DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator

Department of Electrical Engineering and Computer Science
University of California, Berkeley CA 94720

Final Report 1997-98, Micro Project #97-089
Industrial Sponsors: Cadence, Hewlett-Packard, Hughes, Philips, Rockwell

ABSTRACT

The focus of this project is on design methodology for complex real-time systems where a variety of design methodologies and implementation technologies must be combined. Design methodologies are encapsulated in one or more models of computation, while implementation technologies are implemented as synthesis tools. Applications that use more than one model of computation and/or more than one synthesis tool are said to be heterogeneous. Hardware/software codesign is one example of heterogeneous design. Project results have been disseminated via the *Ptolemy* software, in addition to papers. The overall Ptolemy project is fairly large, with additional support from DARPA, SRC, and a number of other companies, and is strongly collaborative. The MICRO project has focused on real-time signal processing, although the larger project is broader.

1. The Context

The objectives of the Ptolemy Project include many aspects of designing signal processing and communications systems, ranging from designing and simulating algorithms to synthesizing hardware and software, parallelizing algorithms, and prototyping real-time systems. Research ideas developed in the project are implemented and tested in the Ptolemy software environment. The Ptolemy software environment, which serves as our laboratory, is a system-level design framework that allows mixing models of computation and implementation languages.

In designing digital signal processing and communications systems, often the best available design tools are domain specific. The tools must be able to interact. Ptolemy allows the interaction of diverse models of computation by using the object-oriented principles of polymorphism and information hiding. For example, using Ptolemy, a high-level dataflow model of a signal processing system can be connected to a hardware simulator that in turn may be connected to a discrete-event model of a communication network.

A part of the Ptolemy project concerns programming methodologies commonly called “graphical dataflow programming” that are used in industry for signal processing and experimentally for other applications. By “graphical” we mean simply that the program is explicitly specified by a directed graph where the nodes represent computations and the arcs represent streams of data. The graphs are typically hierarchical, in that a node in a graph may represent another directed graph. In Ptolemy the nodes in the graph are subprograms specified in C++.

It is common in the signal processing community to use a visual syntax to specify such graphs, in which case the model is often called “visual dataflow programming.” But it is by no means essential to use a visual syntax.

Hierarchy in graphical program structure can be viewed as an alternative to the more usual abstraction of subprograms via procedures, functions, or objects. It is better suited than any of these to a visual syntax, and also better suited to signal processing.

Some other examples of graphical dataflow programming environments intended for signal processing (including image processing) are HP-Ptolemy, from the EE-Soft division of Hewlett-Packard, Khoros, from the University of New Mexico (now distributed by Khoral Research, Inc.), the signal processing worksystem (SPW), from Cadence, COSSAP, from Synopsys (formerly Cadis), and Simulink, from The MathWorks. These software environments all claim variants of dataflow semantics.

Most graphical signal processing environments do not define a language in a strict sense. In fact, some designers of such environments advocate minimal semantics, arguing that the graphical organization by itself is sufficient to be useful. The semantics of a program in such environments is determined by the contents of the graph nodes, either subgraphs or subprograms. Subprograms are usually specified in a conventional programming language such as C. Most such environments, however, including HP-Ptolemy, Khoros, SPW, Simulink, and COSSAP, take a middle ground, permitting the nodes in a graph or subgraph to contain arbitrary subprograms, but defining precise semantics for the interaction between nodes. We call the language used to define the subprograms in nodes the *host language*. We call the language defining the interaction between nodes the *coordination language*.

Many possibilities have been explored for precise semantics of coordination languages. Many of these limit expressiveness in exchange for considerable advantages such as compile-time predictability. In Ptolemy, a *domain* defines the semantics of a coordination language, but domains are modular objects that can be mixed and matched at will. Thus we gain flexibility without the sloppiness of unspecified semantics in the coordination language.

Graphical programs can be either interpreted or compiled. It is common in signal processing environments to provide both options. The output of compilation can be a standard procedural language, such as C, assembly code for programmable DSP processors, or even specifications of silicon implementations. We have put considerable effort into optimized compilation in the past, although current work is focussing on modeling rather than compilation.

2. Results of Micro Support

2.1. Ptolemy II

We have constructed an entirely new generation of design software that we are calling Ptolemy II. It is written in Java, is fully network-integrated, is capable of operating within the worldwide web and enterprise software architectures, and is multithreaded.

Ptolemy II is a complete, from the ground up, redesign of the Ptolemy 0.x software environment, now called Ptolemy Classic, which supports heterogeneous modeling and design of concurrent systems. It offers a unified infrastructure for implementations of a number of models of computation. The overall architecture consists of a set of packages that provide generic support for all models of computation and a set of packages that provide more specialized support for particular models of computation. Examples of the former include packages that contain math libraries, graph algorithms, an interpreted expression language, signal plotters, and interfaces to media capabilities such as audio. Examples of the latter include packages that support clustered graph representations of models, packages that support executable models, and *domains*, which are packages that implement a particular model of computation.

The predecessor to Ptolemy II, Ptolemy Classic, still has many active users and developers, and may continue to evolve for some time. Ptolemy II has a somewhat different emphasis, and through its use of Java, concurrency, and integration with the network, is aggressively experimental. Some of the major capabilities in Ptolemy II that we believe to be new technology in modeling and design environments include:

- *Higher level concurrent design in Java™*. Java support for concurrent design is very low level, based on threads and monitors. Maintaining safety and liveness can be quite difficult. Ptolemy II includes a number of domains that support design of concurrent systems at a much higher level of abstraction. These include, at varying levels of maturity, process networks, communicating sequential processes (rendezvous based), dataflow, synchronous/reactive modeling, continuous-time modeling, and hierarchical concurrent finite-state machines.
- *Better modularization through the use of packages*. Ptolemy II is divided into packages that can be used independently and distributed on the net, or drawn on demand from a server. This breaks with tradition in design software, where tools are usually embedded in huge integrated systems with interdependent parts.
- *Complete separation of the abstract syntax from the semantics*. Ptolemy designs are structured as clustered graphs. Ptolemy II defines a clean and thorough abstract syntax for such clustered graphs, and separates into distinct packages the infrastructure supporting such graphs from mechanisms that attach semantics (such as dataflow, analog circuits, finite-state machines, etc.) to the graphs.
- *Improved heterogeneity.* Previous realizations of Ptolemy provided a wormhole mechanism for hierarchically coupling heterogeneous models of computation. This mechanism is improved in Ptolemy II through the use of opaque composite actors, which provide better support for models of computation that are very different from dataflow, the best supported model in prior versions of Ptolemy software. These include hierarchical concurrent finite-state machines and continuous-time modeling techniques.
- *Thread-safe concurrent execution*. Ptolemy models are typically concurrent, but in the past, support for concurrent execution of a Ptolemy model has been primitive. Ptolemy II supports concurrency throughout, allowing for instance for a model to mutate (modify its clustered graph structure) while the user interface simultaneously modifies the structure in different ways. Consistency is maintained through the use of monitors and read/write semaphores built upon the lower level synchronization primitives of Java.
- *A software architecture based on object modeling*. Since the first Ptolemy implementation, software engineering has seen the emergence of sophisticated object modeling and design pattern concepts. We have applied these concepts to the design of Ptolemy II, and they have resulted in a more consistent, cleaner, and more robust design. We have also applied a simplified software engineering process that includes systematic design and code reviews.
- *A truly polymorphic type system*. Earlier implementations of Ptolemy supported rudimentary polymorphism through the "anytype" particle. Even with such limited polymorphism, type resolution proved challenging, and the implementation is ad-hoc and fragile. Ptolemy II has a more modern type system based on a partial order of types and monotonic type refinement functions associated with functional blocks. Type resolution consists of finding a fixed point, using algorithms inspired by the type system in ML.
- *Domain-polymorphic actors*. In earlier implementations of Ptolemy, actor libraries were separated by domain. Through the notion of subdomains, actors could operate in more than one domain. In Ptolemy II, this idea is taken much further. Actors with intrinsically polymorphic functionality can be written to operate in a much larger set of domains. The mechanism they use to communicate with other actors depends on the domain in which they are used. This is managed through a concept that we call a **process level type system**.

2.2. Status

We have released a highly preliminary version of Ptolemy II that includes the following packages:

actor This package supports executable entities that receive and send data through ports. It includes both untyped and typed actors. For typed actors, it implements a sophisticated type system that supports polymorphism. It includes the base class Director for domain-specific classes that control the execution of a model.

actor.util This subpackage contains utilities that support directors in various domains. Specifically, it contains a simple FIFO Queue and a sophisticated pri-

ority queue called a calendar queue.

data This package provides classes that encapsulate and manipulate data that is transported between actors in Ptolemy models.

data.expr This class supports an extensible expression language and an interpreter for that language. Parameters can have values specified by expressions. These expressions may refer to other parameters. Dependencies between parameters are handled transparently, as in a spreadsheet, where updating the value of one will result in the update of all those that depend on it.

graph This package provides algorithms for manipulating and analyzing mathematical graphs. Mathematical graphs are simpler than Ptolemy II clustered graphs in that there is no hierarchy, and arcs link exactly two nodes. This package is expected to supply a growing library of algorithms.

kernel This package provides the software architecture for the key abstract syntax, clustered graphs. The classes in this package support entities with ports, and relations that connect the ports. Clustering is where a collection of entities is encapsulated in a single composite entity, and a subset of the ports of the inside entities are exposed as ports of the cluster entity.

kernel.util This subpackage of the kernel package provides a collection of utility classes that do not depend on the kernel package. It is separated into a subpackage so that these utility classes can be used without the kernel. The utilities include a collection of exceptions, classes supporting named objects with attributes, lists of named objects, a specialized cross-reference list class, and a thread class that helps Ptolemy keep track of executing threads.

math This package encapsulates mathematical functions and methods for operating on matrices and vectors. It also includes a complex number class and a class supporting fractions.

plot This package provides two-dimensional signal plotting widgets.

3. Future Capabilities

Capabilities that we anticipate making available in the future include:

- *Extensible XML-based file formats*. XML is an emerging standard for representation of information that focuses on the logical relationships between pieces of information. Human-readable representations are generated with the help of style sheets. Ptolemy II will use XML as its primary format for persistent design data.
- *Interoperability through software components*. Ptolemy II will use distributed software component technology such as CORBA, JINI, or COM, in a number of ways. Components (actors) in a Ptolemy II model will be implementable on a remote server. Also, components may be parameterized where parameter values are supplied by a server (this mechanism supports *reduced-order modeling*, where the model is provided by the server). Ptolemy II models will be exported via a server. And finally, Ptolemy II will support migrating software components.
- *Embedded software synthesis.* Pertinent Ptolemy II domains will be tuned to run on a Java virtual machine on an embedded CPU. Hardware, firmware, and configurable hardware components will expose abstractions to this Java software that obey the model of computation of the pertinent domain. Java's native code interface will be used to define a stub for the embedded hardware components so that they are indistinguishable from any other Java thread within the model of computation. Domains that seem particularly well suited to this approach include PN and CSP.
- *Embedded hardware synthesis*. Earlier versions of Ptolemy had only very weak mechanisms for migrating designs from idealized floating-point simulations through fixed-point simulations to embedded software, FPGA, and hardware designs. Ptolemy II will separate the interface definition of component blocks from their implementation, allowing libraries to be constructed where compatibility across implementation technologies is assured. This work is currently being prototyped in Ptolemy 0.7.1.
- *Integrated verification tools.* Modern verification tools based on model checking could be integrated with Ptolemy II at least to the extent that finite state machine models can be checked. We believe that the separation of control logic from concurrency will greatly facilitate verification, since only much smaller cross-sections of the system behavior will be offered to the verification tools.

3.1. Theory

We have completed a paper that develops the semantics of hierarchical finite-state machines that are composed using various concurrency models, particularly dataflow, discrete-events, and synchronous/reactive modeling. This paper has been accepted for publication in the *IEEE Transactions on CAD*. We have also had a paper accepted to the same journal that defines a mathematical meta model enabling the rigorous comparison of models of computation. Finally, we have written a paper that was invited to *The Annals of Software Engineering*. This paper gives a formal semantics for discrete-event languages, which includes most popular hardware description languages.

3.2. Information Dissemination Policy

The Ptolemy web site, http://ptolemy.eecs.berkeley.edu, is used to distribute all software (including source code) and documentation, together with updated summary sheets, answers to frequently asked questions, and tutorials. We use the most liberal copyright permitted by the University of California, one which has proven effective in promoting technology transfer. A Usenet news group called comp.soft-sys.ptolemy and a mailing list ptolemy-hackers@ptolemy.eecs.berkeley.edu are used to communicate with outside users. Postings to the mailing list are cross-posted to the news group. Postings are archived and searchable from our web site

4. Publications

This project has generated a rather large number of publications during this reporting period. Here are some of the highlights.

4.1. Journal Articles

[1] E. A. Lee, “Modeling Concurrent Real-time Processes Using Discrete Events,” Invited paper to *Annals of Soft-*

ware Engineering, Special Volume on Real-Time Software Engineering, to appear, 1998. Also UCB/ERL Memorandum M98/7, March 4th 1998.
(http://ptolemy.eecs.berkeley.edu/papers/98/realtime/)

[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of Embedded Software from Synchronous Dataflow Specifications," Invited paper, to appear in J. of VLSI Signal Processing, 1998.
(http://ptolemy.eecs.berkeley.edu/papers/98/synthesis/)

[3] E. A. Lee and D. G. Messerschmitt, "Engineering an Education for the Future," *IEEE Computer Magazine*, Vol. 31, No. 1, January, 1998.
(http://ptolemy.eecs.berkeley.edu/papers/98/curriculum/)

4.2. Conference Papers

[4] Bilung Lee and Edward A. Lee, "Interaction of Finite State Machines with Concurrency Models," *Proc. of Thirty Second Annual Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, November 1998.

[5] Rajarshi Gupta, Kiran and Edward A. Lee, "Computationally Efficient Version of the Decision Feedback Equalizer," to appear in ICASSP 99, September 1998.

[6] Eric K. Pauer, Cory S. Myers, Paul D. Fiore, John M. Smith, Christopher M. Crawford, Edward A. Lee, James Lundblad and Christopher Hylands, "Algorithm Analysis and Mapping Environment for Adaptive Computing Systems," Presented at the Second Annual Workshop on High Performance Embedded Computing, MIT Labs, Lexington, MA, September, 1998.

[7] H. J. Reekie and E. A. Lee, "The Tycho Slate: Complex Drawing and Editing in Tcl/Tk," April 27, 1998. Submitted to the Sixth Annual Tcl/Tk Conference, September 14-18, 1998, San Diego, California.
(http://ptolemy.eecs.berkeley.edu/papers/98/slate)

4.3. PhD Theses

[8] M. Williamson "Synthesis of Parallel Hardware Implementations from Synchronous Dataflow Graph Specifications" **Ph.D. thesis**, Memorandum UCB/ERL M98/45, Electronics Research Laboratory, University of California, Berkeley, May, 1998.
(http://ptolemy.eecs.berkeley.edu/papers/98/SDFToParallelVHDL)

4.4. Masters Reports

[9] Neil Smyth, "Communicating Sequential ProcessesDomain in Ptolemy II," MS Report, UCB/ERL Memorandum M98/70, Dept. of EECS, University of California, Berkeley, CA 94720, December 1998.

[10] Jie Liu, "Continuous Time and Mixed-Signal Simulation in Ptolemy II," MS Report, UCB/ERL Memorandum M98/74, Dept. of EECS, University of California, Berkeley, CA 94720, December 1998.

[11] M. Goel, "Process Networks in Ptolemy II" MS Report, ERL Technical Report UCB/ERL No. M98/69, University of California, Berkeley, CA 94720, December 16, 1998.
(http://ptolemy.eecs.berkeley.edu/papers/98/PNinPtolemyII)

4.5. Other Technical Reports

[12] Edward A. Lee, "Overview of the Ptolemy Project", ERL Technical Report UCB/ERL No. M98/71, University of California, Berkeley, CA 94720, November 23, 1998.

[13] J. Davis, R. Galicia, M. Goel, C. Hylands, E.A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay and Y. Xiong, "Heterogeneous Concurrent Modeling and Design in Java," Technical Report UCB/ERL No. M98/72, University of California, Berkeley, CA 94720, November 23, 1998.

[14] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Resynchronization for multiprocessor DSP implementation - part 1: Maximum-throughput resynchronization." Tech. Rep., Digital Signal Processing Laboratory, University of Maryland, College Park, July 1998. Revised from Memorandum UCB/ERL 96/55, Electronics Research Laboratory, University of California at Berkeley, October, 1996.

[15] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Resynchronization for multiprocessor DSP implementation - part 2: Latency-constrained resynchronization."Tech. Rep., Digital Signal Processing Laboratory, University of Maryland, College Park, July 1998. Revised from Memorandum UCB/ERL 96/56, Electronics Research Laboratory, University of California at Berkeley, October, 1996.

[16] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," April 13, 1998 (revised from Memorandum UCB/ERL M97/57, Electronics Research Laboratory, University of California, Berkeley, CA 94720, August 1997).
(http://ptolemy.eecs.berkeley.edu/papers/98/starcharts)

[17] B. Lee and E. A. Lee, "Hierarchical Concurrent Finite State Machines in Ptolemy," *Proc. of International Conference on Application of Concurrency to System Design*, p. 34-40, Fukushima, Japan, March 1998.

[18] Jie Liu, Marcello Lajolo, and Alberto Sangiovanni-Vincentelli, "Software Timing Analysis Using SW/HW Cosimulation and Instruction Set Simulator," *Proc. of the Sixth International Workshop on Hardware/Software Codesign*, p. 65-70, Seattle, Washington, March 1998.

[19] E. A. Lee and A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation,", March 12, 1998. (Revised from ERL Memorandum UCB/ERL M97/11, University of California, Berkeley, CA 94720, January 30, 1997).
(http://ptolemy.eecs.berkeley.edu/papers/98/framework/)