DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator

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

Final Report 2000-01, Micro Project #00-053
Industrial Sponsors: Agilent, Cadence, Hughes (Boeing), 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 programming environments intended for signal processing the Advanced Development System (ADS), which is based on Ptolemy Classic, from Agilent, the signal processing worksystem (SPW), from Cadence, CoCentric Design Studio, from Synopsys, and Simulink, from The MathWorks. SPW and CoCentric both use dataflow models that were developed as part of this project.

All of these software environments define applications as assemblies of components that are coordinated in some way. Many possibilities have been explored for precise semantics of the coordination. Many of these limit expressiveness in exchange for considerable advantages such as compile-time predictability. In Ptolemy, a *domain* defines the semantics of the coordination between components. Domains are modular objects that can be mixed and matched at will, thus getting a rich and rigorous approach to heterogeneous modeling.

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. A major part of the work in the next period will be on such compilation.

2. Results of Micro Support

2.1. Ptolemy II

We have built a second generation of design software called 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.

2.2. Hardware in the loop

Embedded systems are a key application area for the Ptolemy project. These systems are characterized by closely interacting with the physical world through various sensors and actuators. Ptolemy's support for modeling heterogeneous systems allows us to easily embed a discrete model of a digital system into the continuous model of its physical environment, and often this is an appropriate modeling approach.

However, in many cases it would be preferable to execute the model of the digital system directly in its physical context, raising the issue of interfacing a model to sensors and actuators. Winthrop Williams has worked on extending the Ptolemy software infrastructure to allow models to communicate with external hardware.

The first such attempt successfully demonstrated remote teleoperation with force feedback at a sample rate of 125 Hz. Difficulties encountered are raising questions of how models of computation can and should interact with physical processes. Models may be augmented to define how they engage physicality, for example by encompassing threads which may block awaiting hardware events.

A major objective of this work is to move towards software development tools which facilitate understanding of the system being designed. Models of computation are a part of this quest, as are semantics and syntactic representations. This research is biased towards syntactic systems like Ptolemy II which include diagrams analogous to the schematic diagrams of electronic circuitry.

2.3. Network components

A large number of embedded systems applications require the coordination of physically separated components. Distributing system components across a network can improve robustness of a system and simplify its architecture by allowing components to run concurrently and independently. It also facilities exploitation of the intrinsically parallel nature of specialized hardware, offering the promise of improved execution speed.

Traditional distributed computing is built on the client-server model, which lacks object-orientation and is difficult to scale up. Middleware technologies, like the Common Object Request Broker Architecture (CORBA) and the Distributed Common Object Model (DCOM) offer object models and scalability, but the programming model is too liberal to analyze formal properties of a system.

As a first step, we have demonstrated importing a remote model as a component of a larger model, executing the model in a distributed fashion over a network. We have also implemented a publish/subscribe type of message passing mechanism based on JINI and JavaSpaces. Current work includes the definition of a clean interface for distributed components and their interaction, and the exploration of real-time issues in the context of this framework.

In order to support distributed components, four distributed objects--- ports, receivers, parameters, and executable interfaces---were exported via CORBA. These objects together provide an abstraction of components in Ptolemy II. A remote (composite) actor is viewed as a service that can be accessed through these objects.

2.4. Graphical modeling and animation

One of the many applications of heterogeneous modeling is the construction of digital systems interacting with a *mechanical* environment. Of course, such a system can be observed through a number of variables, and the Ptolemy software infrastructure contains a rich library of actors for displaying and graphing the changes of such variables.

In the case of mechanical systems, however, a direct animation of its movements provides a much clearer and more intuitive representation of the changes in the system over time. Chamberlain Fong worked on designing and implementing a 3D visualization engine for in the Ptolemy II framework.

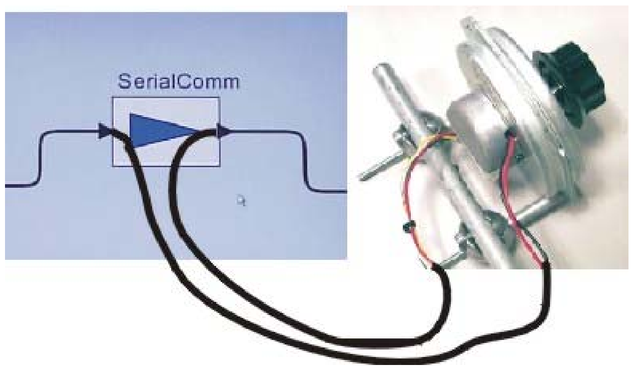

Figure 1. Hardware in the loop.

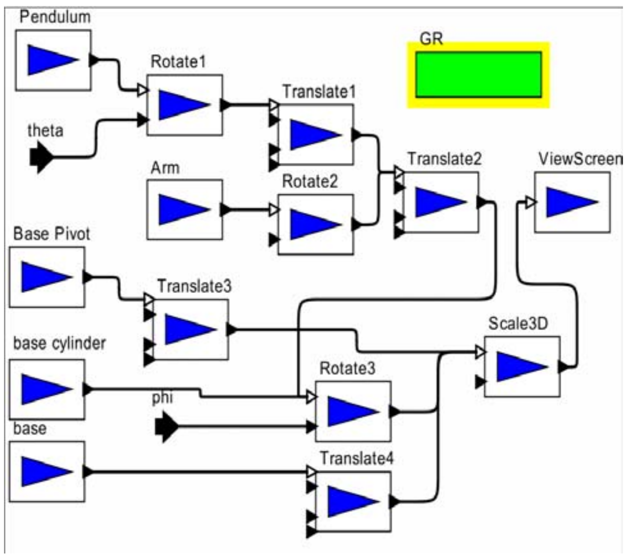

Figure 2. Specification of 3-D graphics display.

Based on the Java 3D infrastructure, he designed a visual language for expressing three-dimensional models, which depend on parameters that define, e.g., the position, size, color, or orientation of some of the objects in the model (see figure 2). Actual values for these parameters are supplied by an executing Ptolemy model connected to the 3D-model. With these values changing over time, the 3D-model produces an animation. Combined with "traditional" displays of the change of values during the execution of a model, 3D-animation makes it much easier for users to understand the operation of the system that they model (see figure 3).

2.5. Type system extensions: structured and behavioral types

Type systems in modern programming languages are the most widely used and effective (semi)formal verification tools for software. The theory underlying type systems is simple, robust, and extremely powerful. The mathematical structure of a set of types is abstractly a lattice, and the theory can be applied to any property of the software that can be characterized using a lattice. In particular, it need not be restricted to classical data types. In view of this, we have designed a set of Java classes in Ptolemy II that provide support for any type system that can be represented using such a lattice. This generic infrastructure has been first applied to the basic type system problem, that of ensuring consistent data typing, and supporting (polymorphic) type inference. But the infrastructure is designed to be extended to more sophisticated uses.

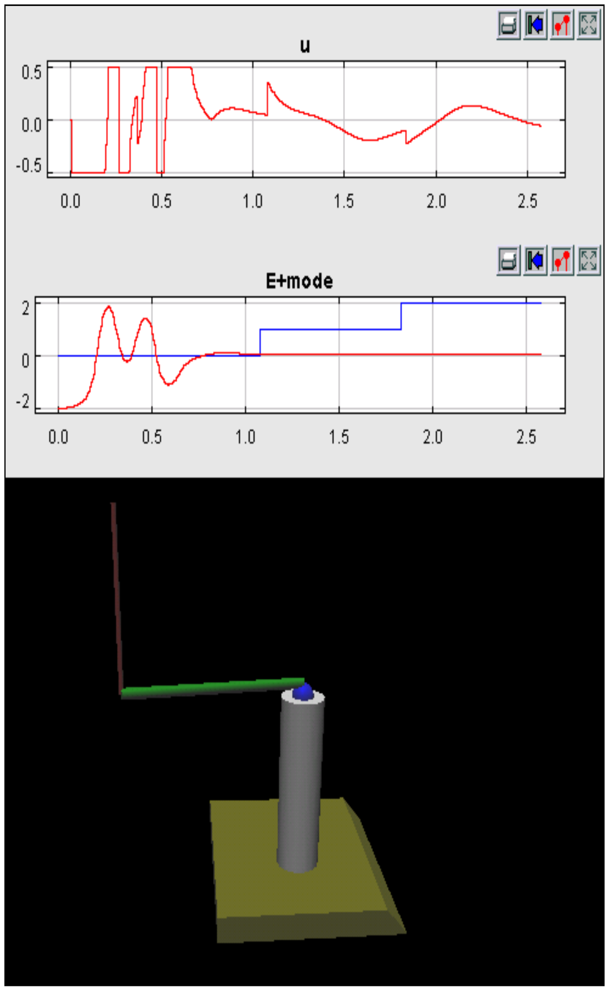

Figure 3. 3-D graphics display animating a model.

We are currently developing two extensions. One extension is to add support for structured types such as array and record types. The goal is to allow the elements of arrays and records to contain data tokens of arbitrary types, including structured types, and be able to specify type constraints on them. Type constraints and type inference must propagate transparently across operators that construct and disassemble such structures. One of the major difficulties in this extension is that the type lattice becomes infinite, which raises questions on the convergence of type checks and inference.

A more innovative (and speculative) extension is to process-level types. These types represent dynamic properties of an application, rather than the static data types traditionally dealt with in a type system. [7] One of the dynamic properties we have studied is communication protocol. We have attempted to characterize different communication protocols as types and describe these types using automata. Furthermore, we have organized these types into a system-level type lattice using the simulation relation. This lattice provides significant insight into the relation between various protocols and may help design polymorphic components that can work with multiple protocols. For example, we can specify the behavior of a polymorphic component using a non-deterministic automaton. If this automaton simulates the automata of some specific protocols, the component will be able to work with those protocols.

2.6. Status

This report period saw the release of the first full version of the Ptolemy II software (version 1.0.1 in March 2001). This was the first major release to include Vergil, a graphical user interface supporting block diagram editing of Ptolemy II models. It also includes a set of mature and experimental domains, and a more comprehensive actor library than previous releases. Ptolemy II 1.0.1 supports an XML schema called MoML for specifying component-based models.

On March 22-23, 2001 the 4th Biennial Ptolemy Mini conference was held at the Claremont Hotel in Berkeley, with 93 attendees from 44 organizations worldwide

3. Publications

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

3.1. Journal Articles

[1] Edward A. Lee, "What's Ahead for Embedded Software?," *IEEE Computer*, September 2000, pp. 18-26.

3.2. Conference Papers

[2] Jozsef Ludvig, James McCarthy, Stephen Neuendorffer, and Sonia R. Sachs, "Reprogrammable Platforms for High-speed Data Acquisition," Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers, Asilomar Hotel Conference Grounds, November 4-7, 2001.

[3] Johan Eker, Chamberlain Fong, Jörn W. Janneck, and Jie Liu, "Design and Simulation of Heterogeneous Control Systems using Ptolemy II," IFAC Conference on New Technologies for Computer Control (NTCC'01), Hong Kong, China, November 2001.

[4] Edward A. Lee and Yuhong Xiong, "System-Level Types for Component-Based Design," First Workshop on Embedded Software, EMSOFT2001, Lake Tahoe, CA, USA, Oct. 8-10, 2001.

[5] Jie Liu, Stan Jefferson, and Edward A. Lee, "Motivating Hierarchical Run-Time Models in Measurement and Control Systems," 2001 American Control Conference, June 25-27, 2001, Arlington, VA, pp. 3457-3462.

[6] Edward A. Lee, "Computing for Embedded Systems," IEEE Instrumentation and Measurement Technology Conference, Budapest, Hungary, May 21-23, 2001.

3.3. Ph.D. Dissertations

[7] Jie Liu, "Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems," Ph.D. thesis, Technical Memorandum UCB/ERL M01/41, University of California, Berkeley, CA 94720, December 20th, 2001.

3.4. Masters Reports

[8] Paul Whitaker, "The Simulation of Synchronous Reactive Systems In Ptolemy II," Master's Report, Memorandum UCB/ERL M01/20, Electronics Research Laboratory, University of California, Berkeley, May 2001.

[9] C. Fong, "Discrete-Time Dataflow Models for Visual Simulation in Ptolemy II," Master's Report, Memorandum UCB/ERL M01/9, Electronics Research Laboratory, University of California, Berkeley, January 2001

3.5. Other Technical Reports

[10] Stephen A. Edwards and Edward A. Lee, "The Semantics and Execution of a Synchronous Block-Diagram Language,"Technical Memorandum UCB/ERL M01/33, University of California, Berkeley, CA 94720, October 25, 2001

[11] Edward A. Lee, "Soft Walls - Modifying Flight Control Systems to Limit the Flight Space of Commercial Aircraft (Draft 2),"Revised from Technical Memorandum UCB/ERL M01/31, University of California, Berkeley, CA 94720, October 3, 2001.

[12] Edward A. Lee, "Overview of the Ptolemy Project," Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, March 6, 2001.