a very efficient way.

Simulation domains can be either timed or untimed. Untimed domains carry no notion of time in their semantic model. Instead of chronology, they deal only with the order of particles or actions. Timed domains have a notion of *simulated time*, where each particle or action is modeled as occurring at some particular point in this simulated time. Particles and actions are processed chronologically. Timed and untimed domains can be mixed. From the perspective of a timed domain, actions in an untimed domain will appear to be instantaneous. Moreover, timed domains can exist at several levels of the hierarchy, or in parallel at a given level of the hierarchy, separated by untimed domains, and their chronologies will be synchronized. That is, the notion of simulated time in Ptolemy is a global notion. When particles and actions are processed chronologically in each timed domain that is present, then they will be processed chronologically globally.

In this chapter, we also introduce the `Target` class. The basic role of this class is in managing design flow. In a simulation domain, the target selects the scheduler to use (there can be several schedulers in a single domain) and starts and stops a simulation. In a code generation domain, the target also selects the scheduler, but then also generates the code, compiles it, and runs it on a suitable platform. Targets can be defined hierarchically; for example, a multiprocessor target may consist of several, possibly heterogeneous execution platforms, each specified itself as a target. In this example, the top level target might handle the partitioning and interprocessor communication, and the lower level targets might handle the code generation, compilation, and execution. Targets play a much bigger role in code generation domains than in simulation domains.

Ptolemy users often prematurely set out to make a new domain. While it is the intent of Ptolemy to support such experimentation, this task should be undertaken with some trepidation. Although any software engineer can create a domain that will work, defining a useful and correct model of computation is a much harder task. It is very easy, for example, to define a non-determinate model of computation. This means that the behavior of an application will depend on implementation details in the scheduler that are not explicitly known to the user. As a consequence, a user make a small, seemingly innocuous change in an application, and unexpectedly get radically different behavior. At Berkeley, many more domains have been built than are currently distributed. Sometimes, domains have been discarded because of unexpected subtleties in the model of computation. In other cases, domains have been built on top of third-party software or hardware that has become obsolete.

A prerequisite for creating any new domain is understanding the existing domains in Ptolemy. Frequently, one of these domains will meet your needs with simpler extensions, like a new target or a family of stars. If, for example, you are unhappy with the performance of a scheduler, it may make more sense to define a new scheduler (and a target to support it) within an existing domain, rather than creating a new domain.

This chapter gives a brief introduction to the simulation and code generation domains released in Ptolemy 0.7. It also highlights the domains that were present in earlier versions of Ptolemy but are no longer released. This chapter ends with an overview of the interfaces to foreign tools, such as simulators, interpreters, and compilers.

4.2 Synchronous dataflow (SDF)

The SDF domain in Ptolemy is the oldest, most mature domain. Much of its basic capability was ported from Gabriel, the predecessor system [Bie90][Lee89], although it has been extended considerably. SDF is a special case of the dataflow model of computation developed by Dennis [Den75]. The specialization of the model of computation is to those dataflow graphs where the flow of control is completely predictable at compile time. It is a good match for synchronous signal processing systems, those with sample rates that are rational multiples of one another.

The SDF domain is suitable for fixed and adaptive digital filtering, in the time or frequency domains. It naturally supports multirate applications, and its rich star library includes polyphase real and complex FIR filters. Applications with examples in the demo library include speech coding, sample-rate conversion, analysis-synthesis filter banks, modems, phase-locked loops, channel simulation, linear prediction, chaos, filter design, Kalman filtering, phased array beamforming, spectral estimation, sound synthesis, image processing, and video coding. The SDF domain has been used for a number of years at Berkeley for instruction in signal processing, at both the graduate and undergraduate level. The exercises that are assigned to the students are included in the SDF chapter.

4.3 Higher-Order Functions (HOF)

A function is *higher-order* if it takes a function as an argument and/or returns a function. A classic example is *mapcar* in Lisp, which takes two arguments, a function and a list. Its behavior is to apply the function to each element of the list and to return a list of the results. The HOF domain implements a similar function, in the form of a star called `Map`, that can apply any other star (or galaxy) to the sequence(s) at its inputs. Many other useful higher-order functions are also provided by this domain.

The HOF domain provides a collection of stars designed to be usable in all other Ptolemy domains. It is intended to be included as a subdomain by all other domains.

4.4 Dynamic dataflow (DDF)

The predictable control flow of SDF allows for efficient scheduling, but limits the range of applications. In particular, data-dependent flow of control is only allowed within the confines of a star. To support broader applications, the DDF domain uses dynamic (run-time) scheduling. For long runs, involving many iterations, this is more expensive than the static scheduling that is possible with SDF. But in exchange for this additional cost, we get a model of computation that is as versatile as that of conventional programming languages. It supports conditionals, data-dependent iteration, and true recursion.

Although the DDF domain is, in principle, a fully general programming environment, it is nonetheless better suited to some applications than others. We have found that signal processing applications with a limited amount of run-time control are a good match. Examples include systems with multiple modes of operation, such as modems (which have start-up sequences and often implement multiple standards), signal coding algorithms (which often offer a range of compression schemes), and asynchronous signal processing applications, such as timing recovery and arbitrary sample-rate conversion. The demos provided with the domain show how to realize conditionals, iteration, and recursion.

The SDF domain is a subdomain of DDF, which means that SDF stars can be used in DDF systems. For greater efficiency on long runs, the two domains can also be mixed using the Ptolemy hierarchy. A galaxy within a DDF system can be SDF, meaning that it will use an SDF scheduler. Conversely, a galaxy within an SDF system can be DDF.

4.5 Boolean dataflow (BDF)

Boolean dataflow was developed by Joe Buck as part of his Ph.D. thesis research [Buc93c]. Like DDF, it supports run-time flow of control. Unlike DDF, it attempts to construct a compile-time schedule. Thus it achieves the efficiency of SDF with the generality of DDF. It currently supports a somewhat more limited range of stars than DDF, and does not support recursion, but the model of computation is, in principle, equally general. Its applications are the same as those of DDF.

The basic mechanism used in BDF is to construct an *annotated schedule,* by which we mean a static schedule where each firing in the schedule is annotated with the Boolean conditions under which it occurs. Thus, any sequence of firings can depend on a sequence of Boolean values computed during the execution. Executing the annotated schedule involves much less overhead than executing a dynamic dataflow schedule.

4.6 Process Network (PN)

The process network domain, created by Thomas M. Parks and documented in his Ph.D. thesis [Par95], implements Kahn process networks, a generalization of dataflow where processes replace actors. It has some of the flavor of the recently removed CP domain, in that it implements concurrent processes, but unlike the CP domain, it is determinate and has no model of time. The PN domain is implemented using POSIX threads. In principle, PN systems can run in parallel on multiprocessor workstations with appropriate OS support for threads.

The SDF, BDF and DDF domains are subdomains of PN, which means that these stars can be used directly in PN systems. When stars from these domains are used in a PN system, each dataflow actor becomes a dataflow process [Lee95]. For greater efficiency, dataflow domains can be mixed with PN using the Ptolemy hierarchy. A galaxy within a PN system can be SDF, BDF, or DDF, using a scheduler appropriate for that domain. The galaxy as a whole becomes a single process in the PN system.

4.7 Synchronous Reactive (SR)

The Synchronous Reactive domain, created by Stephen Edwards and documented in his Ph.D. thesis [Edw97], is a new and very experimental domain. The Synchronous Reactive domain is a statically-scheduled simulation domain in Ptolemy designed for concurrent, control-dominated systems. To allow precise control over timing, it adopts the synchronous model of time, which is logically equivalent to assuming that computation is instantaneous

SR is similar to existing Ptolemy domains, but differs from them in important ways. Like Synchronous Dataflow (SDF), it is statically scheduled and deterministic, but it does not have buffered communication or multi-rate behavior. SR is better for control-dominated systems that need control over when things happen relative to each other; SDF is better for data-dominated systems, especially those with multi-rate behavior.

SR also resembles the Discrete Event (DE) domain. Like DE, its communication channels transmit events, but unlike DE, it is deterministic, statically scheduled, and allows zero-delay feedback loops. DE is better for *modeling* the behavior of systems (i.e., to better understand their behavior), whereas SR is better for *specifying* a system's behavior (i.e., as a way to actually build it).

4.8 Finite State Machine (FSM)

The Finite State Machine domain, created by Bilung Lee, is a new and very experimental domain. The finite state machine (FSM) has been one of the most popular models for describing control-oriented systems, e.g., real-time process controllers. The FSM domain uses Tycho graphical user interface for the specification of FSM blocks. Currently FSM can interoperate with SDF and DE domains.

4.9 Discrete Event (DE)

The DE domain is a relatively mature domain using an event-driven model of computation. In this domain, particles carry time stamps, and represent events that occur at arbitrary points in simulated time. Events are processed in chronological order. Two schedulers are available. The default scheduler is based on the "calendar queue" mechanism developed by Randy Brown and was written by Anindo Banerjea and Ed Knightly. Since this scheduler is relatively new, the older and simpler but less efficient scheduler is also provided.

DE schedulers maintain an event queue, which is a list of events sorted chronologically by time stamp. The scheduler selects the next event on the list, and determines which star should be fired to process the event. The difference between the efficient calendar queue scheduler and the naive simple scheduler is in the efficiency with which this queue is updated and accessed. Considerable effort was put into consistent and predictable handling of simultaneous events.

The DE domain is suitable for high-level modeling of communications networks, queueing systems, hardware systems, and transportation networks. The demos included with the domain include a variety of queueing systems, shared resource management, communication network protocols, packet-switched networks, wireless networks, and multimedia systems. The latter class of applications take advantage of the ability that Ptolemy has to mix domains by modeling speech and video encoding algorithms using the SDF domain and a packet switched network using the DE domain. There are also some more specialized uses of the DE domain, such as modeling shot noise and synchronizing a simulation to a real-time clock.

4.10 Multidimensional Synchronous Dataflow (MDSDF)

The MDSDF domain was developed by Mike Chen and is still very experimental. This domain is an extension of the Synchronous Dataflow model to multidimensional streams and is based on the work of Edward Lee [Lee93b]. MDSDF provides the ability to express a greater variety of dataflow schedules in a graphically compact way. It also allows nested resettable loops and delays. Additionally, MDSDF has the potential for revealing data parallelism in algorithms. The current implementation of the MDSDF domain only allows two dimensional streams, although we hope that many of the ideas used in the development of the domain can be generalized to higher dimensions.

For a full discussion of the MDSDF domain in Ptolemy, see [Che94].

4.11 Code generation (CG)

The CG domain is the base from which all other code generation domains (such as CGC and CG56) are derived. This domain supports a general dataflow model equivalent to the BDF and SDF models. The stars in this domain do little more than generate comments when fired, but they can serve to demonstrate and test the features of scheduling algorithms. In this domain, you can build test systems, view the generated code (comments) for multiple processors, and display a Gantt chart for parallel schedules. In derived domains, real code is generated, compiled, downloaded and executed, all under control of the selected target. In Ptolemy 0.7, one serious weakness of the code generation domains is that they only support scalar data types (complex, floating-point, integer, and fixed-point) on the input and output ports.

4.12 Code generation in C (CGC)

The CGC domain uses Boolean-controlled dataflow semantics, and has `C` as its target language. We have made every effort to name stars and their parameters consistently so that it is easy to move from one domain to another. With a little effort, one could create CGC versions of all SDF stars. If this were accomplished, then retargeting from one domain to another would be a simple matter of changing domains and targets and running the system again.

The generated `C` code is statically scheduled, and the memory used to buffer data between stars is statically allocated. Moreover, for many of the stars, the code that is generated depends on the values of the parameters. One way to think of this is that the parameters of the star are evaluated at code generation time, so no run-time overhead is incurred from the added flexibility of parameterizing the star.

There are several targets to choose from in the CGC domain. The `bdf-CGC` target supports the boolean-controlled dataflow model of computation. It must be used whenever stars with BDF semantics are present in a program graph. The `default-CGC` target supports the SDF model of computation, so it can be used when the program graph contains only stars with SDF semantics. The `TclTk_Target` target also supports SDF, and must be used whenever Tcl/Tk stars are present in the program graph. The `unixMulti_C` target supports SDF and partitions the program graph for execution on multiple workstations on a network.

4.13 Code generation for the Motorola DSP56000 (CG56)

This domain synthesizes assembly code for the Motorola DSP56000 family. The code generation techniques that are used are described in [Pin93]. They are derived from techniques used in Gabriel [Bie90]. We have used this domain to generate real-time implementations of various modem standards, touchtone generators, and touchtone decoders [Eva96], on an Ariel S-56X 560001 board.

4.14 Code generation in VHDL (VHDL, VHDLB)

This pair of domains is for generating code in VHDL (VHSIC Hardware Description Language). The VHDL domain supports functional models using the SDF model of computation, while VHDLB supports behavioral models using the native VHDL discrete event model of computation. Since the VHDL domain is based on the SDF model, it is independent of any notion of time. The VHDLB domain supports time delays and time-dependent behavior of blocks. The VHDL domain is intended for modeling systems at the functional block level, as in DSP functions for filtering and transforms, or in digital logic functions, independent of implementation issues. The VHDLB domain is intended for modeling the behavior of components and their interactions in system designs at all levels of abstraction.

Within the VHDL domain there are a number of different `Targets` to choose from. The default target, `default-VHDL`, generates sequential VHDL code in a single process within a single entity, following the execution order from the SDF scheduler. This code is suitable for efficient simulation, since it does not generate events on signals. The `SimVSS-VHDL` target is derived from `default-VHDL` and it provides facilities for simulation using the Synopsys VSS VHDL simulator. Communication actors and facilities in the `SimVSS-VHDL` target support code synthesis and co-simulation of heterogeneous CG systems under the `CompileCGSubsystems` target developed by José Luis Pino. There is also a `SimMT-VHDL` target for use with the Model Technology VHDL simulator. The `struct-VHDL` target generates VHDL code where individual actor firings are encapsulated in separate entities connected by VHDL signals. This target generates code which is intended for circuit synthesis. The `Synth-VHDL` target, derived from `struct-VHDL`, provides facilities for synthesizing circuit representations from the structural code using the Synopsys Design Analyzer toolset. Because the VHDL domain uses SDF semantics, it supports retargeting from other domains with SDF semantics (SDF, CGC, etc.) provided that the stars in the original graph are available in the VHDL domain. As this experimental domain evolves, more options for VHDL code generation from dataflow graphs will be provided. These options will include varying degrees of user control and automation depending on the target and the optimization goals of the code generation, particularly in VHDL circuit synthesis.

Unlike the VHDL domain, the older and less-developed VHDLB domain is much simpler in its operation. When a universe in the VHDLB domain is run, the graph is traversed and a codefile is generated in a pop-up window and in a subdirectory which reflects the topology and hierarchy of the graph. The generated VHDL code will reference VHDL entities which are expected to be included in other files. There is a VHDL codefile in the `$PTOLEMY/src/domains/vhdlb/lib` directory for each VHDL star in the main star palettes of the `$PTOLEMY/src/domains/vhdlb/icons` directory. Adding a new star is a matter of writing VHDL code for the entity and adding a star file in the stars subdirectory of the VHDLB domain which reflects the inputs, outputs, and parameters of that star. The existing stars should serve as examples for how new stars can be written.

Table 4-1 below summarizes the various domains

Domain	Description
Synchronous Data Flow (SDF)	• Oldest and most mature domain; it is a sub-domain of DDF, BDF, and PN. • Special case of data flow model of computation developed by Dennis. • Flow is completely predictable at compile time thus allows for efficient scheduling. • Allows for static scheduling. • Good match for synchronous signal processing systems with sample rates that are rational multiples of one another. • Supports multi-rate applications and has a rich star library. • Range of applications is limited.
Dynamic Data Flow (DDF)	• Versatile model of computation as it supports conditionals, data-dependent iteration, and true recursion. • More general than SDF. • Uses dynamic (run-time) scheduling which is more expensive than static scheduling. • Good match for signal processing applications with a limited amount of run-time control.
Boolean Data Flow (BDF)	• Relatively new domain which supports run-time flow of control. • Attempts to construct a compile-time schedule to try and achieve efficiency of SDF with generality of DDF. • More limited than DDF. • Constructs an annotated schedule: execution of a task is annotated with a boolean condition.
Integer and State Controlled Data Flow (STDF)	• Very new to Ptolemy and still experimental. • Realizes data flow control by integer control data and port statuses. It is an extension to BDF. • Scheduling is static and conditional like BDF. • It has user-defined evaluation functions.
Discrete Event (DE)	• Relatively mature domain which uses an event-driven model of computation. • Particles carry time-stamps which represent events that occur at arbitrary points in simulated time. • Events are processed in chronological order.
Finite State Machine (FSM)	• Very new to Ptolemy and still experimental. • Good match for control-oriented systems like real-time process controllers. • Uses a directed node-and-arc graph called a state transition diagram (STD) to describe the FSM.

Domain	Description
Higher Order Functions (HOF)	• Implements behavior of functions that may take a function as an argument and return a function. • HOF collection of stars may be used in all other domains. • Intended to be included only as a sub-domain by other domains.
Process Network (PN)	• Relatively new domain that implements Kahn process networks which is a generalization of data flow – processes replace actors. • Implements concurrent processes but without a model of time. • Uses POSIX threads. • SDF, BDF, and DDF are sub-domains of PN.
Multidimensional Synchronous Data Flow (MDSDF)	• Relatively new and experimental. • Extends SDF to multidimensional streams. • Provides ability to express a greater variety of dataflow schedules in a graphically compact way. • Currently only implements a two-dimensional stream.
Synchronous/Reactive (SR)	• Very new to Ptolemy and still experimental. • Implements model of computation based on model of time used in Esterel. • Good match for specifying discrete reactive controllers.
Code Generation (CG)	• Base domain from which all code generation domains are derived. • Supports a dataflow model that is equivalent to BDF and SDF semantics. • This domain only generates comments, allows viewing of the generated comments, and displays a Gantt Chart for parallel schedules. • Can only support scalar data types on the input and output ports. • All derived domains obey SDF semantics. • Useful for testing and debugging schedulers. • Targets include bdf-CGC which supports BDF, default-CGC which supports SDF semantics, TclTk_Target which supports SDF and must be used when Tcl/Tk stars are present, and unixMulti_C which supports SDF semantics and partitions the graph for multiple workstations on a network.
Code Generation in C (CGC)	• Uses data flow semantics and generates C code. • Generated C code is statically scheduled and memory used to buffer data between stars is statically allocated.
Code Generation for the Motorola DSP 56000 (CG56)	• Synthesizes assembly code for the Motorola DSP56000 family.

Domain	Description
Code Generation in VHDL (VHDL, VHDLB)	• Relatively new and experimental • Generates VHDL code. • VHDL domain supports SDF semantics whereas VHDLB supports behavioral models using native VHDL discrete event model of computation. • Many targets to choose from. • VHDL domain is good for modeling systems at functional block level whereas VHDLB is good for modeling behavior of components and their interactions at all levels of abstraction.

TABLE 4-1: Summary of the various Ptolemy domains.

The table below summarizes the various schedulers

Scheduler Name	Features
Default SDF Scheduler	• Performed at compile time. • Many possible schedules but schedule is chosen based on a heuristic that minimizes resource costs and amount of buffering required. • No looping employed so if there are large sample rate changes, size of generated code is large.
Joe's Scheduler	• Performed at compile time. • Sample rates are merged wherever deadlock does not occur. • Loops introduced to match the sample rates. • Results in hierarchical clustering. • Heuristic solution so some looping possibilities are undetected.
SJS (Shuvra-Joe-Soonhoi) Scheduler	• Performed at compile time. • Uses Joe's Scheduler at front end and then uses an algorithm on the remaining graph to find the maximum amount of looping available.
Acyclic Loop Scheduler	• Performed at compile time. • Constructs a single appearance schedule that minimizes amount of buffering required. • Only intended for acyclic dataflow graphs.

TABLE 4-2: Summary of Uniprocessor schedulers

Table 4-3 below summarizes the multiprocessor schedulers.

Scheduler Name	Features
Hu’s Level-based List Scheduler	• Performed at compile time. • Most widely used. • Tasks assigned priorities and placed in a list in order of decreasing priority. • Ignores communication costs when assigning functional blocks to processors.
Sih’s Dynamic Level Scheduler	• Performed at compile-time. • Assumes that communication and computation can be overlapped. • Accounts for interprocessor communication overheads and interconnection topology.
Sih’s Declustering Scheduler	• Performed at compile-time. • Addresses trade-off between exploiting parallelism and interprocessor communication overheads. • Analyzes a schedule and finds the most promising placements of APEG nodes. • Not single pass but takes an iterative approach.
Pino’s Hierarchical Scheduler	• Performed at compile time. • Partially expands the APEG. • Can use any of the above parallel schedulers as a top-level scheduler. • Supports user-specified clustering. • Realizes multiple orders of magnitude speedup in scheduling time and reduction in memory usage.

TABLE 4-3: Summary of multiprocessor schedulers.

4.15 Domains that have been removed

This section highlights the experimental domains that were removed from the Ptolemy distribution. If any users are interested in resurrecting these domains, then please send e-mail to `ptolemy@ptolemy.eecs.berkeley.edu`.

4.15.1 Circuit simulation (Thor)

Like DE, the Thor domain was event-driven. However, it was specialized to register-transfer level simulation of digital circuits. It was based on the Thor simulator developed at Stanford, which in turn was based on a simulation kernel developed at the University of Colorado. The domain was written by Suengjun Lee. Its capabilities were similar to a variety of commercial circuit simulators. The Thor domain was based on very old circuit simulation technology. Contemporary equivalents include VHDL and Verilog simulators. The VHDL domains thus replace Thor, at least in part, although currently the star library is not as rich.

4.15.2 Communicating processes (CP)

The CP domain, developed by Suengjun Lee and based on thread classes developed by Thomas M. Parks, modeled multiple simultaneous processes with independent flow of control. It was based on the Sun lightweight process library. Because of this dependence on proprietary code, it was only available on Sun workstations. CP was a timed domain. Processes communicated by exchanging particles with time stamps. The particles are processed in chronological order, in that the process with the oldest time stamps at its inputs is activated. From the perspective of the star writer, the star is always running, presumably in an infinite loop, responding to input events and producing output events. Because of this model, the domain was well suited to high-level modeling of distributed systems. Each component in the system would be represented as a program that appears to run as an autonomous agent, concurrently with other components of the system.

The CP domain is probably the most useful of the domains that have been removed. The problem is that it is based on the Sun lightweight process library, which Sun Microsystems is no longer supporting. The Lightweight Processes library does not run on recent releases of the Solaris operating systems. We took a stab at porting the domain to use Posix threads, the modern replacement, but this task overwhelmed the resources we had available. We would be very interested in volunteers interested in pursuing this.

4.15.3 Message queueing (MQ)

The MQ domain was based on an object-oriented approach for software development developed by E. C. Arnold and D. W. Brown at AT&T Bell Laboratories. The run-time environment viewed components as addressable objects capable of receiving messages, sending messages, and maintaining an individual state managed solely by the component's methods. Each message contained sufficient information for its destination object to perform appropriate updating of internal data structures and to produce other messages.

By applying this model in Ptolemy, stars would pass message particles to one another. Connections between pairs of stars are bidirectional so that client-server relationships can be established over these links. From the name of this domain, it can be understood that messages sent from a particular star to another were always processed in sequence. However, the execution order of stars in the system is arbitrary, and a star, when fired, steps through its portholes in an arbitrary fashion as well, processing the first incoming message arriving at each port, should one exist.

The MQ domain was well-suited to the development of call-processing software. Its use in the modeling of system control was illustrated in a sophisticated cell-relay network simulation. This large-scale, heterogeneous demo used the SDF domain to specify space-division packet switching fabrics, the DE domain to model "timed" network subsystems, and the MQ domain to describe a centralized network controller.

Although it introduced a number of interesting features, this domain did not find wide usage, so it has been removed.

4.15.4 Code generation for the Sproc multiprocessor DSP (Sproc)

The Sproc multiprocessor DSP used to be made by Star Semiconductor [Mur93], however, neither the processor nor the company now exist, so this domain has been removed.

4.15.5 Code generation for the Motorola DSP96000 (CG96)

This domain is similar to CG56, except that the synthesized language is assembly code for the Motorola DSP96000 (floating-point) family. This processor is no longer being developed or improved by Motorola, so we have removed this domain.

4.15.6 Code generation in Silage (Silage)

This was a code generation domain for the Silage language. Silage is an applicative functional textual language that was developed to describe fixed-point digital signal processing (DSP) algorithms, which are well-suited for hardware implementation. Silage descriptions serve as the input specification for some high-level synthesis tools (e.g. Hyper from U.C. Berkeley, Cathedral from IMEC, and the DSP Station from Mentor Graphics). Asawaree Kalavade used the Ptolemy interface to Hyper to estimate costs of hardware implementations in hardware/software codesign experiments [Kal93,Kal94,Kal96]. Berkeley, however, is moving away from Silage, and there appears to be little future for it, so we have removed this domain.

4.15.7 Functional Code Generation in VHDL (VHDLF)

The VHDLF domain was originally intended to contrast with the VHDLB domain. It supported structural code generation using VHDL blocks with no execution delay or timing behavior, just functionality. The semantics for the VHDLF domain were not strictly defined, and the scheduling depended on how the underlying VHDL code blocks associated with each VHDLF star were written. The VHDLF domain has been replaced by the VHDL domain. The VHDL domain is not meant to be used in the same way as the VHDLF domain, however. The VHDL domain is for generating code from functional block diagrams with SDF semantics.

4.16 Interfaces to Foreign Tools

The Ptolemy design environment is a collection of dozens of collaborating tools with interfaces to dozens of others, as shown in Figure 4-4. Within Ptolemy there are many different domains, schedulers, and targets, each of which is a tool in its own right. Those tools are derived from a common framework provided by the Ptolemy kernel. Other tools, such as an expression evaluator for parameter expressions and filter design programs, are also embedded.

Not every Ptolemy interface is listed in Figure 4-4. We have written several targets and domains that we have not released to the public. For example, we are developing a CGC target for the UltraSparc Visual Instruction Set. We have developed but never released code generation domains for the AT&T DSP 3 multiprocessor system and the Philips Video Signal Processor system [Shi94]. We have eliminated several domains, as listed in Section 4.15, such as a code generation for the Sproc multiprocessor DSP system [Mur93].

This rest of this section focuses on Ptolemy interfaces to foreign tools that are not included in the Ptolemy release. These foreign tools are standalone programs, such as compilers, assemblers, interpreters, and simulators.

4.16.1 Specification and Layout

Defining systems, subsystems, blocks, and connections can be expressed graphically using pigi (see Chapter 2) and textually using ptcl (see Chapter 3). Graphical descriptions can be converted to textual specifications. The two interfaces can work together. By running pigi with the -console option, one can evaluate ptcl commands in the pigi console. The pigi run-

Released with Ptolemy	*System Design*	*Not Released with Ptolemy*
oct/vem	Specification and Layout	Emacs, vi, xedit
higher-order functions		
ptcl		
Tycho		
expression evaluator	Parameter Calculation	MATLAB
Tcl		Mathematica
simulation domains	Algorithm Prototyping	MATLAB
filter designer		Utah Raster Toolkit
		Esterel
Tcl/Tk, Itcl/Tk	Display and Visualization	MATLAB
pxgraph, xv		soundtool, audiotool
simulation domains	Simulation	sim56000, sim96000
multirate schedulers		Synopsys VSS
run-time schedulers		Model Technology VSIM
wormholes		IPUS, ArrayOL domains
code generation domains	Synthesis	asm56000, Ariel S-56X card
gcc, g++, gmake		commercial C/C++ compilers
parallel schedulers		Synopsys Design Analyzer
hierarchical scheduler		

TABLE 4-4: Ptolemy interfaces to various tools.

control panel allows control of runs by ptcl scripts. Blocks exists that execute ptcl scripts.

Schematic entry is implemented by vem, and schematics are databased in oct. Scalable systems can be specified graphically using higher-order functions. Tycho provides a language-sensitive editor and an evolving framework for future graphical user interfaces.

4.16.2 Parameter Calculation

Parameter calculation maps system parameters into parameters in the subsystems, blocks, and connections. The calculation is controlled by an expression evaluator, which supports calls to ptcl. Because ptcl has interfaces to MATLAB and Mathematica, MATLAB and Mathematica expressions can be embedded in parameter specifications.

4.16.3 Algorithm Prototyping and Visualization

A designer can develop domain-specific algorithms in Ptolemy, such as for speech coding. For filter design, one can use MATLAB or Ptolemy's filter design programs. A variety of Unix and X windows utilities are used for display and visualization of data.

4.16.4 Simulation

Ptolemy provides the ability to simulate complex systems. The key is the notion of a wormhole that allows a designer to mix domain-specific algorithms together to cosimulate the functionality or behavior of the system. wormholes & hierarchical scheduling

4.16.5 Synthesis

Ptolemy provides mature abilities to synthesis dataflow systems. From dataflow graphs, Ptolemy can generate C, C++, Motorola 56000 assembly code, and VHDL for uniprocessor and multi-processor systems.

Ptolemy provides the ability to synthesis complex systems. The key is the notion of hierarchical scheduling that allows multiple implementation technologies to cosimulate. Mixing this with wormholes allows simulation domains to participate in cosimulation. This combinations allows a complex system to be simulated at a variety of levels of detail.