An Overview of the Ptolemy Project

Authors:

Shuvra Bhattacharyya
Joseph T. Buck
Wan-Teh Chang
Michael J. Chen
Brian L. Evans
Edwin E. Goei
Soonhoi Ha
Paul Haskell
Chih-Tsung Huang
Wei-Jen Huang
Christopher Hylands
Asawaree Kalavade
Alan Kamas
Allen Lao
Edward A. Lee
Seungjun Lee
David G. Messerschmitt
Praveen Murthy
Thomas M. Parks
José Luis Pino
Gilbert Sih
S. Sriram
Michael C. Williamson
Kennard White

DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCES

UNIVERSITY OF CALIFORNIA AT BERKELEY

Contents:

Introduction

Ptolemy is a system-level design framework that allows mixing of multiple models of computation. The project began in January of 1990, under the joint direction of Professors Lee and Messerschmitt. It has grown to directly involve approximately 20 research students and four FTE staff positions. The largest current sponsors of the Ptolemy project are the Advanced Research Projects Agencies (ARPA), in cooperation with The United States Air Force, as part of the RASSP effort (rapid prototyping of application-specific signal processors), and the Semiconductor Research Corporation (SRC). The Ptolemy project and related projects have also received funding and other support from the National Science Foundation (NSF), the Office of Naval Technology (ONT) via the Naval Research Laboratory (NRL), the State of California MICRO Program, and 17 companies. At Berkeley, in addition to the core research project, approximately 100 undergraduate and graduate students are currently using Ptolemy as an integral part of two courses, and a number of other research groups at Berkeley are using Ptolemy to conduct simulation and design integral to their research. The latest version of Ptolemy (designated 0.5[1]) is now available at the ftp site ptolemy.eecs.berkeley.edu.

The ambitious objectives of the project include practically all aspects of designing signal processing and communications systems, ranging from the design of algorithms and communication strategies, through simulation, hardware and software design, parallel computing, and real-time prototyping. To accommodate this breadth, Ptolemy must support a plethora of widely differing design styles. Hence, the core of Ptolemy is an object-oriented toolkit and library that makes few assumptions about the system to be modeled; rather, standard interfaces are provided for generic objects and more specialized, application-specific objects are derived from these.

This document gives a brief non-technical overview of the project.

A. The Design of the Ptolemy Kernel

Ptolemy is an object-oriented system and achieves its goals using the principle of polymorphism. The Ptolemy kernel, written in C++, defines the basic classes that allow the components of the system to function together; application-specific objects are derived from these. Information hiding and data abstraction are key to the design, so that it is easy to modify; a key goal is that the system be extensible in many dimensions without the need to modify the kernel.

The basic unit of computation in Ptolemy is the Block; a Block may be atomic (Star) or composite (Galaxy). Objects derived from class Scheduler determine the order of execution of the Blocks; the combination of a type of scheduler, blocks whose behavior corresponds to the behavior expected by that scheduler, and other supporting objects make up a domain. Hence

1. Since Ptolemy is research software, distributed free of charge and without support, all versions distributed by the University are designated 0.x.

the domain determines the operational semantics of a network of blocks. Blocks may have state variables, which serve as user-settable parameters or as named variables that Blocks can modify and users can monitor. Blocks exchange data between themselves as they execute; data are communicated in chunks called Particles, which may be of many types — integer, real, complex, fixed-point, matrix, or a general structure (a data packet, for example).

The kernel supports methods for building up simulations based on user commands — creating instances of blocks, connecting them together, and running the simulation. A schema compiler has been developed for easily defining new blocks. The window at the lower right of figure 2 uses this syntax. Two user interfaces are currently available, the graphical user interface shown in the figures and a textual interpreter based on Tcl [11][12][13]. More details about the software architecture can be found in [3].

A.1 Models of Computation

Each design style is supported by one or more *domains*, where a domain is an extensible library of functional blocks and a model of computation. By "model of computation" we mean the operational semantics of a network of functional blocks. Key to Ptolemy is the ability to combine multiple domains in a single design. Some of the domains we are currently working on are summarized in figure 1 and table 1. A few of these are explained below with some illustrative examples of their uses.

A.2 Multirate Signal Processing

The *synchronous dataflow* (SDF) domain models synchronous multirate signal processing applications, as in figure 2. This figure shows a simple transmitter for a data modem. Symbols representing data are generated, pulses are formed for in-phase and quadrature parts, and these parts are modulated on two carriers 90 degrees out of phase with one another.

A typical block from the SDF library is the raised cosine pulse shaper, labeled in the block diagram at the lower left. A small segment of the definition of this block is given at the lower right, where we can see that it is derived from another block called FIR. Indeed, it is simply a finite impulse-response filter with a frequency response that has the shape of a raised cosine. It is defined as a C++ class derived from the more basic filter class. More interestingly, the filter changes the sample rate of the signal. The parameter window just above the block definition indicates that the filter will interpolate by a factor of 16. In the SDF domain, this means that every firing of the raised cosine filter will consume one new input sample and produce 16. Above the block diagram is an eye diagram, showing how 4-level symbols have been mapped into a signal with gradual transitions between symbols. This plot is modeled after an oscilloscope display because such a display would be familiar to modem designers. To the right, a constellation display shows that there are four levels in each of the in-phase and quadrature parts. At the top, a section of the transmit signal is shown. More interesting applications in this class that have been developed include spectral analysis systems, speech coders, video coders, audio sample-rate converters, beamformers, wavelet transforms, channel models, and digital communication receiver structures.

A.3 Vector Operations

Many vector operations can be conceptually performed by serializing the vector in a stream of data. An example is shown in figure 3. The system shown performs 128-point FFTs on data from each of 8 sensors. The synthetic excitation consists of two plane waves at different frequencies. The direction of arrival of the plane waves is rotated through the complete circle in 200 steps, and 128 samples are generated at each step. Ptolemy generates the plots shown in about 1 minute on a Sparc 2 using the SDF domain. This corresponds to processing roughly 400 samples per second from all 8 sensors. Many practical systems have many more sensors, may use higher order FFTs, and may be more complex, including for example noise spectrum equalization.

The representation in figure 3 is entirely compatible with SDF principles. This basic technique for handling vector operations is commonly used in Ptolemy today. Most interestingly for future applications of Ptolemy, this representation imposes no fundamental limitations on parallelism. Interdependencies between iteration stages are easily represented, and easily analyzed by a scheduler. Conceptually, the single FFT block in figure 3 is multiplexed among the many signals from the sensors. In implementation, however, a scheduler can analyze the

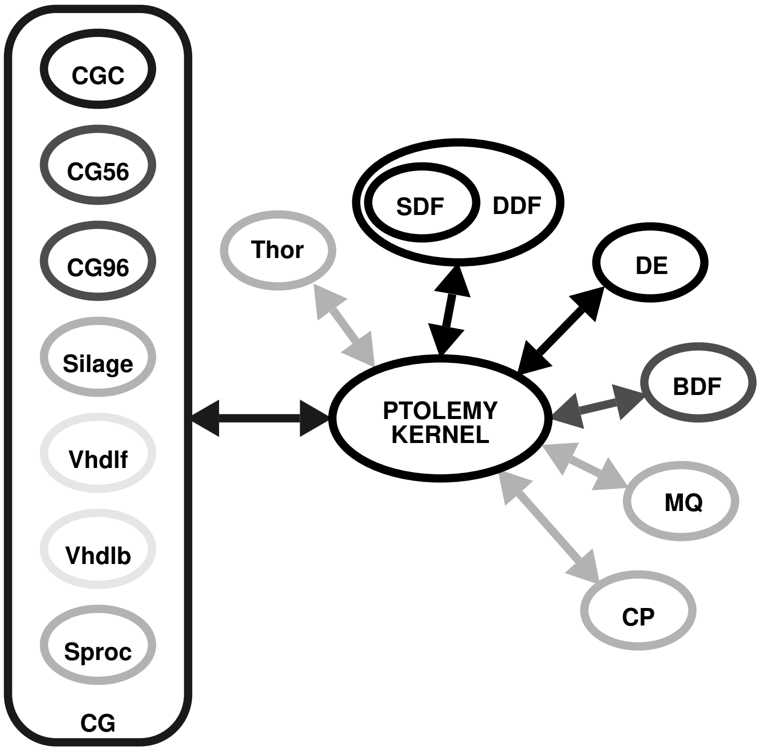

FIGURE 1. A summary of the domains that have been developed and are included in the 0.5 distribution. The domains with the lighter grey outlines are preliminary and experimental, and form a significant part of our current research. The domains on the left are all code generation (CG) domains.

dependencies among multiple firings of the FFT block, and synthesize a parallel implementation. More about this later.

A.4 Asynchronous Signal Processing

Asynchronous signal processing applications can be built using the dynamic dataflow (DDF) domain in Ptolemy. An example is shown in figure 4. Here, a phase-locked loop controls the downsampling of an amplitude-shift-keyed (ASK) signal. This PLL is being used to perform timing recovery for the digital communication signal.

The DDF domain uses run-time scheduling, whereas the SDF domain uses compile-time scheduling. A recent innovation was the development of the BDF domain, which does compile-time scheduling for most dynamic dataflow graphs [4].

A.5 Queueing and Communications Networks

A radically different application is shown in figure 5. It uses the discrete-event (DE) domain in Ptolemy, in which signals consist of sequences of data objects with associated time stamps. The execution is chronological rather than data-driven. The figure shows a system with two

Name	Expansion	Principal Use
SDF	synchronous dataflow	synchronous signal processing
DDF	dynamic dataflow	asynchronous signal processing
DE	discrete event	communication network modeling or high-level system modeling
BDF	Boolean dataflow	asynchronous signal processing
Thor	(name given at Stanford)	RTL hardware simulation
MQ	Message Queue	Telecom switching software
CP	communicating processes	communication network modeling high-level system modeling
CGC	code generation - C	synchronous signal processing
CG56	code generation - DSP56000	synchronous signal processing
CG96	code generation - DSP96000	synchronous signal processing
Silage	(a functional language)	VLSI hardware synthesis
Vhdlf	functional modeling in VHDL	synchronous signal processing
Vhdlb	behavioral modeling in VHDL	RTL or high-level system modeling
Sproc	(name of a multiprocessor DSP from Star)	firmware synthesis (synchronous DSP)

Table 1: An expansion of the domain names in figure 1. The Verilog and OE domains have been developed by Prof. Borriello's group at the Univ. of Washington.

queues and two servers, where the first queue discards incoming events when it fills up, whereas the second queue blocks arrivals instead. The two signal plots monitor the sizes of the queues. Practical applications of this capability include modeling of communication networks and protocols and high-level modeling of hardware architectures.

We are pursuing simulation methodologies for communications networks (such as broadband packet networks), and in particular methodologies that support the simulation of interactions between three key elements: signal processing (such as video compression), transport (such as

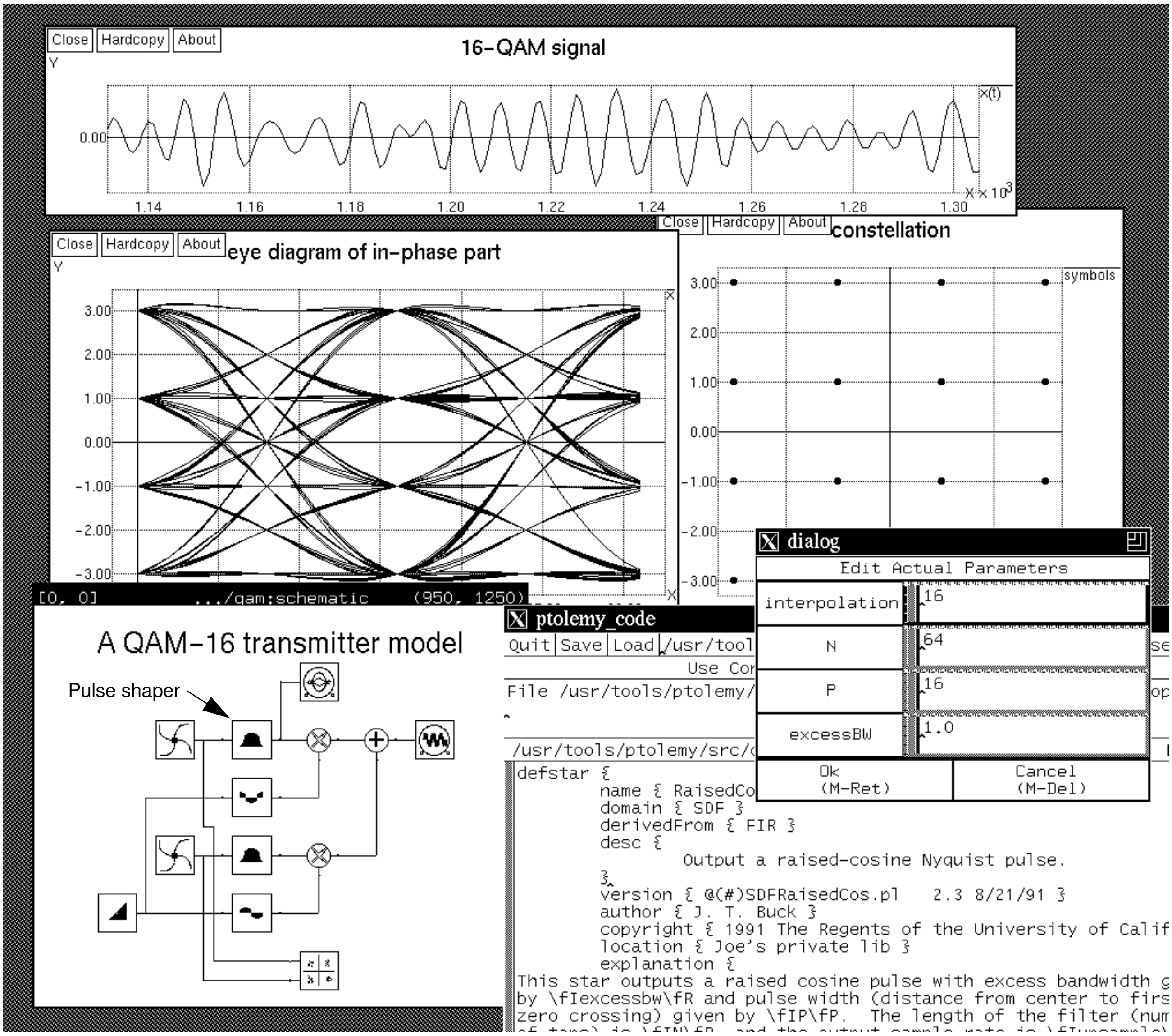

FIGURE 2. A multirate signal processing application represented in the synchronous dataflow domain in Ptolemy.

cell-relay or ATM), and control (signaling and call processing). A multimedia networking application that combines all these elements is shown in figure 6.

One serious task that we are setting out to accomplish is to add a set of carefully-designed and well-documented models to Ptolemy in two areas (that complement our current set of DSP models): packet networking and wireless networking (fading channel, modulation, and multi-access). In both these application areas we have several interested research groups at Berkeley that are using Ptolemy and contributing to the model libraries. Task forces are currently working to define the basic abstractions. Beyond the ability to model heterogeneous systems, it turns out that the abstraction mechanisms of object-oriented programming are very powerful in decoupling different parts of the system. For example, our models will allow different lev-

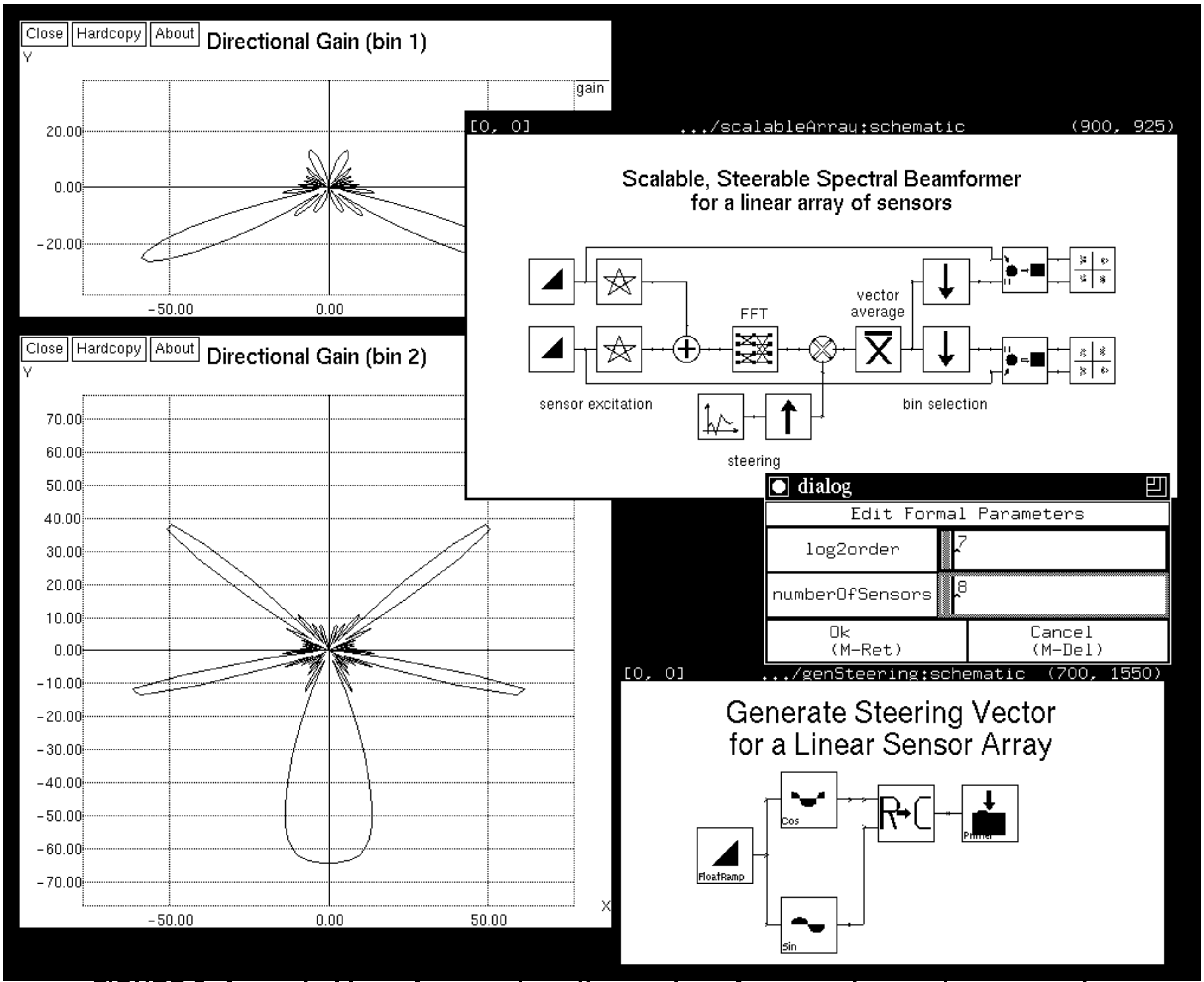

FIGURE 3. A spectral beamformer where the number of sensors is a system parameter (In these plots, there are eight sensors). The two plots of gain vs. azimuth correspond to two frequency bins. A simple steering vector (generated in the lower block diagram) is being used, so the azimuth of the main lobe in the two frequency bins is different.

els of link modeling (packet discard, bit-by-bit error models, full simulation of the physical-layer signal processing) to be freely substituted with no effect whatever on the signal processing or networking simulations.

A.6 Software, Firmware, and Hardware Synthesis

The domains described above are all used for algorithm development and system simulation. Ptolemy also supports system implementation through code generation. For example, any SDF system can be translated into a stand-alone C++ program that runs autonomously of Ptolemy. Small, efficient C programs can also be generated using the SDF model of computation in the CGC domain (see figure 7). The compactness of these programs follows from maximizing the use of compile-time information in the SDF model of computation. Several other code generation capabilities are under various stages of development, including synthesis of C code for real-time control applications and generation of assembly code for the Motorola DSP56001 and DSP96002 and the Sproc multiprocessor DSP from Star Semiconductor. We

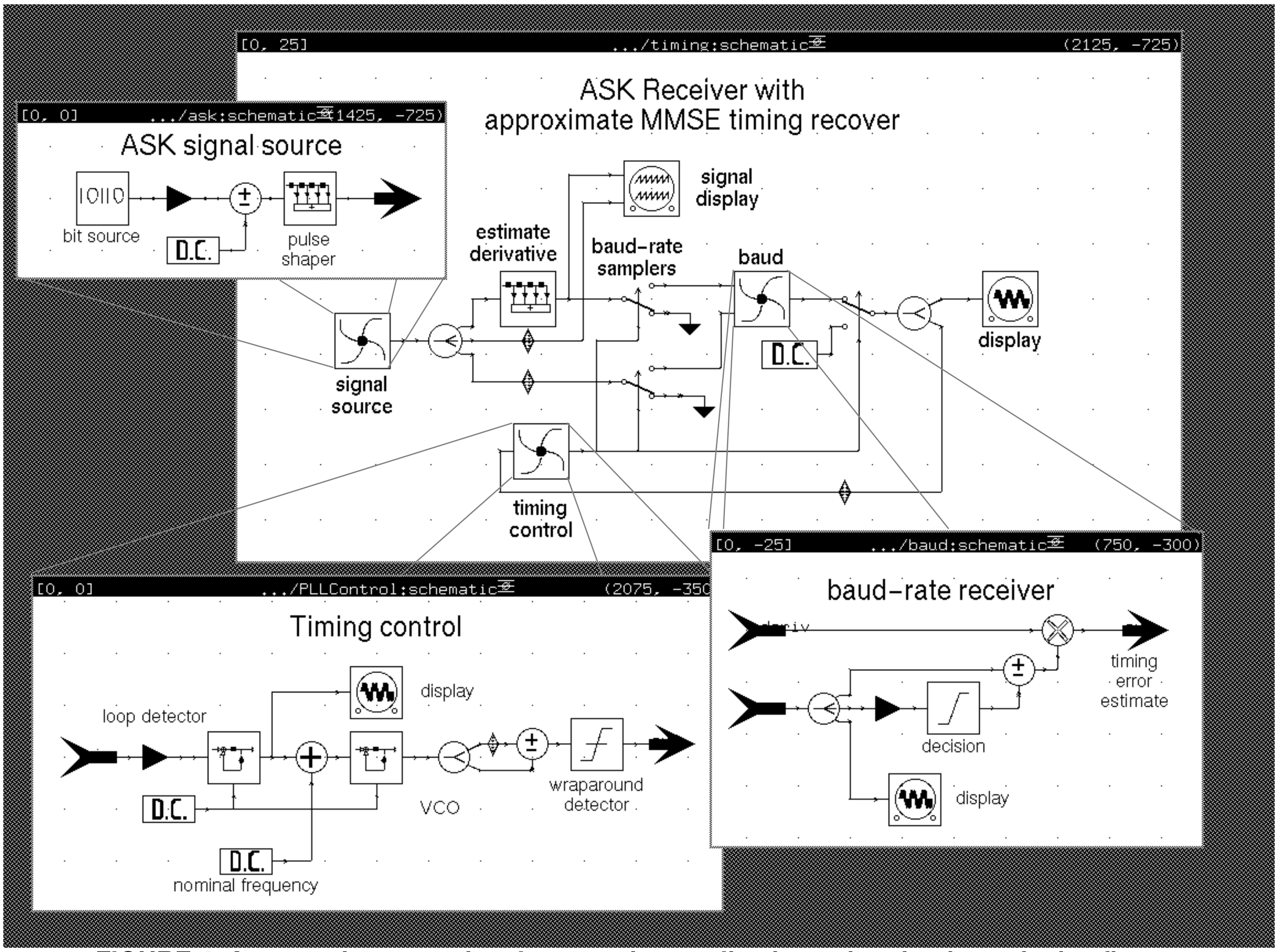

FIGURE 4. An asynchronous signal processing application using the dynamic dataflow domain in Ptolemy.

have also developed a domain that generates Silage code for coupling into hardware synthesis tools developed at Berkeley and at Mentor Graphics, and two domains that generate VHDL for hardware simulation and synthesis.

Most of these code generators operate under either the SDF or the BDF models of computation. More detail on the basic techniques is given in [14].

A.7 Control Software

Control software requires a much different paradigm than signal processing and networking. For example, we typically want to run the same control software within the Ptolemy environment as will later be used in the actual system. Furthermore, it is often the case that the control software does not require a timing model (it operates instantaneously with respect to a simulation).

Our efforts have focused on call processing and signaling in communication networks. The message queue supports call processing and signaling software; that is, event-driven systems. The MQ domain is similar to the DE domain in that it is event-driven, and signals consist of events carrying time stamps. However, signals travel along arcs that are dynamically created and destroyed during the run of the simulation. This domain is not as amenable to graphical representation as some of the others.

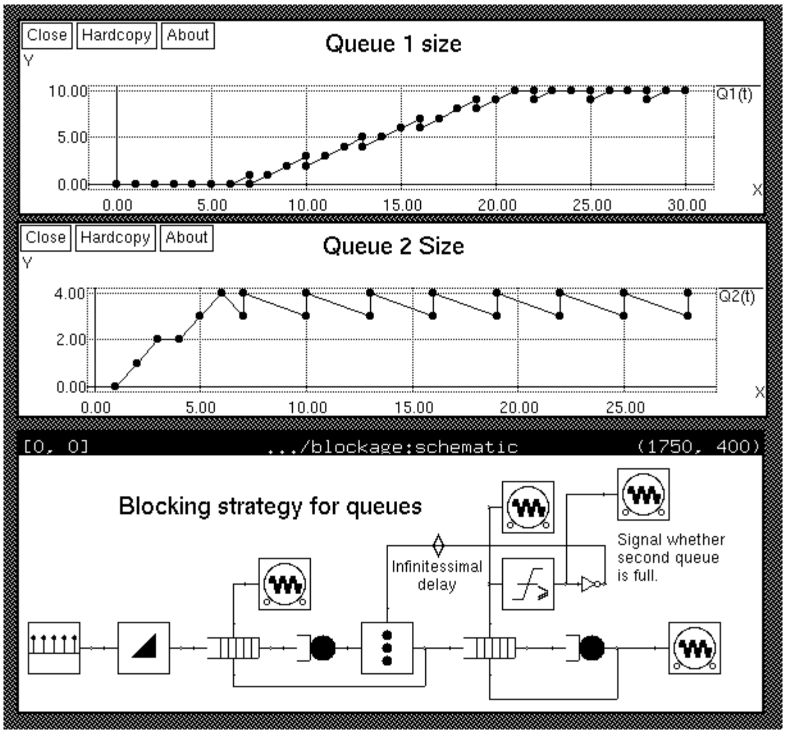

FIGURE 5. A queueing system implemented in the discrete-event domain in Ptolemy.

A separate effort is pursuing the integration of a hierarchical state machine domain. The interoperability of this domain with others could prove extremely useful, especially for developing embedded software.

A.8 System-Level Design

Ptolemy is equally useful for high-level modeling of hardware systems, where contention for shared resources, or asynchronous behavior is of interest. For example, in figure 8, the DE domain is used to simulate four clients randomly accessing a shared memory through an arbitration network with queueing. In the figure, the arbitration network grants access to the clients according to round-robin protocol. When a client gains access, it services all access

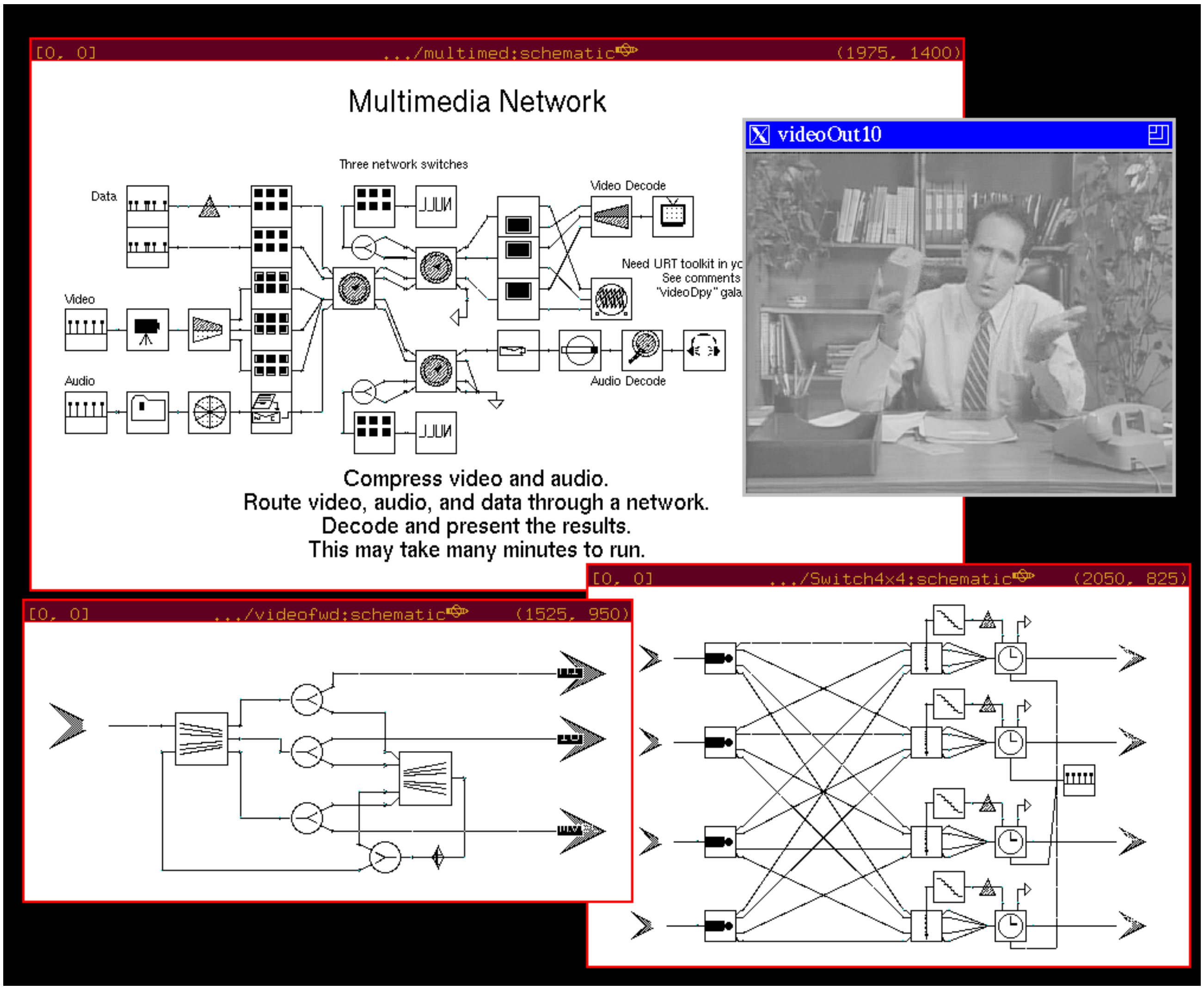

FIGURE 6. A multimedia networking application that combines discrete-event modeling of an ATM (asynchronous transfer mode) packet-switched network with video and audio coding and decoding (application developed by Paul Haskell).

requests that have been queued and then releases the bus. This kind of simulation can be used, for example, to measure latency in accesses, and to determine probabilities of queue overflow.

A.9 Hardware Design

The "Thor" domain in Ptolemy, named after the register-transfer level hardware simulator from Stanford, simulates digital logic. A simple example is shown in figure 9. This domain is similar to the DE domain in that it is event-driven, and signals consist of events carrying time stamps. However, the scheduler is optimized for simulating hardware. This domain has been developed just enough to allow experimentation with hardware/software codesign. It has not been sufficiently developed to compete with commercial RTL circuit simulators.

A.10 Communicating Processes

An experimental domain is the communicating processes (CP) domain, in which blocks are separate processes that run under the Sun lightweight process scheduler. This approach is

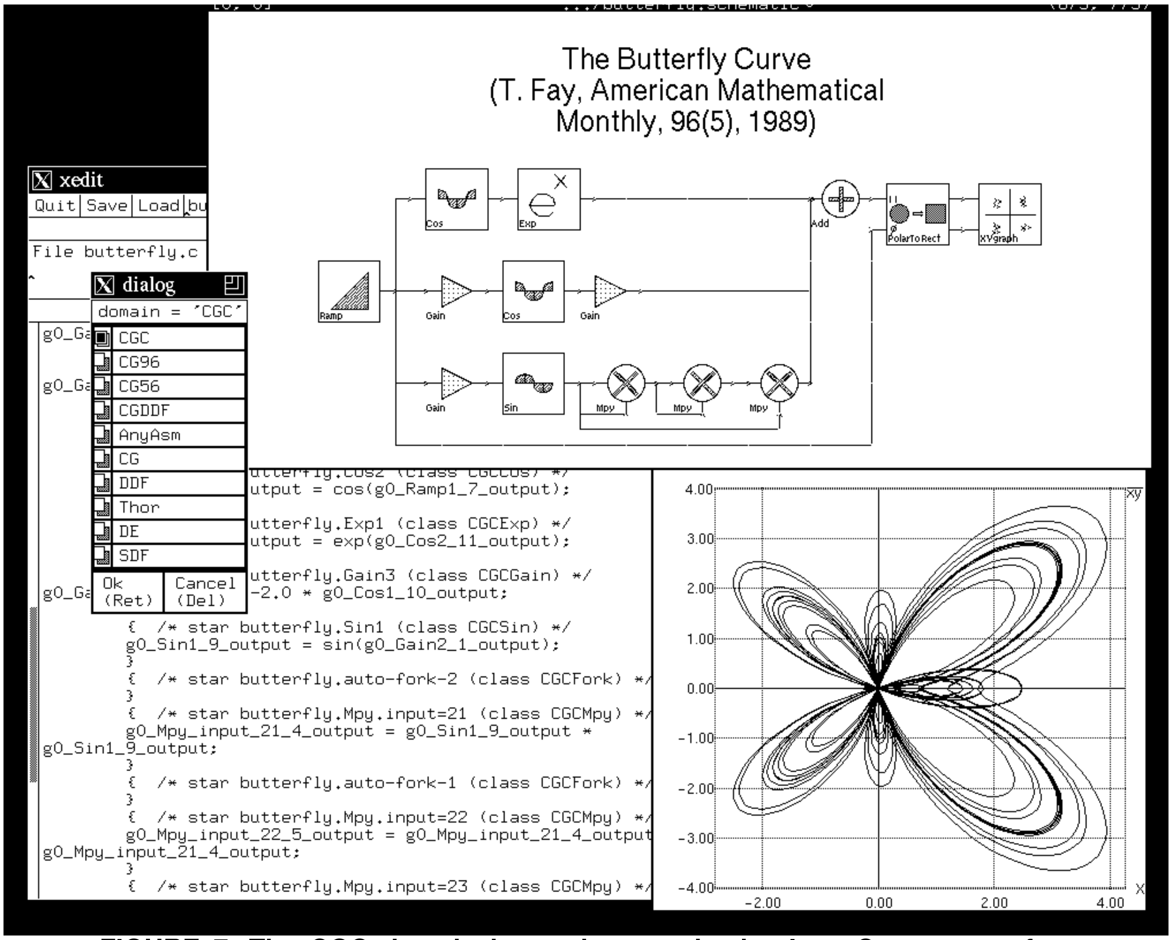

FIGURE 7. The CGC domain is used to synthesize lean C programs from synchronous dataflow graphs.

sometimes much more convenient than DE for high-level system modeling. However, its dependence on the Sun lightweight process library makes it platform specific.

B. Formal Methods

A significant part of the Ptolemy project is concerned with understanding models of computation and the interaction between diverse models of computation. This work dates back to our development of the synchronous dataflow model of computation [8]. More recent develop-

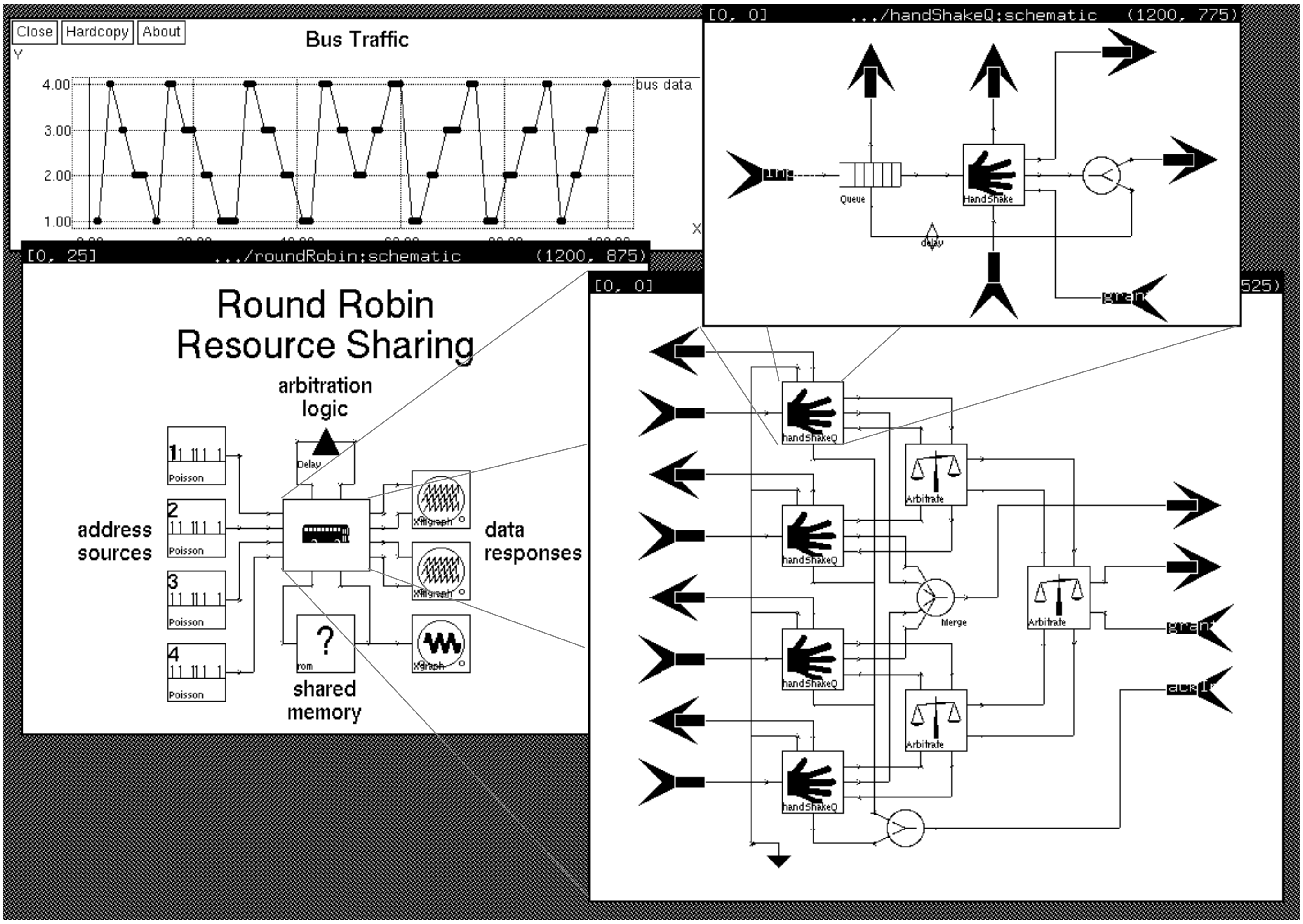

FIGURE 8. An illustration of the use of the DE domain in Ptolemy for system-level modeling of hardware. At the lower left, four clients demand access to a shared memory according to a Poisson process. The access is regulated by an arbitration network, implemented in the subsystems at the right. These allocate access according to a round-robin protocol, resulting in the regular bus traffic shown at the upper left (application developed by Tom Parks).

ments include the Boolean dataflow model of computation [3][9], a multidimensional dataflow model [10], and a formal approach to optimal scheduling problems [1][2].

Formal methods can guarantee deadlock-free operation, bounded data memory, compact program representation, and real-time response, and can enable synthesis of software and hardware as well as partitioning and scheduling. This promise has only been realized when the specification languages are small and have simple semantics. The heterogeneity supported by Ptolemy makes such specification languages practical and broadly useful. We have been applying such methods to signal processing systems.

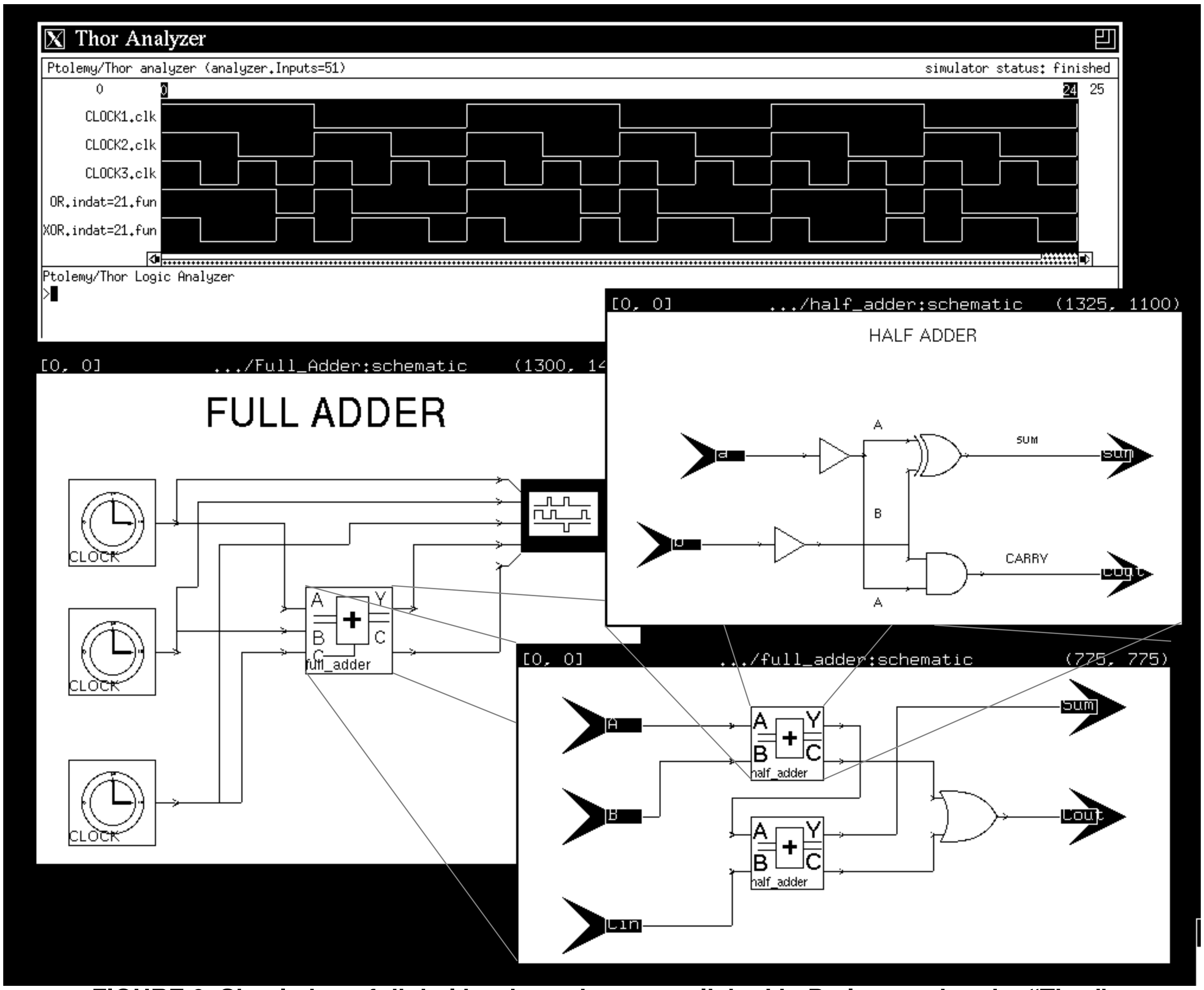

FIGURE 9. Simulation of digital hardware is accomplished in Ptolemy using the "Thor" domain. The system shown implements a one-bit full adder out of elementary gates and tests it with three clocks with different periods.

C. Multi-Paradigm Computing

Ptolemy is unique in that it supports heterogeneity in simulation and modelling, and hence can be used to build new, application-specific design environments, and to combine these environments with others designed under Ptolemy. Heterogeneous simulation environments or programs built using Ptolemy as a foundation can subsequently be merged together to design large, complex, and heterogeneous applications.

C.1 Communication Networks with Signal Processing

Until recently, communication networks have been segregated according to media format: data, voice, images, or video. This segregation is unnatural, but technologically convenient. Designing networks that support an integration of these formats is difficult. The latency and throughput requirements vary widely, as does the tolerance of errors. The opportunities for data compression, and technologies required to accomplish the data compression, also vary widely. The design of a packet-switched network capable of reliably and efficiently transmitting data, speech, images, and video, requires consideration of all these issues. For example, to carry real-time speech, an encoding algorithm that tolerates lost or delayed packets may be required. The encoder may also be required to monitor the network for congestion, and adapt its compression ratio accordingly. A video transmission network may need to perform video compositing within the network, rather than just at the terminals.

Such multimedia designs are fundamentally heterogenous. Network design, which is naturally done with the DE domain in Ptolemy, must be combined with signal processing design, which is most naturally accomplished in the SDF or DDF domains.

A simplified example that combines the SDF and DE domains is shown in figure 10. In this example, the lower diagram models a highly simplified packet-switched communication network, in which packets randomly traverse one of two paths. The upper path has no delay. The lower path has random delay. At the receiving end, a queue stores incoming packets until they are needed by the destination equipment. Above this network model is a highly simplified signal processing system making use of the network. A sinusoid is generated and packetized, one sample per packet, and launched into the network. At the receiving end, packets are used to reconstruct the sinusoid at the same sample rate. The packets are used in the order of arrival, so the samples of the sinusoid get randomly scrambled, as shown in the upper plot. Real-time constraints are modeled, so if packets do not arrive in time, earlier packets are re-used.

In practical applications with similar structure, both the network model and the signal processing will be much more elaborate. Ptolemy is currently being used to evaluate video encoding algorithms for transmission over ATM (asynchronous transfer mode) networks, as shown in figure 6.

C.2 Hardware/Software Codesign

Most electronic systems mix custom circuit designs with programmable commodity parts. Ptolemy supports such designs as a unit, since using the various domains described above, all parts of the system can be modeled. For example, the Thor domain can be combined with a code generation domain to design boards that mix custom hardware with programmable DSPs. Since the hardware and software are both modeled within the same software framework, a designer can easily explore tradeoffs between hardware and software implementations of various functions.

An example of such a design is shown in figure 11. The top window shows the top level only of an algorithm that simulates the impairments of a telephone channel. This algorithm is fairly complicated, including linear and non-linear distortion, frequency offset, phase jitter, and additive Gaussian noise. The design is built in a code generation domain compatible with the SDF model of computation. The bottom window shows a hardware design containing two programmable DSPs communicating through a dual ported shared memory. This hardware might be used to implement the telephone channel simulator for production-line testing of voiceband data modems.

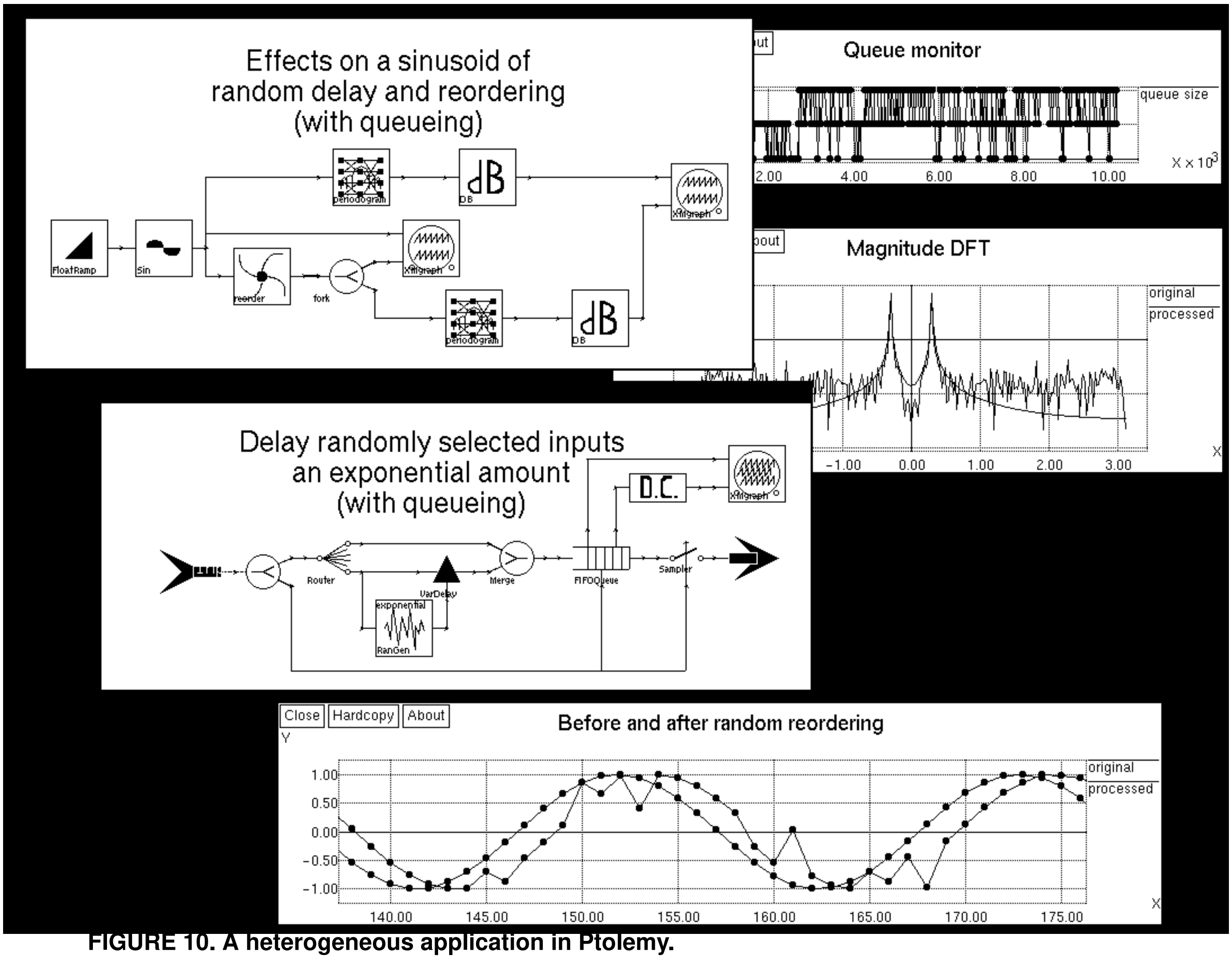

FIGURE 10. A heterogeneous application in Ptolemy.

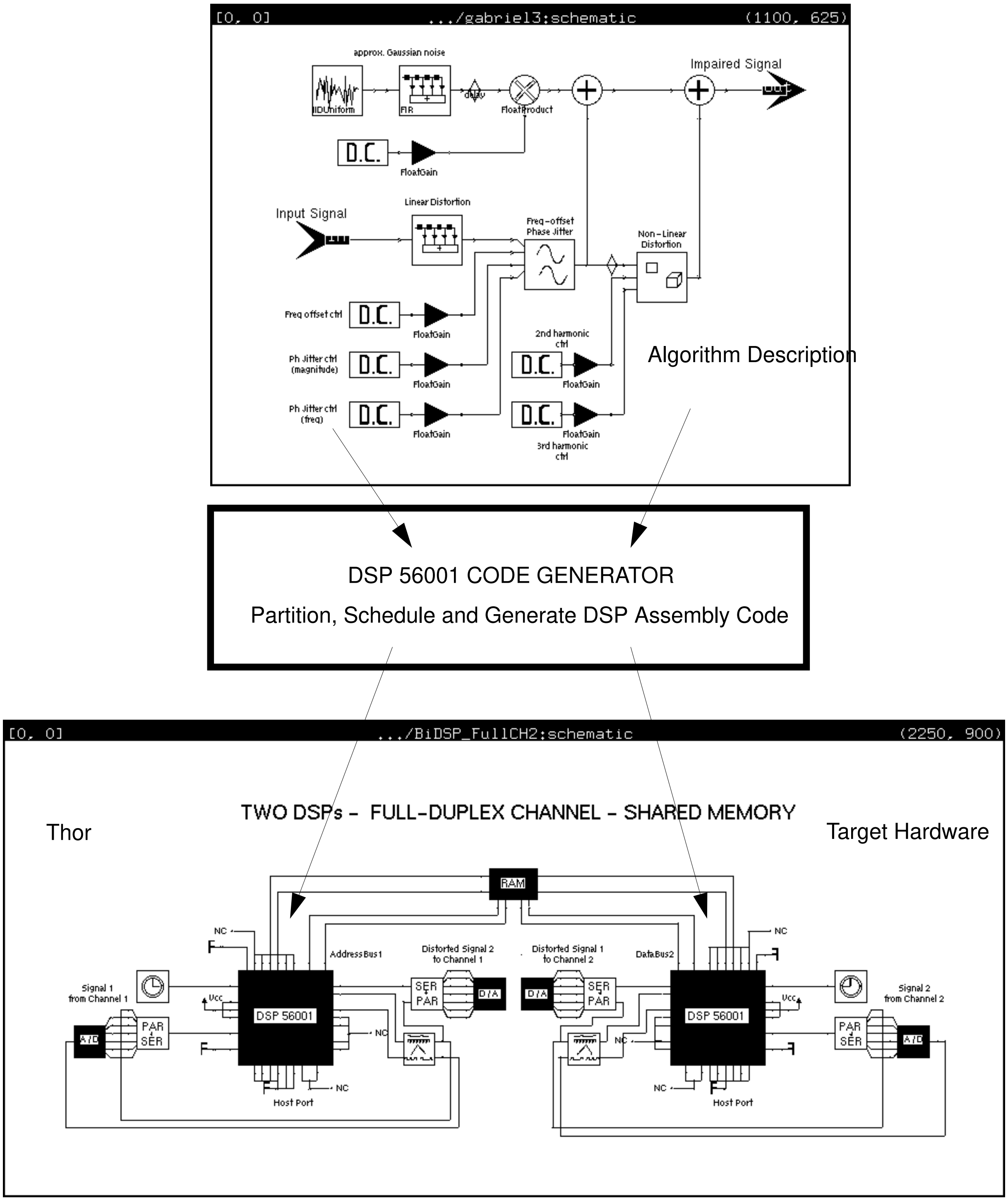

FIGURE 11. A hardware design (bottom) containing programmable DSPs can be developed together with the software (top) that will run on the DSPs. This figure shows the top level only of a telephone channel simulation algorithm (top window) being mapped onto a board design with two Motorola DSP56001 DSPs.

The design shown in figure 11 is one of many that could accomplish the stated objectives. Using Ptolemy, the entire design, ranging from algorithm development to circuit design, can be carried out within a unified environment. This enables exploration of many design alternatives before resources are committed to hardware prototyping. Further work in this direction has included coupling with high-level hardware synthesis tools and exploration of design-flow management. This work is described in [7].

A second project in this area is concerned with the interconnect and synchronization of multiple processors cooperating on a single task in an embedded system. A very lean interconnect methodology has been developed that is compatible with the parallel scheduling techniques described below and the code generation techniques described above. In this methodology, processors communicate using the so-called *ordered-transaction principle* [17].

C.3 Multi-Platform Implementations

The interface between domains in Ptolemy is being generalized to permit different parts of a Ptolemy system to run on distinct hardware platforms. A simple form of this distributes Ptolemy simulations among multiple Unix workstations. A more elaborate form generates code for non-Unix machines, in C or assembly language, downloads the code, and manages the communication between system components transparently.

A practical application for the latter capability is the design of a multimedia workstation, where programmable hardware (FPGAs) might be used to interface to a variety of audio and display devices, one or more programmable DSPs might be used to manage encoding and decoding of real-time signals, a processor running a real-time O/S might handle all real-time control, and a processor running Unix might manage the interaction with the user. Ptolemy would unify this highly heterogeneous environment. We currently have early prototypes that unify a workstation CPU with one or more attached programmable DSPs.

D. Parallel Programming using Ptolemy

Parallel programs are not so much “written” as “designed”. Writing is sequential, left-to-right, top-to-bottom, while design builds relationships between parts. Even with ordinary structured and object-oriented programming, the writing metaphor is weak. It requires augmentation with specialized text editors and hypertext facilities such as class browsers and tag managers. But the breakdown of the metaphor is complete with parallel programming.

A “design” makes a whole from parts without any implication of sequentialism. In parallel programs, the parts may be processing elements or concurrent processes, but far better, they should be parts of the application being designed. The interaction between these parts, and between these parts and a user, is after all what concerns the designer (the “programmer”).

D.1 Requirements of parallel programming

Parallel programming requires visualizing the parallelism in an application. This need not be any more unnatural than visualizing parallelism in, say, a circuit schematic. Half-hearted enhancements to traditional languages, like "for-all" constructs, hardly satisfy the need, since they only hint to a compiler of some inner-loop parallelism that might be exploited. Communicating sequential processes, such as those supported by the Occam language, in principle allow more flexibility, but they leave too much of the scheduling work to the programmer. Some attempts to automatically exploit parallelism have met limited success either because they started with sequential languages, or because they were too specialized. Specialized techniques, such as automated mapping onto systolic arrays, can be effective for specific algorithms, but few if any applications consist entirely of a single algorithm. To be useful, these techniques have to be combined with others into an environment that achieves generality through a suite of specialized methods.

Parallel programming also requires fundamentally new compiler and debugger technology where the focus is less on code generation and more on scheduling. By "scheduling", we mean partitioning, mapping, ordering, and timing of tasks on processors. It also includes routing of communications. The compiler should assume responsibility for as many of these as possible, but with hooks for the programmer to tune the schedule if desired. Dataflow machines have shown that it is not reasonable in most cases for run-time hardware to assume such responsibility. And the limited penetration of today's commercial parallel machines has shown that programmers are not willing to assume such responsibility themselves.

D.2 The role of graphics in parallel programming

Compilation of parallel programs and programmer visualization of parallelism both demand the same missing ingredient: elegant representations of programs that are compatible with parallel computing. This need not mean new languages, but it almost certainly means new program structuring. Our experience indicates that combinations of concise, textual representations of arithmetic and sequential operations with graphical representations of higher-level program structure can be effective, at least in the domain of signal processing.

D.3 Current Status of Parallel Programming Using Ptolemy

Three parallel schedulers have been built into a generic domain in Ptolemy [15][16][5][6]. By "generic domain" we mean a domain that form a base class for specific code generation domains. Hence, the schedulers can be used for a variety of targets without any changes. An example program in such a domain is shown in figure 12. This domain consists only of base classes to be used in actual code generation domains, and hence does not actually run any application. Instead of functional blocks, it has a suite of generic blocks, each with a parameter that specifies its execution time. Furthermore, the target is parameterized so that we can modify the interprocessor communication times and test the efficacy of the scheduler. The value of each parameter used to generate figure 12 is given below:

- Runtimes of the blocks: ranging from one time unit to five time units.
- Time to send one data packet: one time unit.

- Time to receive one data packet: one time unit.
- Communication interferes with processing (no separate communication unit).
- Number of processors: 4
- Processors contend for communication resources.

A Gantt chart is shown below the block diagram. A vertical line in the Gantt chart causes the blocks that fire at that time to be highlighted. Note that there is only sufficient concurrency in this test program to use three of the four processors, so the scheduler does not make use of the last processor.

All of the parameters can be modified, and the scheduler will adapt. For instance, if the communication time is increased from one time unit to 20, say, we find that the scheduler maps the entire application onto a single processor. Communication is too expensive to justify parallelizing such a simple application.

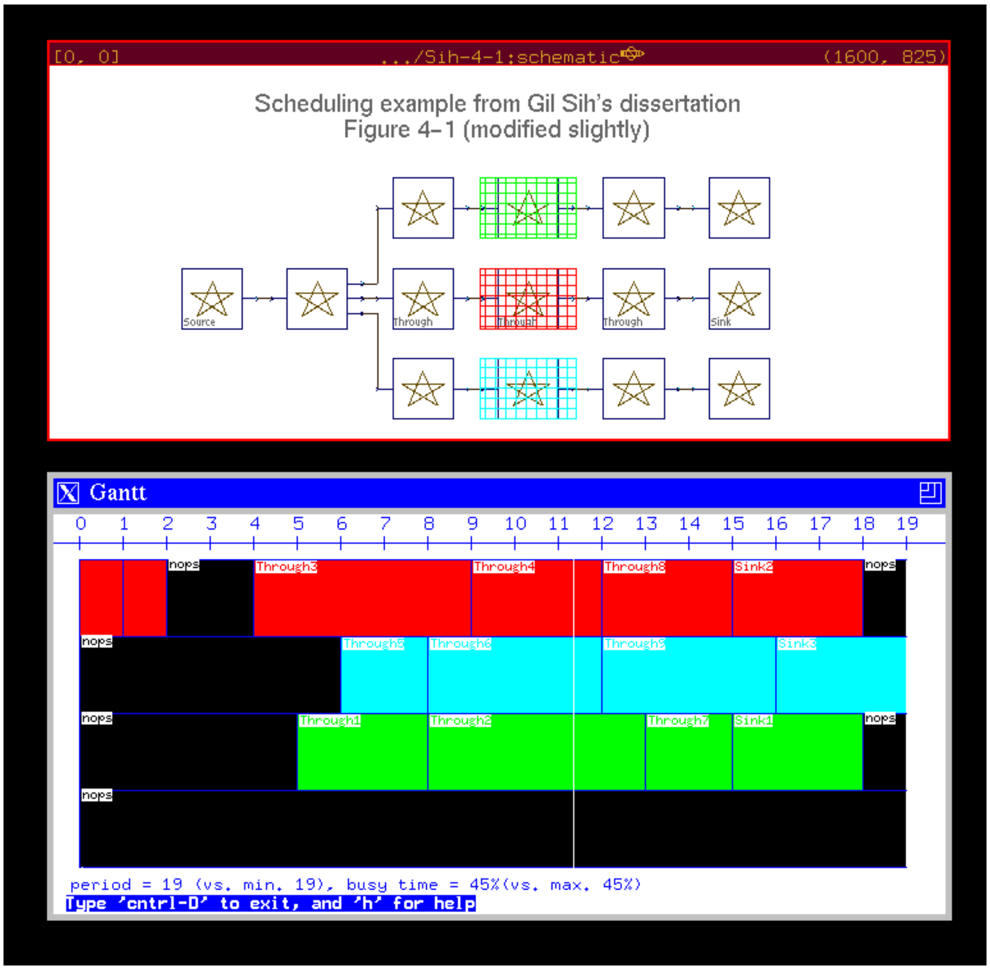

FIGURE 12. A program in a generic code generation domain, and a parallel schedule generated for that program.

The first two schedulers support only the SDF model of computation. A third scheduler has been developed that supports the DDF model of computation. All of these schedulers will be compatible with any code generation domain compatible with these models of computation. Hence, they can be used to generate parallel programs in any language.

E. Representative Applications

Ptolemy is currently be used for a number of applications, at Berkeley and elsewhere. We will briefly describe some of the current activity at Berkeley.

E.1 Signal Processing

A wide variety of signal processing applications have been developed using Ptolemy, including several adaptive filtering applications, power spectrum estimation, several parametric and waveform coding techniques based on linear prediction, communication channel modeling, digital communication receivers, beamforming, digital filter design, chaos simulations, phase-locked loops, image coding, and music synthesis. Many of these applications are distributed as with the Ptolemy code as demonstrations. Ptolemy is also being used to teach both graduate statistical signal processing and undergraduate digital signal processing.

E.2 Wireless Networking

The wireless work is collaborative with a number of other faculty, and is attempting to achieve high speed wireless access to the network. We are using Ptolemy as a basis for the fading channel, modulation, PLL simulations. An important capability of Ptolemy is to model at the timing and circuit simulation levels, so we can study interactions between circuit designs (preamps etc.) and other parts of the system.

E.3 Cell-Relay Networks

To demonstrate a large heterogeneous simulation, we have constructed a full cell-relay (ATM) network. While it is simplified in some respects, it includes the three key elements: cell-relay transport (DE domain), call processing (MQ domain), and video codec (SDF domain). CPE's set up random calls through the network, which are routed by the call processing, and signal processing based CPE's transmit packets through the network. The 2x2 switching elements in the Batcher-Banyon switches are modeled in SDF, including their routing tables, and the interface of these routing tables to the MQ call processing was an interesting challenge.

Currently another project is focusing on the buffering and associated control at the interface between the synchronous signal processing world and the asynchronous transport: the cell assembly operation. As a design exercise, we are carrying this design down to the board level, utilizing the DE and Thor domains of Ptolemy. What is interesting here is the ability to model the interaction of this board with the network (congestion control) and signal processing (flow control) in the context of the hardware model.

E.4 Video Signal Processing

We are developing models in Ptolemy for standard video signal processing algorithms (DCT etc.). Later this will be merged with the networking simulations (cell relay) to study issues in the interaction between transport and signal processing for multipoint and interactive video services. For example, the compression must be designed taking into account the cell-loss mechanisms, and also must accept flow-control directives from the network.

E.5 Power Networks

Prof. Wu at Berkeley is using Ptolemy for the modeling of the communication and control infrastructure of an electric power network. Theirs was actually the first large-scale communication system modeling effort in Ptolemy. Subsequently they plan to model the power grid itself, and then connect these two heterogenous large-scale systems.

F. Further information

The Ptolemy software, plus a number of papers and other documents, including this one, are available at our ftp site, ptolemy.eecs.berkeley.edu. One of the documents available at that site is a complete list of papers that have been produced by the Ptolemy group, including many that, for various reasons, are not available by ftp.

G. References

[1] S. Bhattacharyya and E. A. Lee, “Memory Management for Synchronous Dataflow Programs,” to appear in *IEEE Tr. on Signal Processing*, May 1994. Updated from: Technical Report UCB/ERL M92/128, EECS Dept., UC Berkeley, November 18, 1992.

[2] S. Bhattacharyya and E. A. Lee, “Scheduling Synchronous Dataflow Graphs for Efficient Looping,” **to appear** in *J. of VLSI Signal Processing,* 1993.

[3] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, “Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems”, *International Journal of Computer Simulation*, special issue on “Simulation Software Development,” January, 1994.

[4] J. T. Buck, *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model,* Tech. Report UCB/ERL 93/69, Ph. D. Dissertation, Dept. of EECS, University of California, Berkeley, CA 94720, 1993.

[5] Soonhoi Ha and E.A. Lee, “Compile-Time Scheduling and Assignment of Dataflow Program Graphs with Data-Dependent Iteration,” *IEEE Transactions on Computers*, November, 1991.

[6] S. Ha, “Compile-Time Scheduling of Dataflow Program Graphs with Dynamic Constructs,” **Ph.D. Dissertation**, EECS Dept., University of California, Berkeley, CA 94720, April 1992.

[7] A. Kalavade, and E.A. Lee, “A Hardware/Software Codesign Methodology for DSP Applications,” *IEEE Design and Test*, September 1993.

[8] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow,” *IEEE Proceedings*, September, 1987.

[9] E. A. Lee, “Consistency in Dataflow Graphs”, *IEEE Transactions on Parallel and Distributed Systems”,* Vol. 2, No. 2, April 1991.

[10] E. A. Lee, “Multidimensional Streams Rooted in Dataflow”, *Proc. IFIP WG10.3 Working Conference on Architectures and Compilation Techniques for Fine and Medium-Grain Parallelism* (Orlando, FL, USA January, 1993), North-Holland, New York, 1993.

[11] J. K. Ousterhout, "Tcl: An Embeddable Command Language," USENIX Conference Proceedings, Winter, 1990.

[12] J. K. Ousterhout, "An X11 Toolkit Bassed on the Tcl Language," USENIX Conference Proceedings, Winter, 1991.

[13] J. K. Ousterhout, *An Introduction to Tcl and Tk,* Addison-Wesley Publishing Co., 1994.

[14] J. Pino, S. Ha, E. Lee, J. Buck, “Software Synthesis for DSP Using Ptolemy”, invited paper in the *Journal on VLSI Signal Processing*, special issue on “Synthesis for DSP”, **to appear**, 1993.

[15] G. C. Sih and E. A. Lee, “Declustering: A New Multiprocessor Scheduling Technique,” *IEEE Trans. on Parallel and Distributed Systems,* Vol. 4, No. 6, pp. 625-637, June 1993.

[16] G. C. Sih and E.A. Lee, “A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures”, *IEEE Trans. on Parallel and Distributed Systems,* Vol. 4, No. 2, February, 1993.

[17] S. Sriram and E. A. Lee, “Design and Implementation of an Ordered Memory Access Architecture,” *Proc. of ICASSP* ‘93, Minneapolis, MN, April, 1993.