DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator

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

Final Report 2002-03, Micro Project #02-037
Industrial Sponsors: Agilent, Atmel, National Semiconductor

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. Checking valid composition

Modern programming languages and design systems have heavily embraced the use of data type systems for robust specification of programs. Ptolemy II, in particular, includes a rather sophisticated data type system that allows for insertion of automatic conversions between types. Data types systems generally focus on the analysis of properties of a program or model by approximating all behaviors of the system. By formally guaranteeing type-safety properties, data type systems ensure that run-time type errors (such as interpreting an integer as a floating point number) cannot happen.

However, there may be many properties of a model that a designer cares about, in addition to type-safety. In component-based design systems the varied interactions between components often result in complex interactions. While it is easy for a designer to specify the behavioral constraints that these interactions must satisfy, it is somewhat more difficult for a designer to ensure that those constraints are satisfied. In this period, we have developed the notion of behavioral type systems that extend data type systems to include these behavioral constraints. The goal of a behavioral type system is to automatically verify that these constraints are satisfied.

We have worked on development of a behavioral type system with the goal to ensure that actors in Ptolemy II satisfy the constraints of the model of computation in which they are used. This *system-level type system* is based on interface automata to express the assumptions and guarantees of a model of computation and actor and uses model checking techniques to explore the state space of the product automata to ensure correctness. This work has required extending existing notions of interface automata to more directly represent method calls and synchronization monitors to properly represent actor interfaces.

We have also developed an independent behavioral type system that focuses on system reconfiguration. This *reconfiguration type system* analyzes a model to determine which parameters are reconfigured and at what points during the execution of a model reconfiguration occurs. This information can be used to guarantee that particular parameters (such as parameters representing data types, or parameters used for dataflow scheduling) are not reconfigured at run-time. The information can also be used at a finer level of granularity to support parameterized dataflow scheduling where parameters are allowed to change, as long as they are not reconfigured during the execution of a schedule. Information about reconfiguration can also be used during code generation to balance optimization trade-offs or guide selection of an appropriate implementation architecture, such as an FPGA. This type system is particular interesting in the context of Ptolemy II because it is independent of how the reconfiguration is modeled and also independent of the particular models of computation used.

The reconfiguration type system analyzes the reconfiguration in a model through *quiescent points*. The quiescent points of an actor are those execution points where the actor is inactive and not performing communication or computation. The structure of quiescent points of actors in a hierarchical model is also hierarchical; when a model is quiescent, every actor in the model must also be quiescent. During a quiescent points of an actor, the actor is allowed to reconfigure the model by modifying the value of a single parameter. Ptolemy II responds to this reconfiguration by updating the values of all parameters dependent on the reconfigured parameter. Essentially, the reconfiguration type system incorporates all possible sources of reconfiguration and all parameter dependencies to describe the set of quiescent points during which reconfiguration can occur.

We have also developed a behavioral type system that will improve the ability of Ptolemy II to deal effectively with hierarchical components. This *dependence type system* describes how data produced from individual output ports of a hierarchically described component depend on data received from individual input ports. This dependence information is often required to ensure that models have unique and well-defined behaviors, particularly in the discrete-event model of computation. Without this dependence information, the model can still be executed, but is not guaranteed to be consistent or deterministic.

2.3. Sensor Networks

A large number of embedded systems applications require the coordination of physically separated components, or networked embedded sub-systems. In particular, embedded systems consisting of small sensor nodes communicating through ad-hoc wireless networks have recently become an interesting platform for distributed monitoring and surveillance applications. However, the embedded nature of these platforms and the complex interaction between individual applications and ad-hoc network infrastructure makes developing applications particularly difficult.

To address these issues, we have developed a version of Ptolemy II that is targeted to modeling sensor networks. This tool, called VisualSense, extends Ptolemy II to represent component models where communication links are not explicitly specified. Instead, output ports reference a communication channel by name and the channel broadcasts messages to input ports that also reference the channel by name. This model is appropriate for modeling sensor networks where a communication broadcast can only be received by other sensor nodes that are sufficiently close to receive data.

The goal of VisualSense is the construction of not only models of sensor nodes, but to leverage hierarchical models in Ptolemy II to describe different aspects of the system, including channel propagation and interference, communication protocols for managing the ad-hoc network, physical processes being monitored, signal processing occurring in sensor nodes, and management and display functionality executing on a base station. This ability goes directly to targeting the complex interactions in sensor networks. We have already made significant progress in the capabilities of the modeling environment, although this work will continue.

Currently, VisualSense contains several channel models that represent propagation delay, probabilistic message loss, communication range and power loss through propagation. These abstract channel models can represent not only wireless radio communication in sensor networks, but also other communication media such as sound or light. Models of sensor nodes can interact with these propagation models to add transmitter or receiver specific properties such as transmission power, receiver sensitivity, and

antenna directionality models. We have also demonstrated models of message collision and multi-hop propagation through the ad-hoc network.

In addition to the above built-in channel models, VisualSense includes the ability to model the behavior of a sensor node using a hierarchical model. This capability allows the use of arbitrary Ptolemy II models to represent individual sensor nodes. CT models can be used to represent a physical process being sensed. Giotto models can be used to represent real-time operating systems. DE models can be used to represent the time of incoming events and the triggering of how those events are processed. SDF models can be used to represent embedded signal processing algorithms running on the sensor.

By combining these various models, VisualSense allows the complex interactions between the parts of these systems to be simulated without concern for the low-level implementation. Our future goals are to increase the expressiveness of the modeling environment and look for a path to target existing implementation technologies through code generation. Although significant optimization will have to occur in order to execute on heavily resource-constrained sensor nodes, we anticipate that most high-level design work can be done using VisualSense without concern for premature optimization.

2.4. Cache-aware scheduling

When dealing with embedded systems with real-time constraints, memory caches and memory hierarchy make analyzing systems much more difficult. In particular, although caches tend to improve average case performance of a signal processing system, they do not appreciably affect worse-case performance. In our approach to system design that is based on code generation, dealing with caches systematically is even more important, since we would like to abstract a designer from dealing with such issues at the assembly-code level.

We have developed an approach to scheduling SDF models that considers memory hierarchy when making scheduling decisions. In particular, the approach deals with scheduling well-ordered SDF models on a single embedded Digital Signal Processor (DSP) with Harvard memory architecture (i.e., separate instruction and data memory). In order to achieve more predictable and efficient behavior for SDF models, the data cache is managed using a specialized hardware cache-replacement policy and the instruction cache is managed as a scratchpad memory with software-controlled replacement policy. By using these improved cache-replacement policies, analyzing the behavior a schedule without executing it becomes simpler. Simplified analysis allows a greedy scheduling algorithm to make better trade-offs between instruction cache usage and data cache usage.

The intuition behind the scheduling algorithm is that a static scheduler essentially makes decisions about whether to continue to execute a single actor (making use of more data memory, while executing the actor code out of instruction cache) or to switch to another actor (which will generally reduce the amount of data memory used while possibly incurring an instruction-cache miss. In other words, reducing data cache misses often increases instruction cache misses and vice versa. Because of this interaction, existing scheduling techniques often create schedules that perform poorly with respect to cache usage. Sanjeev's heuristic algorithm approximates instruction and cache miss penalties, resulting in a schedule that executes faster than a naive schedule.

2.5. Status

In this report period a new major version of the Ptolemy II software was released (version 3.0 in July 2003). This release also included an updated release of the targeted HyVisual tool for modeling hybrid systems. Hybrid systems are systems with continuous-time dynamics, discrete events, and discrete mode changes. This visual modeler supports construction of hierarchical hybrid systems. It uses a block-diagram representation of ordinary differential equations (ODEs) to define continuous dynamics. It uses a bubble-and-arc diagram representation of finite state machines to define discrete behavior.

HyVisual is illustrative of the ability to create domain-specific, targetted, and separately branded tools that use the Ptolemy II infrastructure. We expect to do much more of that in the future.

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 and Yuhong Xiong, "A Behavioral Type System and Its Application in Ptolemy II," to appear in *Aspects of Computing Journal,* special issue on "Semantic Foundations of Engineering Design Languages." This version: November 10, 2003.

[2] Stephen A. Edwards and Edward A. Lee, "The Semantics and Execution of a Synchronous Block-Diagram Language," *Science of Computer Programming*, Vol. 48, no. 1, July 2003.

[3] Edward A. Lee, Stephen Neuendorffer and Michael J. Wirthlin, "Actor-Oriented Design of Embedded Hardware and Software Systems," Invited paper, *Journal of Circuits, Systems, and Computers*, Vol. 12, No. 3 pp. 231-260, 2003.

3.2. Conference Papers

[4] Jie Liu and Edward A. Lee, "On the Causality of Mixed-Signal and Hybrid Models," 6th International Workshop on Hybrid Systems: Computation and Control (HSCC '03), April 3-5, 2003, Prague, Czech.

[5] Yan Jin, Robert Esser, Charles Lakos, Jörn W. Janneck, "Modular Analysis of Dataflow Process Networks," Proceedings Fundamental Approaches to Software Engineering (FASE) 2003.

[6] Elaine Cheong, Judy Liebman, Jie Liu, and Feng Zhao, "TinyGALS: A Programming Model for Event-Driven Embedded Systems," *Proceedings of the 18th Annual ACM Symposium on Applied Computing* (SAC'03), Melbourne, FL, Mar. 9-12, 2003.

3.3. Masters Reports

[7] Ye Zhou, "Communication Systems Modeling in Ptolemy II," Master's Report, Technical Memorandum No. UCB/

ERL M03/53, University of California, Berkeley, CA, 94720, USA, December 18, 2003.

[8] James Yeh, "Image and Video Processing Libraries in Ptolemy II," Master's Report, Technical Memorandum No. UCB/ERL M03/52, University of California, Berkeley, CA, 94720, USA, December 16, 2003.

[9] Yang Zhao, "A Model of Computation with Push and Pull Processing," Master's Report, Technical Memorandum No. UCB/ERL M03/51, University of California, Berkeley, CA, 94720, USA, December 16, 2003.

[10] Elaine Cheong, "Design and Implementation of TinyGALS: A Programming Model for Event-Driven Embedded Systems," Master's Report, *Technical Memorandum No. UCB/ERL M03/14*, University of California, Berkeley, CA, 94720, USA, May 23, 2003.

[11] Adam Cataldo, "Control Algorithms for Soft Walls," Master's Report, *Technical Memorandum UCB/ERL M03/42*, University of California, Berkeley, CA 94720, January 21, 2004.

3.4. Other Technical Reports

[12] Stephen Neuendorffer and Edward A. Lee, "Hierarchical Reconfiguration of Dataflow Models," *Technical Memorandum UCB/ERL M04/2*, University of California, Berkeley, CA 94720, USA, January 2004.

[13] Johan Eker and Jorn W. Janneck, "CAL Language Report: Specification of the CAL actor language," *Technical Memorandum No. UCB/ERL M03/48*, University of California, Berkeley, CA, 94720, USA, December 1, 2003

[14] Christopher Hylands Brooks and Edward A. Lee, "Ptolemy II Coding Style," *Technical Memorandum UCB/ERL M03/44*, University of California at Berkeley, November 24, 2003.

[15] Edward A. Lee, "Soft Walls: Frequently Asked Questions," *Technical Memorandum UCB/ERL M03/31*, University of California, Berkeley, CA 94720, July 21, 2003.

[16] Stephen Neuendorffer, "Implementation Issues in Hybrid Embedded Systems," *Technical Memorandum No. UCB/ERL M03/22*, University of California, Berkeley, CA, 94720, USA, June 24, 2003.