between blocks (if any is needed). This mechanism is very simple, and language independent. We have built code generators for a number of languages, as indicated in figure 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 mechanism, the functionality of the blocks is arbitrary and can be defined by the end user.

6. Conclusions

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.

The Ptolemy kernel has been used successfully outside Berkeley for a number of domain designs. A notable example is the work of Berkeley Design Technology, Inc., as part of the Martin Marietta RASSP program, to use the Ptolemy to connect the SPW and Bones tools from the Alta Group at Cadence.

More information about the Ptolemy project, plus access to all of the software and documentation, is available on the World Wide Web via the URL “http://ptolemy.eecs.berkeley.edu”.

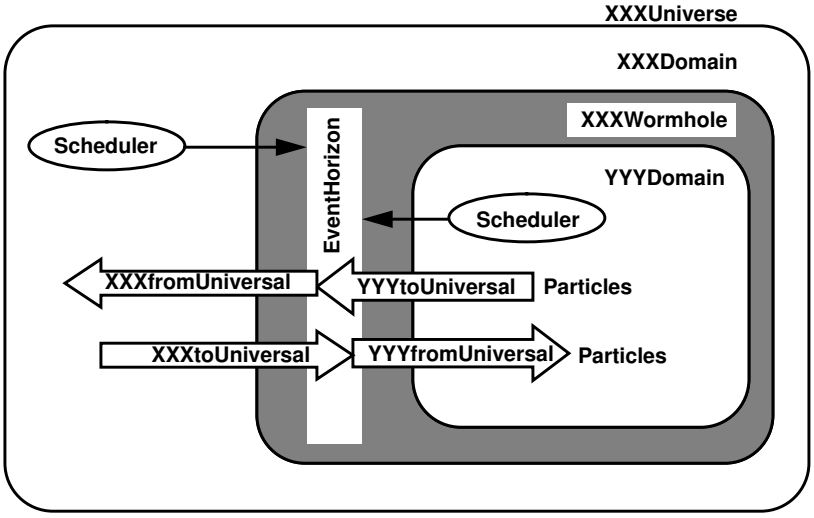

Figure 7. The universal EventHorizon provides an interface between the external and internal domains.

"synchronous dataflow" allows all scheduling to be done at compile time. The Ptolemy kernel supports such specialization by allowing nested domains, as shown in figure 6. For example, the SDF domain (see figure 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.

4. Mixing Models of Computation

Domains in Ptolemy can be mixed. Thus, one system-level design can contain multiple subsystems that are designed or specified using different styles. The kernel support for this is shown in figure 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 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 braod 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.

5. Code Generation

The domains shown in figure 2 are divided into two classes: simulation and code generation. In the simulation domains, a scheduler invokes the run methods of the blocks in a system specification, and those methods perform some function associated with the design. In code generation domains, the scheduler also invokes the run methods of the constituent blocks, but these run methods synthesize code in some language. I.e., they generate code to perform some function, rather than performing the function directly. The Target is responsible then for generating the connecting code

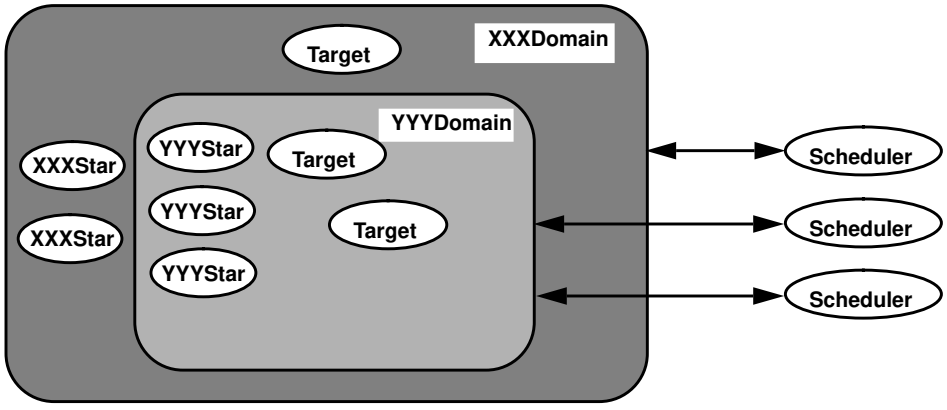

Figure 6. A Domain (XXX) consists of a set of Stars, Targets and Schedulers that support a particular model of computation. A sub-Domain (YYY) may support a more specialized model of computation.

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 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 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

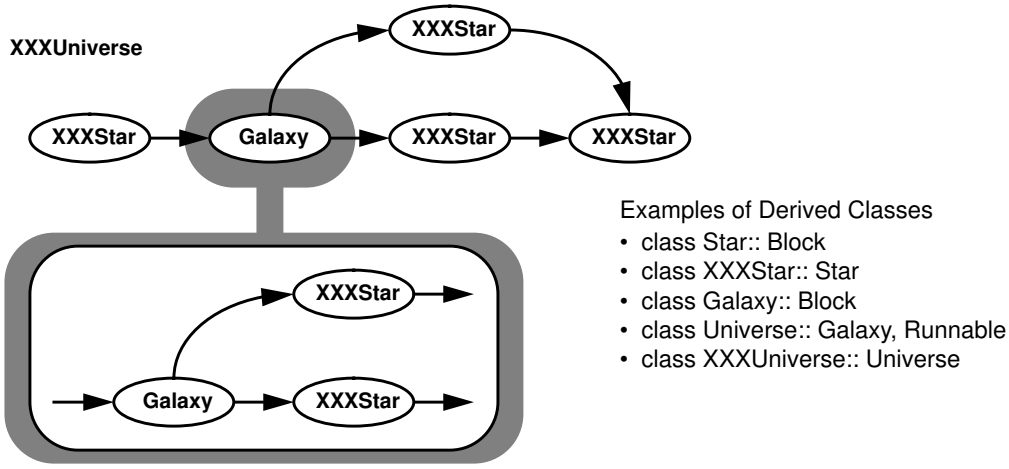

Figure 4. A complete Ptolemy application (a Universe) consists of a network of Blocks. Blocks may be Stars (atomic) or Galaxies (composite). The "XXX" prefix symbolizes a particular domain (or model of computation).

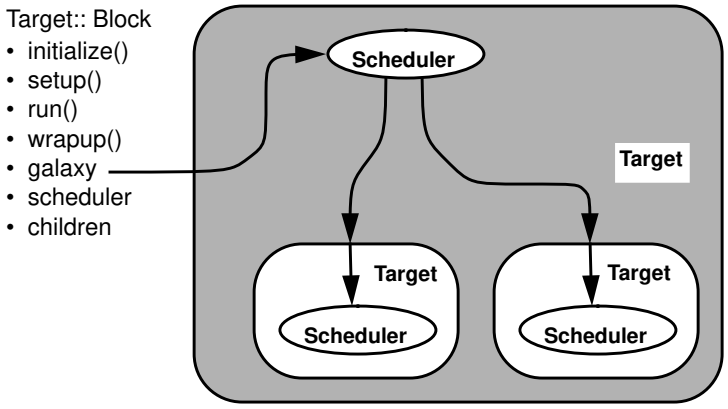

Figure 5. A Target, derived from Block, manages a simulation or synthesis execution. It can invoke it's own Scheduler on a Galaxy, which can in turn invoke Schedulers in sub-Targets.

The Ptolemy kernel provides the most extensive support for domains where a design is represented as a network of blocks, as shown in figure 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 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 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."

3. 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++ classes derived from kernel base classes to support this domain. These classes might be called "XXXStar," "XXXUniverse," etc., as shown in figure 4.

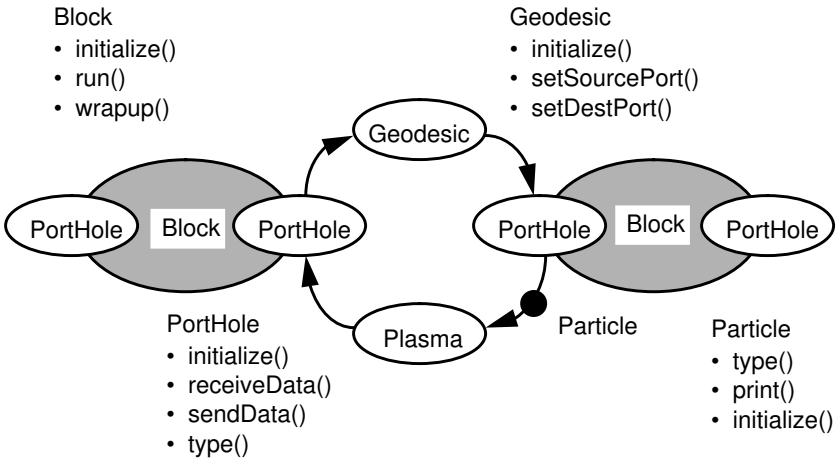

Figure 3. Block objects in Ptolemy can send and receive data encapsulated in Particles through Portholes. Buffering and transport is handled by the Geodesic and garbage collection by the Plasma. Some methods are shown.

tains the user interface (VEM) and the design database (OCT), and the other contains the Ptolemy kernel. An alternative is to run Ptolemy without the graphical user interface, as a single process, as shown in figure 1(b). In this case, a textual interpreter called "ptcl" (Ptolemy Tcl) is used. It is possible to design other user interfaces for the system, and we are experimenting with one based on Tk.

The executables "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 2. Many different styles of design are represented by these domains. More are constantly being developed both at Berkeley and elsewhere, to experiment with or support alternative styles.

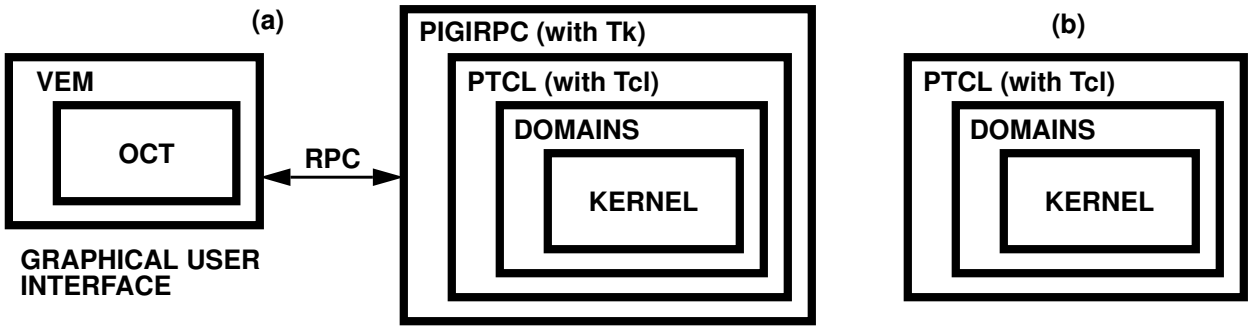

Figure 1. The overall organization of Ptolemy version 0.5.1, showing two possible execution styles. This report concentrates on the *kernel* and its relationship to the *domains*.

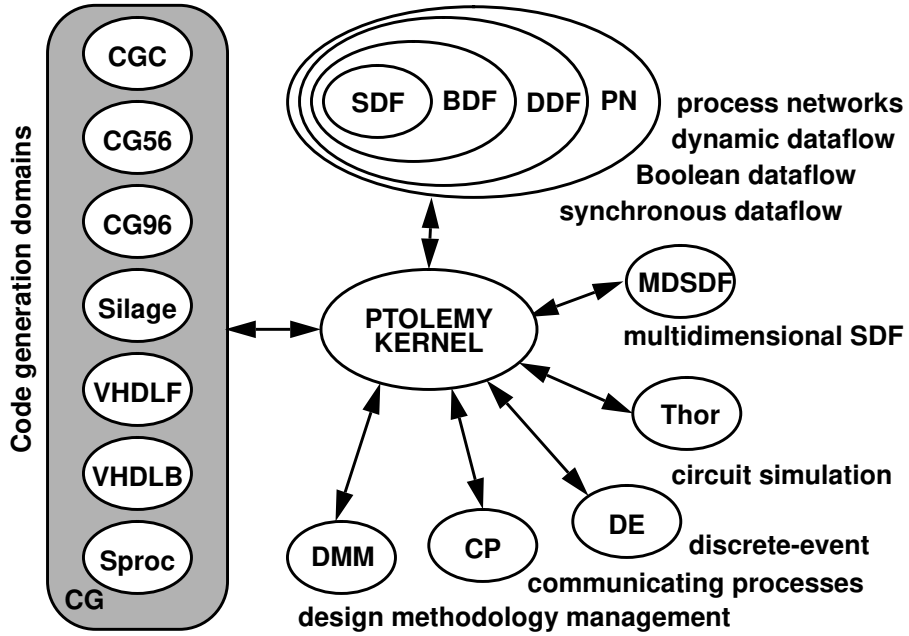

Figure 2. The most recent view of the set of domains developed at Berkeley. This article discusses only CG, which underlies all of code generation.

The Ptolemy Kernel — Supporting Heterogeneous Design

DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCES
UNIVERSITY OF CALIFORNIA AT BERKELEY

by The Ptolemy Team[1]

Proposed article for the RASSP Digest Newsletter

1. Introduction

A technology base team at the University of California at Berkeley is developing a software environment called *Ptolemy* that supports heterogeneous design. An early contribution of this effort has been the design of 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 any (or all) of three 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.
- Compilation — The specification is analyzed and translated into optimized code in any target language.

At Berkeley, we have built a variety of domains that operate in the first two modes only, although code generation domains often have elements of optimization from the third.

The use of an object-oriented software technology permits each of these domains to interact with one another without knowledge of each others' features or semantics. Thus, using a variety of domains, a team of designers can model each subsystem of a complex, heterogeneous system in a manner that is natural and efficient for each subsystem.

2. The Design of the Kernel

The overall organization of the latest release of the Ptolemy system is shown in figure 1. A typical use of Ptolemy involves starting two UNIX™ processes, as shown in figure 1(a): the first con-

1 The current Ptolemy team is: Shuvra Bhattacharyya, Joseph T. Buck, Wan-Teh Chang, Brian L. Evans, Steve X. Gu, Sangjin Hong, Christopher Hylands, Asawaree Kalavade, Alan Kamas, Allen Lao, Bilung Lee, Edward A. Lee, Xiao Mei, David G. Messerschmitt, Praveen K. Murthy, Thomas M. Parks, José Luis Pino, Farhana Shiekh, S. Sriram, Juergen Teich, Warren W. Tsai, Patrick J. Warner, and Michael C. Williamson.