DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator

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

Final Report 1996-97, Micro Project #96-078
Industrial Sponsors: Alta Group of Cadence Design Systems, 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 Khoros, from the University of New Mexico (now distributed by Khoral Research, Inc.), the signal processing worksystem (SPW), from the Alta Group at Cadence (formerly Comdisco Systems), 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 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.

2. Results of Micro Support

2.1. Java Exploration

At the start of this project, we began a serious investigation of the Java language as a possible basis for our future software development. This evaluation came out strongly in favor of using Java, complementing it with Tcl (for scripting) and Tk (for user interface work). The user interface capability in Java is extremely limited compared to Tk, although the situation is improving. The use of a scripting language in combination with Java has compelling advantages. Rather than using Tcl/Tk in pure form, we are using Itcl, an object-oriented extension.

Initially we had a number of key questions that needed to be addressed. First, since the language is conceived around interpreted byte code, will it be capable of delivering the performance needed by signal processing applications? We made some measurements, comparing interpreted Java code against compiled Java code using a just-in-time Java compiler, and found about a factor of 30 improvement in speed. Although there is still a performance penalty compared to optimized C++ code, we believe that there is no fundamental reason for Java to be slower than C++ code, if compiled into native code. Java compilers and interpreters will improve over time, as there is a huge investment in them in industry. There is also a large investment in minimal Java virtual machines designed for embedded systems.

A second key question is whether the Java thread library provides an adequate platform for constructing concurrent applications. The basic mechanism, wait and notify, is too low level for most applications programming. In particular, direct use of such low-level primitives makes validation of concurrent programs very difficult. For example, there is no direct way of ensuring determinacy and preventing deadlock. Indeed, our experience writing threaded Java code underscores the need for a higher level abstraction above threads.

2.1.1 Java Ptolemy Kernel

We are in the process of designing and implementing a set of Java classes that realize key functionality in the Ptolemy kernel. The design of these classes has been painstaking and careful. We have emphasized systematic software engineering over speed of development. One reason for this is that the group of students, staff, and faculty working on the software are all relatively inexperienced with Java, and most of them are also relatively unfamiliar with the Ptolemy kernel. We have instituted careful code review mechanisms, and have rewritten the most critical code several times. The result so far has been to build a small and very well-designed core software infrastructure, and to build a cohesive team of experts.

A key idea in the design of these classes is to define a small core data structure supporting uninterpreted hierarchical graphs. Such graphs provide an abstract syntax for netlists, state transition diagrams, block diagrams, etc. Although this idea is present in the original Ptolemy kernel, in fact the Ptolemy kernel has much more semantics than we would like. Much of the effort involved in implementing models of computation that are very different from dataflow stems from having to work around certain assumptions in the kernel that, in retrospect, proved to be particular to dataflow.

2.2. Network Integrated Design

We made Tycho internet aware. Tycho is the Itcl side of our software effort. It provides an infrastructure for data management and user interface design. The software architecture supports transparent access to URLs, as if they were local files. This mechanism is portable (it works under NT as well as Solaris) and robust. We prototyped a web-based client-server architecture for Ptolemy.

2.3. Java-Tcl Interaction

In the near term, our software environment will mix Java and Itcl. Over the last year, we have been through several Java/Itcl interfaces. The long term solution appears to be TclBlend, recently released by Sun Microsystems. Currently, TclBlend does not work with Itcl, but it does work with Tcl (the non-object-oriented base language). TclBlend is a Tcl/Java interface that uses C code that calls the Java Native Invocation (JNI) module in JDK1.1 (the currently stable Java development kit). We have found problems, for example, with multiprocessor platforms (where SunScript's own test suites fail).

2.4. Java Signal Plotter

We have released on the net two versions of our first Java module, a versatile signal plotter. See the web page, which contains a number of demonstrations:

http://ptolemy.eecs.berkeley.edu/java/ptplot

The plotter is backwards compatible with pxgraph, the signal plotter that has been used in Ptolemy, but adds the interactive animated plotting feature of TkPlot, which is also used in Ptolemy. Thus, it will replace these two facilities.

2.5. Tycho Information Models (TIM)

We have defined a simple framework to support exchange of information and cooperation between modular tools. The aim is to make it easy to describe a model, to describe relationships between models, and to define new models. A Tycho information model (TIM) is the unit of information read and generated by tools. It is conceptually similar to the file representation of a compound document in systems like OpenDoc -- that is, it contains structured data, but it is not an object or a database. Models have attributes, which contain information about the model and its relation to other models, entities, which are conceptually similar to objects in object models, and associations between entities, which are similar to associations or links in object models. We have redesigned the preferences manager in Tycho using TIM to make it more modular.

2.6. Object Modeling

We have organized a study group that is examining current practice in object modeling in general and the UML language in particular. This study is being applied to critically examine the current and future design of Ptolemy.

2.7. Process Network Domain in Java and Tycho

The Process Network (PN) domain in Ptolemy implements an asynchronous, highly distributable concurrent model of computation that generalizes dataflow. It appears to be a very good candidate for providing a higher-level abstraction in Java (above threads) for concurrent programming. The current Ptolemy implementation uses threads via C++.

We subsequently further developed this work, and built a complete Java infrastructure supporting the PN model of computation. The next step will be to create a Java Ptolemy PN domain, which is likely to be the first Java domain.

2.8. Filter Design

We have constructed the basic skeleton of a software architecture for interactive filter design. The idea is that, like PtPlot, this will form another of the modular pieces of Ptolemy software in Java. In the current design conception, a filter is a model (or subject), extending a Java Observable class. Its pole/zero plot, frequency response plot, impulse response plot, and other dependent data will extend the Java Observer class, implementing a view in a model-view paradigm. There will be a object called Manager that creates the plots (observers) and filter (subject). The Manager also handles the initial setup for the filter from the user. However all the changes of the filter data are handled between filter and plots, using the notify and update functions that are built in to the Java base classes.

2.9. Generalized Hybrid Systems

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. It is argued that all three combinations are useful, and that therefore the concurrency model should be selected independently of the decision to use hierarchical FSMs. In contrast, most formalisms that combine FSMs with concurrency models, such as Statecharts (and its variants) and Hybrid systems, tightly integrate the FSM semantics with the concurrency semantics. We have prototyped an implementation of two of the three combinations described.

One of the several contributions in this paper is the definition of a new dataflow model of computation called Heterochronous Dataflow (HDF) that combines FSMs with dataflow is such a way as to preserve certain formal properties of synchronous dataflow. This is particularly important for embedded system design, where questions such as deadlock and bounded memory must be answered at design time.

2.10. Type Systems

We have conceptualized a formal approach to type systems for system-level design that we believe will solve many of the problems we have encountered in the past, and also will scale up to encompass semantic as well as syntactic issues in heterogeneous systems.

The problem we are addressing is that modular system components expose interfaces of different types, and interconnecting such components requires resolving the type differences. In classical programming languages, these types define the layout of data in memory (a syntactic issue), and to a more limited degree, its semantic interpretation (e.g. a double precision IEEE floating point number versus a long integer). In modern object-oriented systems, type issues become somewhat more complex because of polymorphism, where objects of fundamentally different types expose the same interface. In system-level design, the issue becomes still more complicated because the semantic interpretations get considerably richer. For example, two lists of numbers may be syntactically identical, but one may represent a time-domain signal while the other represents a frequency-domain signal.

In the early 1970s, Dana Scott proposed the use of partial orders for representing and analyzing type systems. We have realized that this is exactly the approach we need. In this approach, an "information order" is used, where a type is "less than" another type if it is less specific. Thus, for example, type "Number" is "less than" type "Double" in Java. The least type in a scalar type system would be the Ptolemy "Anytype" (this is called the "bottom" of the partial order). The partial order can be given a "top" as well, where "top" represents a type conflict, i.e. an unresolvable type. Such an order will (usually) be finite, and therefore the mathematical structure of the type system becomes a lattice.

The type signature of a module (corresponding to a C++ template, for example) will be given by a function that given some guess about the type of the interface ports returns a new guess that is at least as specific. In terms of the partial order, such a function is monotonic. Moreover, any composition of such functions is monotonic. Resolving the type of an interconnection of modules becomes a matter of iteratively applying these monotonic functions until they converge on a resolved type for every signal interfacing two modules. This convergence point is called a "fixed point." The well-known Knaster-Tarsky fixed point theorem states that any monotonic function over a lattice has a least fixed point. "Least" in this case means least specific, thus leaving maximum room for polymorphism.

To practically apply this theory, we can use the scheduler developed by Stephen Edwards for the SR (synchronous/reactive) domain in Ptolemy. That scheduler finds an efficient order in which to evaluate monotonic functions in a finite complete partial order (CPO). A lattice is a CPO, so the result can be used directly, despite the fact that the context for which Edwards developed it was radically different.

This idea for a type system may scale very well up to the process level. One could, for instance, consider as part of the type whether a signal is in the frequency domain or the time domain. Fixed-point data types could also be ordered (more precise is more specific). Moreover, approximate signals could perhaps be ordered by type much like fixed-point signals. The semantics of signals (discrete-event, dataflow, synchronous/reactive) might also be amenable to ordering, allowing inference of interaction semantics between modules in addition to resolution of syntactic types.

We believe that this is a very exciting development, and may prove to be one of the major contributions of this project.

3. Software

Software in the Ptolemy project serves as both a laboratory for experimentation and a mechanism for disseminating results. A new feature of this project is that we expect to be distributing software in the form of smaller packages rather than large monolithic software systems. During this reporting period we com-

pleted two small software releases, Tycho 0.2 and PtPlot 0.1. The first is written in Itcl and the second in Java.

3.1. 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.

3.2. Tycho

Tycho is an object-oriented syntax manager with an underlying heterogeneous technical rationale that was started with DARPA funding prior to the commencement of this particular project. It provides a number of editors and graphical widgets in an extensible, reusable framework. The intent is that visual editors and visualization tools will be fully integrated, although most of this work will be conducted in the second 18 month phase of the project. Documentation for Tycho modules is integrated, using HTML, with an integrated and network capable simplified browser.

Tycho was originally conceived for use with Ptolemy system, but under this project is evolving into a set of modules that can be used independently. Tycho has been used extensively in the development of the Tycho software itself. It is written primarily in Itcl, also called [incr Tcl], developed by Michael McLennan of AT&T. Itcl is an object-oriented extension of Tcl, a "tool command language" written by John Ousterhout of U.C. Berkeley, now under continued development at Sun Microsystems. The window toolkit Tk and its object-oriented extension Itk are also used extensively.

3.3. PtPlot 0.1 and 0.1.1

We released our first small, modular Java package, in part as a way to gain experience with the process. We have learned that "write-once, run anywhere" is largely an unrealized promise of Java, and that Java threads are particularly vulnerable to unpredictable behavior on different platforms. Nonetheless, we have successfully distributed what appears to be a useful package, and judging from the email traffic that it has generated, one that is actually being used.

PtPlot is a set of two dimensional signal plotter components fashioned after the plotting capabilities built into Ptolemy. It is written in Java and is described in detail above. The PtPlot package can be found at:

http://ptolemy.eecs.berkeley.edu/java/ptplot/

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] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proceedings of the IEEE*, Vol. 85, No. 3, March 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/codesign)

[2] W.-T. Chang, S.-H. Ha, and E. A. Lee, "Heterogeneous Simulation — Mixing Discrete-Event Models with Dataflow," invited paper, *Journal on VLSI Signal Processing*, Vol. 13, No. 1, January 1997. (http://ptolemy.eecs.berkeley.edu/papers/96/heterogeneity)

[3] S.S. Bhattacharyya, S. Sriram, and E.A. Lee, "Optimizing Synchronization in Multiprocessor DSP Systems,", *IEEE Tr. on Signal Processing*, Vol. 45, No. 6, June 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/synchronization/)

4.2. Conference Papers

[4] C. Hylands, E. A. Lee, and H. J. Reekie, "The Tycho User Interface System," *5th Annual Tcl/Tk Workshop '97*, Boston, Massachusetts, July, 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/tcltk-97/)

[5] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Software Synthesis for Synchronous Dataflow," *International Conference on Application Specific Systems*, Architectures, and Processors, July, 1997, invited paper. (http://ptolemy.eecs.berkeley.edu/papers/97/softwareSynth)

4.3. Technical Reports

[6] A. Girault, B. Lee, and E. A. Lee, "A Preliminary Study of Hierarchical Finite State Machines with Multiple Concurrency Models," Memorandum UCB/ERL M97/57, Electronics Research Laboratory, University of California, Berkeley, CA 94720, August 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/preliminaryStarcharts)

[7] E. A. Lee and A. Sangiovanni-Vincentelli, "A Denotational Framework for Comparing Models of Computation," ERL Memorandum UCB/ERL M97/11, University of California, Berkeley, CA 94720, January 30, 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/denotational/)

[8] P. K. Murthy and E. A. Lee, "Some cycle-related problems for regular dataflow graphs: complexity and heuristics," UCB/ERL Technical Report M97/76, July 1997.

[9] R. S. Stevens (Naval Research Laboratory), M. Wan, P. Laramie (UCB), T. M. Parks (MIT Lincoln Labs), and E. A. Lee (UCB), "Implementation of Process Networks in Java," UCB/ERL Tech. Report, November 1997.

4.4. PhD Theses

[10] S. A. Edwards, "The Specification and Execution of Heterogeneous Synchronous Reactive Systems," **Ph.D. thesis**, University of California, Berkeley, May 1997. Available as

UCB/ERL M97/31. (http://ptolemy.eecs.berkeley.edu/papers/97/sedwardsThesis/)