DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator

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

Final Report 1998-99, Micro Project #98-090
Industrial Sponsors: Cadence, Hewlett-Packard, Hughes, Philips

ABSTRACT

The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. A software system called Ptolemy II is being constructed in Java. The overall Ptolemy project is fairly large, with additional support from DARPA, GSRC, 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 embedded 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 II the nodes in the graph are subprograms specified in Java.

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 Agilent, the signal processing worksystem (SPW), from Cadence, COSSAP, from Synopsys, 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, 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 are building a second 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 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.

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, at the level of their software architecture. Some of these domains use Java threads as an underlying mechanism, while others offer an alternative to Java threads that is much more efficient and scalable. The Java Language Specification allows for platform dependent threading implementations, which can result in non-determinism. Ptolemy II helps insulate the naive programmer from some of these difficult issues.
- *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.* Ptolemy Classic 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 Ptolemy Classic. 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 Ptolemy Classic was constructed, 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*. Ptolemy Classic 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 Ptolemy Classic, 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*.
- *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 uses XML as its primary format for persistent design data.

2.2. Status

We have released a version of Ptolemy II that includes the following domains:

2.2.1 Communicating Sequential Processes - CSP

In the CSP domain (communicating sequential processes), actors represent concurrently executing processes, implemented as Java threads. These processes communicate by atomic, instantaneous actions called rendezvous (sometimes called synchronous message passing). If two processes are to communicate, and one reaches the point first at which it is ready to communicate, then it stalls until the other process is ready to communicate. “Atomic” means that the two processes are simultaneously involved in the exchange, and that the exchange is initiated and completed in a single uninterruptable step. Rendezvous models are particularly well-matched to applications where resource sharing is a key element, such as client-server database models and multitasking or multiplexing of hardware resources. A key feature of rendezvous-based models is their ability to cleanly model nondeterminate interactions.

2.2.2 Continuous Time - CT

In the CT domain (continuous time), actors represent components that interact via continuous-time signals. Actors typically specify algebraic or differential relations between inputs and outputs. The job of the director in the domain is to find a fixed-point, i.e., a set of continuous-time functions that satisfy all the relations. The CT domain includes an extensible set of differential equation solvers. The domain, therefore, is useful for modeling physical systems with linear or nonlinear algebraic/differential equation descriptions, such as analog circuits and many mechanical systems. Its model of computation is similar to that used in Simulink, Saber, and VHDL-AMS, and is closely related to that in Spice circuit simulators. The CT domain is designed to interoperate with other Ptolemy domains, such as DE, to achieve mixed signal modeling. Physical systems often have simple models that are only valid over a certain regime of operation. Outside that regime, another model may be appropriate. A *modal model* is one that switches between these simple models when the system transitions between regimes. The CT domain interoperates with the FSM domain to create modal models.

2.2.3 Discrete-Events - DE

In the discrete-event (DE) domain, the actors communicate via sequences of events placed in time, along a real time line. An *event* consists of a *value* and *time stamp*. Actors can either be processes that react to events (implemented as Java threads) or functions that fire when new events are supplied. This model of computation is popular for specifying digital hardware and for simulating telecommunications systems, and has been realized in a large number of simulation environments, simulation languages, and hardware description languages, including VHDL and Verilog. DE models are excellent descriptions of concurrent hardware, although increasingly the globally consistent notion of time is problematic. In particular, it over-specifies (or over-models) systems where maintaining such a globally consistent notion is difficult, including large VLSI chips with high clock rates. Every event is placed precisely on a globally consistent time line. The DE domain implements a fairly sophisticated discrete-event simulator. DE simulators in general need to maintain a global queue of pending events sorted by time stamp (this is called a *priority queue*). This can be fairly expensive, since inserting new events into the list requires searching for the right position at which to insert it. The DE domain uses a calendar queue data structure for the global event queue. In addition, the DE domain gives deterministic semantics to simultaneous events, unlike most competing discrete-event simulators. This means that for any two events with the same time stamp, the order in which they are processed can be inferred from the structure of the model. This is done by analyzing the graph structure of the model for data precedences so that in the event of simultaneous time stamps, events can be sorted according to a secondary criterion given by their precedence relationships. VHDL, for example, uses delta time to accomplish the same objective.

2.2.4 Distributed Discrete Events - DDE

The distributed discrete-event (DDE) domain can be viewed either as a variant of DE or as a variant of PN (described below). Still highly experimental, it addresses a key problem with discrete-event modeling, namely that the global event queue imposes a central point of control on a model, greatly limiting the ability to distribute a model over a network. Distributing models might be necessary either to preserve intellectual property, to conserve network bandwidth, or to exploit parallel computing resources. The DDE domain maintains a local notion of time on each connection between actors, instead of a single globally consistent notion of time. Each actor is a process, implemented as a Java thread, that can advance its local time to the minimum of the local times on each of its input connections. The domain systematizes the transmission of null events, which in effect provide guarantees that no event will be supplied with a time stamp less than some specified value.

2.2.5 Finite-State Machines - FSM

The finite-state machine (FSM) domain is radically different from the other Ptolemy II domains. The entities in this domain represent not actors but rather *state*, and the connections represent *transitions* between states. Execution is a strictly ordered sequence of state transitions. The FSM domain leverages the built-in expression language in Ptolemy II to evaluate *guards*, which determine when state transitions can be taken. FSM models are excellent for control logic in embedded systems, particularly safety-critical systems. FSM models are amenable to in-depth formal analysis, and thus can be used to avoid surprising behavior. The FSM domain in Ptolemy II can be hierarchically combined with other domains. We call the resulting formalism "*charts" (pronounced "starcharts") where the star represents a wildcard. Since most other domains represent concurrent computations, *charts model concurrent finite state machines with a variety of concurrency semantics. When combined with CT, they yield hybrid systems and modal models. When combined with SR (described below), they yield something close to Statecharts. When combined with process networks, they resemble SDL.

2.2.6 Process Networks - PN

In the process networks (PN) domain, processes communicate by sending messages through channels that can buffer the messages. The sender of the message need not wait for the receiver to be ready to receive the message. This style of communication is often called asynchronous message passing. There are several variants of this technique, but the PN domain specifically implements one that ensures determinate computation, namely Kahn process networks. In the PN model of computation, the arcs represent sequences of data values (tokens), and the entities represent functions that map input sequences into output sequences. Certain technical restrictions on these functions are necessary to ensure determinacy, meaning that the sequences are fully specified. In particular, the function implemented by an entity must be *prefix monotonic*. The PN domain realizes a subclass of such functions, first described by Kahn and MacQueen, where *blocking reads* ensure monotonicity. The PN domain in Ptolemy II has a highly experimental timed extension. This adds to the blocking reads a method for stalling processes until time advances. We anticipate that this timed extension will make interoperation with timed domains much more practical.

2.2.7 Synchronous Dataflow - SDF

The synchronous dataflow (SDF) domain handles regular computations that operate on streams. Dataflow models, popular in signal processing, are a special case of process networks. Dataflow models construct processes of a process network as sequences of atomic actor *firings*. Synchronous dataflow (SDF) is a particularly restricted special case with the extremely useful

property that deadlock and boundedness are decidable. Moreover, the schedule of firings, parallel or sequential, is computable statically, making SDF an extremely useful specification formalism for embedded real-time software and for hardware.

3. Future Capabilities

Capabilities that we anticipate making available in the future include:

- *Interoperability through software components.* Ptolemy II will use distributed software component technology such as CORBA, JINI, or DCOM, 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. Ptolemy II models will be exported via a server. And finally, Ptolemy II will support migrating software components.
- The *discrete-time (DT) domain* will extend the SDF domain with a notion of time between tokens. Communication between actors takes the form of a sequence of tokens where the time between tokens is uniform. Multirate models, where distinct connections have distinct time intervals between tokens, will be supported.
- In the *synchronous/reactive (SR) domain*, the arcs will represent data values that are aligned with global clock ticks. Thus, they are discrete signals, but unlike discrete time, a signal need not have a value at every clock tick. The entities represent relations between input and output values at each tick, and are usually partial functions with certain technical restrictions to ensure determinacy. Examples of languages that use the SR model of computation include Esterel, Signal, Lustre, and Argos.
- *Embedded software synthesis.* Pertinent Ptolemy II domains will be tuned to run on a Java virtual machine on an embedded CPU. Domains that seem particularly well suited to this approach include PN and CSP.
- *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.
- *Reflection of dynamics.* Java supports reflection of static structure, but not of dynamic properties of process-based objects. For example, the data layout required to communicate with an object is available through the reflection package, but the communication protocol is not. We plan to extend the notion of reflection to reflect such dynamic properties of objects.

4. Publications

This project has generated a 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**, *Annals of Software Engineering*, vol. 7 (1999), pp. 25-45.

[2] W. A. Najjar, E. A. Lee, G. R. Gao, "Advances in the dataflow computational model," *Parallel Computing*, vol. 25 (1999) pp. 1907-1929.

[3] E. A. Lee and D. G. Messerschmitt, "A Highest Education in the Year 2049," *Proceedings of the IEEE*, Volume 87 Number 9, September 1999 (**Invited paper**).

[4] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," *IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems*, Vol. 18, No. 6, June 1999.

[5] Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee, "Synthesis of Embedded Software from Synchronous Dataflow Specifications," *Journal of VLSI Signal Processing Systems*, Vol. 21, No. 2, June 1999.

4.2. Conference Papers

[6] J. Liu, X. Liu, T. J. Koo, B. Sinopoli, S.Sastry, and E. A. Lee, "Hierarchical Hybrid System Simulation," *38th IEEE conference on Decision and Control*, Dec. 1999, Phoenix, AZ.

[7] J. Liu, B. Wu, X. Liu, and E. A. Lee, "Interoperation of Heterogeneous CAD Tools in Ptolemy II," *Symposium on Design, Test, and Microfabrication of MEMS/MOEMS*, March 1999, Paris, France.

4.3. Masters Reports

[8] Lukito Muliadi, "Discrete Event Modeling in Ptolemy II," MS Report, Dept. of EECS, University of California, Berkeley, CA 94720, May 1999.

4.4. Other Technical Reports

[9] Edward A. Lee, "Embedded Software - An Agenda for Research," ERL Technical Report UCB/ERL No. M99/63, Dept. EECS, University of California, Berkeley, CA 94720, December 15, 1999.

[10] 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. M99/40, University of California, Berkeley, CA 94720, July 19, 1999.

[11] John Davis II, Mudit Goel, Christopher Hylands, Bart Kienhuis, Edward A. Lee, Jie Liu, Xiaojun Liu, Lukito Muliadi, Steve Neuendorffer, John Reekie, Neil Smyth, Jeff Tsay and Yuhong Xiong, "Overview of the Ptolemy Project," ERL Technical Report UCB/ERL No. M99/37, Dept. EECS, University of California, Berkeley, CA 94720, July 1999.

[12] J. Reekie, S. Neuendorffer, C. Hylands and E. A. Lee "Software Practice in the Ptolemy Project," Technical Report Series, GSRC-TR-1999-01, Gigascale Semiconductor Research Center, University of California, Berkeley, CA 94720, April 1999.