DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

ANNUAL REPORT

HETEROGENEOUS MODELING AND DESIGN

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY (DARPA)
COMPOSITE CAD PROGRAM

CONTRACTOR:	**University of California at Berkeley**
AGREEMENT NUMBER:	**DAAB07-97-C-J007**
CONTRACT PERIOD:	**11/18/96 - 11/31/99**
DATE:	**December 1, 1997**
TITLE:	**Heterogeneous Modeling And Design**
REPORT PERIOD:	**11/18/96 - 11/15/97**
SPONSOR:	**Air Force Research Laboratory (AFRL)**
TECHNICAL POC:	**James P. Hanna**
REPORT PREPARED BY:	**Edward A. Lee**

Principal Investigator: Edward A. Lee
Organization: University of California at Berkeley

Contents

1. Project Overview

The Heterogeneous Modeling and Design (HMAD) project is a 2 phase, 36 month, Defense Advanced Research Project Agency (DARPA) sponsored program to develop a design methodology, and associated modeling software, for composite, heterogeneous systems. Such systems combine diverse implementation technologies, including microelectromechanical systems (MEMS), microwave circuits, analog circuits, digital circuits, and embedded software. They also combine modeling and design paradigms, including physical modeling using differential equations, continuous-time signal processing, discrete-time signal processing, and discrete-event controllers. They are invariably concurrent, involving diverse modules that operate at the same time and interact through continuous signals or discrete messages. The focus of the project is on the theory and technology of heterogeneous modeling of heterogeneous concurrent systems.

Formerly entitled Distributed Adaptive Signal Processing (DASP), this project focuses on fundamental modeling and design techniques that are not specific to signal processing. Phase 1 is an 18 month effort to develop the supporting infrastructure. The infrastructure consists of modular design tools, domain-specific design techniques, and models for the interaction of dynamic, discrete controllers with static digital and analog subsystems. Phase 2 is an 18 month effort to develop a process level type system theory capturing different kinds of interaction of concurrent modules, and concurrency semantics for regulation of heterogeneous combinations of such interaction semantics. Phase 2 will also focus on formal analysis, debugging, and system level visualization for heterogeneous concurrent systems.

This project is part of a long term effort at Berkeley called the Ptolemy project that has been studying various aspects of design methodology. The Ptolemy project has an established track record of high-impact contributions in methodology, theory, and software, with demonstrable transfer of the resulting technology to industry. The underlying principle of the project, which we expand on below, is to embrace heterogeneity. The Composite CAD program at DARPA appears to be an excellent candidate for targeted heterogeneous methodologies.

2. Heterogeneous Design Principles

This project is about heterogeneous system-level and implementation-level descriptions of systems and the relationships between these two levels. This relationship is generally a combination of modeling and specification. That is, components at the implementation level may be modeled or specified at the system level. Most interesting designs will have some of both.

The heterogeneity at the two levels and the multi-way relationships between levels augment the complexity of the design process and the difficulty that designers have with such designs. A major part of our approach to managing this design complexity is by exploiting visual syntaxes such as block diagrams and bubble-and-arc diagrams. In this project, we are working on such visual syntaxes, their underlying semantics, and the software architecture that supports the use of several such syntaxes in combination.

2.1 BLOCK DIAGRAMS

Visual depictions of electronic systems have always held a strong human appeal, making them extremely effective in conveying information about a design. Many of the domains of interest in this project use such depictions to completely and formally specify systems, most notably in circuit design,

where schematic diagrams can capture all of the essential information needed to implement some systems. Others have failed dramatically, for example flowcharts for capturing the behavior of software. Recently, a number of innovative visual formalisms have been garnering support, including visual dataflow, hierarchical concurrent finite state machines, and object models.

The subset of these that are recognizable as “block diagrams” represent concurrent systems. There are many possible concurrency semantics associated with such diagrams. Formalizing the semantics is essential if these diagrams are to be used for system specification and design. This project is exploring some of the possible concurrency semantics, with the premise that their strengths and weaknesses make them complementary rather than competitive, so that no single model is likely to emerge as a universally useful model.

2.2 SYSTEM-LEVEL DESIGN

By “system-level design” we mean design at the problem level that is relatively unencumbered by implementation issues. Components are represented in terms of their functionality or role in a design rather than in terms of their construction. A typical system-level model of a system, implemented in Ptolemy, is shown in figure 1.

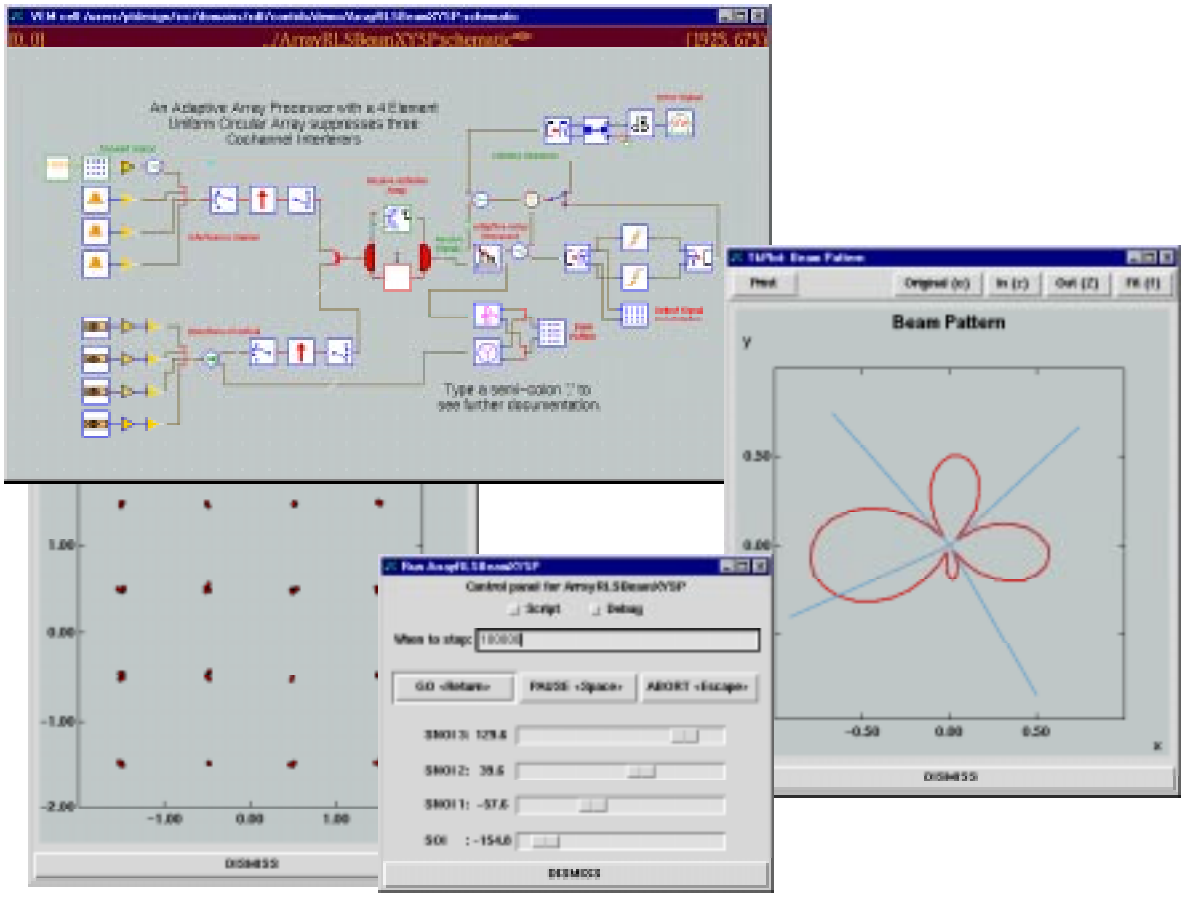

FIGURE 1. A system-level model (developed in Ptolemy by Uwe Trautwein of the Technical University of Ilmenau, Germany) of a beamformer for wireless communications that adaptively nulls interferers.

Such specifications are *modular*, in that large designs are composed of smaller designs, and these smaller designs encapsulate specialized expertise. They are *hierarchical*, in that composite designs themselves become modules, and modules may be very complicated. They are *concurrent*, in that modules logically operate simultaneously. Implementations may be sequential, parallel, or distributed. They are *abstract*, in that the interaction of modules occurs within a model of computation. They are *domain specific*, tuned in this case to the needs of signal processing applications. Often they will need to combine multiple domain-specific subsystems.

Some of the modules in a system-level description will be models of physical components (such as antennas and microwave circuits in the example in figure 1). Others will be specifications of functionality without particular reference to the implementation technology. They could be implemented by any physical realization capable of delivering the specified functionality, such as microelectromechanical systems (MEMS), analog circuits, digital circuits, or embedded software. To be successful, therefore, system-level design must be coupled with high quality synthesis tools that translate system-level specifications into implementations.

The example in figure 1 uses a dataflow model of computation, a particularly convenient and popular means for specification of signal processing systems. However, that example fails to include many aspects of a true system-level model of a wireless communication system. It focuses only on the adaptive nulling, and omits the control logic associated with, for example, multiple access to the radio medium or call processing. The latter aspects of the design are a poor match to dataflow modeling, so alternative models of computation must also be included.

2.3 HETEROGENEOUS IMPLEMENTATIONS

Embedded systems today are typically implemented using a combination of implementation technologies, as suggested in figure 2. Custom digital hardware, for example, may be combined with analog, microwave, or MEMS designs. Hard real-time software, written in assembly code for a specialized processor like a programmable DSP, may be combined with higher-level software, typically written in C, that implements the control logic of the application. And of course, hardware and software are combined within the same design.

2.4 HETEROGENEOUS MODELING AND DESIGN

Typical embedded systems today will exhibit heterogeneity both at the problem level and at the implementation level. Two competing approaches to the design of such systems are the *grand unified approach* and the *heterogeneous approach*. The grand unified approach seeks to find a common representation language for all components, and to develop techniques to synthesize diverse implementations from this representation. The heterogeneous approach uses domain-specific models of computation hierarchically mixed and matched to define a system and seeks to find retargettable synthesis techniques from specifications to diverse implementation technologies. This project is pursuing the latter approach.

The heterogeneous approach has a number of advantages. First and foremost, it is clearly possible, while there is no clearly usable grand unified approach. In addition, it emphasizes domain specific techniques, which match the applications better. Furthermore, because they are more specialized, domain-specific techniques are more amenable to high-level synthesis.

Any particular (known) candidate for a grand unified approach has a number of serious disadvantages. First, it must, of necessity, impose a model of computation. For example, choosing to use an imperative language will impose a sequential model of computation. But any particular model of computation can greatly affect the chosen system architecture. Using an imperative language, for instance,

will strongly bias implementations towards software over hardware. On the other hand, using a discrete-event model of computation, as with structural VHDL, will strongly bias the implementation towards hardware over software. If a grand unified approach fails to impose a model of computation, then it will have all of the disadvantages of the heterogeneous approach and none of the advantages.

In the heterogeneous approach, multiple models of computation may be used at the problem level (figure 1) and the implementation level (figure 2). The core of the project, therefore, is on the relationship between heterogeneous models at these two levels, as suggested in figure 3. This relationship consists of a modeling relationship (where a problem-level description is a model of an implementation), and specification (where a problem-level description is translated into an implementation-level description).

2.5 MODELS OF COMPUTATION

There are a rich variety of models of computation that deal with concurrency in different ways. In this section, we outline some of the most useful models in the Composite CAD domain. All of these will lend an interpretation, or *semantics*, to the same bubble-and-arc, or block-and-arrow diagram shown in figure 4.

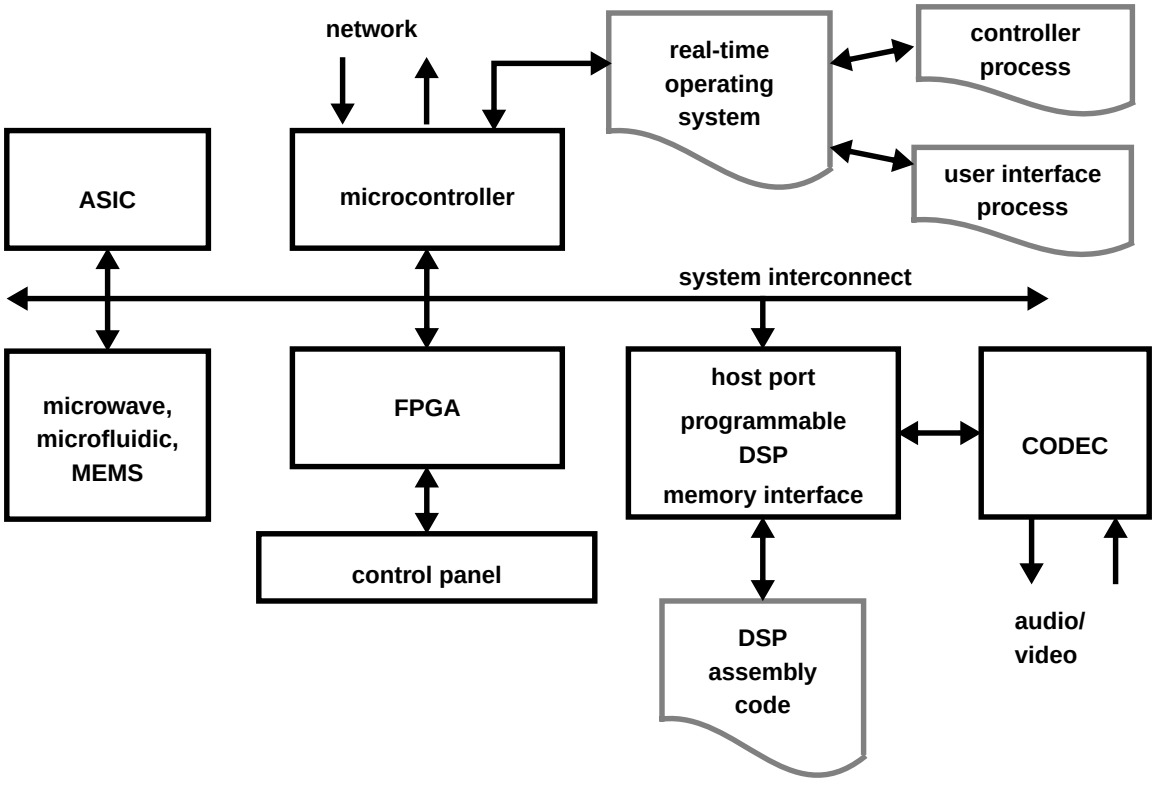

FIGURE 2. Typical hardware architecture for an embedded system. The architecture is highly heterogeneous, and its hardware-software combination is only one of several manifestations of this.

2.5.1 *Differential equations*

One possible semantics for the syntax in figure 4 is that of differential equations. The arcs represent continuous functions of a continuum that is interpreted as time. The bubbles represent relations between these functions. The job of a simulator is to find a fixed-point, i.e., a set of functions that satisfy all the relations.

Differential equations are excellent for modeling analog circuits and many physical systems. This is the model of computation used in Spice circuit simulators. However, they have disadvantages. Since

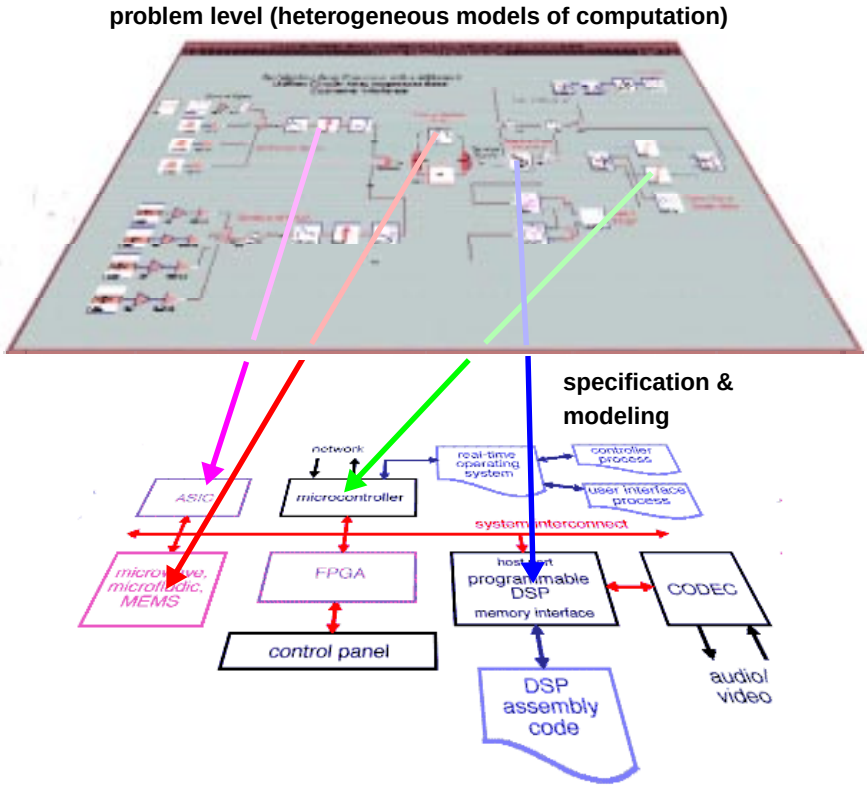

FIGURE 3. The focus of this project is on heterogeneous problem-level modeling, heterogenous implementation-level modeling, and the relationships between these levels.

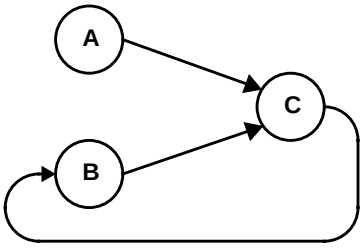

FIGURE 4. A single *syntax* (bubble-and-arc or block-and-arrow diagram) can have a number of possible *semantics* (interpretations).

they directly describe a physical system, they are tightly bound to an implementation, leaving few implementation options. Moreover, they are only applicable to relatively well-understood technologies, where lumped-parameter modeling is appropriate. They must be generalized to partial differential equations for less understood technologies, where solution techniques such as finite elements can be quite costly. For well-understood technologies, they can be expensive to simulate compared to more digital representations of comparable functionality (and hence, they can be expensive to implement in software). Thus, differential equations are best used for implementation-level modeling.

Although at Berkeley we have not yet created the ability to use differential equations in Ptolemy, Hewlett-Packard has recently announced an integration of Ptolemy with their well regarded analog and microwave circuit simulators. For more details, see Section 7 “Technology Transfer” on page 34. We are pursuing many of the fundamental issues associated with such composite modeling under this project.

2.5.2 Difference equations

Differential equations can be discretized to get difference equations, a commonly used model of computation in digital signal processing. This model of computation can be further generalized to support multirate difference equations. In either case, a global *clock* defines the discrete points at which signals have values (at the *ticks*).

Difference equations are considerably easier to implement in software, and hence leave more freedom of implementation. Thus, they can be used at the problem level. Their key weaknesses are the global synchronization implied by the clock, and the awkwardness of specifying irregularly timed events and control logic.

The *synchronous dataflow* domain in Ptolemy is used to model difference equations, although it is slightly more general, and avoids the global synchronization implied by a pure interpretation of difference equations.

2.5.3 Process networks and dataflow

In a Process Network (PN) model of computation, the arcs represent sequences of data values (tokens), and the bubbles 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. Dataflow models, popular in signal processing, are a special case of process networks [25].

PN models are excellent for signal processing. They are loosely coupled, and hence relatively easily to parallelize or distribute. They can be implemented efficiently in both software and hardware (something demonstrated by this project), and hence leave many implementation options open. Thus, they are best used for problem-level specification.

A key weakness of PN models is that they are awkward for specifying control logic.

PN models are implemented in Ptolemy using a hierarchy of four nested domains. These are, from smallest (least general) to largest (most general): *synchronous dataflow* (SDF), *boolean dataflow* (BDF), *dynamic dataflow* (DDF), and *process networks* (PN).

2.5.4 Synchronous/reactive models

In the Synchronous/Reactive (SR) model of computation, the arcs represent data values that are aligned with global clock ticks. Thus, they are discrete signals, as with difference equations, but unlike difference equations, a signal need not have a value at every clock tick. The bubbles represent relations between input and output values at each tick, and are usually partial functions with certain technical

restrictions to ensure determinacy.

SR models are excellent for applications with concurrent and complex control logic. They can be realized in the popular Esterel language and certain variants of the Statecharts language. Because of the tight synchronization, however, some applications are overspecified in the SR model, limiting the implementation alternatives. Moreover, in most realizations, modularity is compromised by the need to seek a global fixed point at each clock tick.

2.5.5 Discrete-event models

In discrete-event (DE) models of computation, the arcs represent sets of *events* placed in time. An event consists of a *value* and *time stamp*. This model of computation is popular for specifying hardware and simulating telecommunications systems, and has been realized in a large number of simulation environments, simulation languages, and hardware description languages, including VHDL and Verilog. Unlike the SR model, there is no global clock tick, but like the SR, differential equations, and difference equations, there is a globally consistent notion of time.

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. A key weakness is that it is relatively expensive to implement in software, as evidenced by the relatively slow simulators.

2.5.6 Rendezvous models

In a rendezvous model, the arcs represent sequences of atomic exchanges of data between sequential processes, where the bubbles represent the processes. “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. Examples of rendezvous models include Hoare’s *communicating sequential processes* (CSP) and Milner’s *calculus of communicating systems* (CCS). This model of computation has been realized in a number of concurrent programming languages, including Lotos and Occam.

Rendezvous models are particular well-matched to applications where resource sharing is a key element, for example, client-server database models and multitasking or multiplexing of hardware resources. A key weakness of rendezvous-based models is that maintaining determinacy can be difficult. Proponents of the approach, of course, cite the ability to model nondeterminacy as a key strength.

2.5.7 Finite-state machines

In FSMs, bubbles represent system *state* and arcs represent state *transitions*. This model of computation is radically different from all the previous ones in that it is not concurrent. Execution is a strictly ordered sequence of state transitions.

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. Moreover, FSMs are easily mapped to either hardware or software implementations, and thus are suitable for use at the problem level.

FSM models have a number of key weaknesses. First, at a very fundamental level, they are not as expressive as the other models of computation described here. They are not sufficiently rich to describe all partially recursive functions. However, this weakness is acceptable in light of the formal analysis that becomes possible. Many questions about designs are decidable for FSMs and undecidable for other models of computation. A second key weakness is that the number of states can get very large even in the face of only modest complexity. This makes the models unwieldy.

The latter problem can be solved by using FSMs in combination with concurrent models of computation. This was first noted by David Harel, who introduced that Statecharts formalism, which combines a loose version of SR with FSMs. FSMs have also been combined differential equations, yielding the so-called *hybrid systems* model of computation.

A major (ongoing) result of this project has been to show that FSMs can be hierarchically combined with all of the concurrent models of computation described above. We call the resulting formalism "*charts" (pronounced "starcharts") where the star represents a wild card. Limited combinations of FSM with synchronous dataflow and discrete-event were implemented and released, although a great deal more work remains to be done.

2.6 CHOOSING MODELS OF COMPUTATION

The rich variety of available concurrent models of computation outlined in the previous section can be daunting to a designer faced with having to select them. Most designers today do not face this choice because they get exposed to only one or two. This is changing, however, as the level of abstraction and domain-specificity of design software both rise. We expect that sophisticated and highly visual user interfaces will be needed to enable designers to cope with this heterogeneity.

An essential difference between concurrent models of computation is their modeling of time. Some are very explicit by taking time to be a real number that advances, and placing events on a time line or evolving continuous signals along the time line. Others are more abstract and take time to be discrete. Others are still more abstract and take time to be merely a constraint imposed by causality. This latter interpretation results in time that is partially ordered, and explains much of the expressiveness in process networks and rendezvous models of computation. Partially ordered time provides a mathematical framework for formally analyzing and comparing models of computation. This observation has led to some key theoretical results partly under this project. These results have profoundly affected our view of Ptolemy domains and their interrelationships.

A grand unified approach would seek a concurrent model of computation that serves all purposes. This could be accomplished by creating a *melange*, a mixture of all of the above, but such a mixture would be extremely difficult to use, and synthesis and simulation tools would be difficult to design. Another alternative would be to choose one concurrent model of computation, say the rendezvous model, and show that all the others are subsumed as special cases. This is relatively easy to do, in theory. Most of these models of computation are sufficiently expressive to be able to subsume most of the others. However, this fails to acknowledge the strengths and weaknesses of each model of computation. Differential equations, for instance, are very good at describing the interaction of point masses in a model of a MEMS system, but not as good at describing the discrete control logic that may be ultimately controlling the actuators in the MEMS system. Similarly, finite-state machines are good at modeling at least simple control logic, but hopelessly inadequate for modeling the interaction of point masses. Thus, to design interesting systems, designers will have to use heterogeneous models.

3. Approach

3.1 ANGLE OF ATTACK

Our approach can be summarized as follows:

- We are investigating the theory and techniques needed to mix diverse models of computation, for example mixed signal, hybrid systems, and more generally systems that mix discrete and continu-

ous events.

- We are developing theory and software for domain-specific modeling of composite concurrent systems, i.e., the diverse models of computation in the previous bullet.
- We are using programming language concepts such as type theories, semantics, and concurrency theories for the modeling of concurrent composite systems.
- We are developing a software architecture for modular, distributed, and heterogeneous design, modeling, simulation, and visualization.
- We are emphasizing modeling and specification methods that are amenable to visual syntaxes such as block diagrams.

With regard to the first two items, the specific issues that the project is addressing include:

- Semantics (what constitutes a behavior of a system)
- Determinacy (how many behaviors are there)
- Simulation (finding a behavior)
- Analysis (finding properties of behaviors)
- Compositionality (encapsulating subsystems)
- Synthesis (translation to implementation)
- Design (choosing implementations)
- Heterogeneity

3.2 EXAMPLES REQUIRING HETEROGENEITY

Throughout the project, we will be considering examples requiring heterogeneity such as the following:

- MEMS device with a discrete controller (differential equations plus discrete-event models)
- Modal models, with regimes of operation (differential equations plus finite-state machines)
- Mixed signal systems (differential equations plus discrete-time and/or discrete-event systems)
- Hardware/software systems (differential equations, discrete-events, discrete-time, finite-state machines, dataflow, rendezvous, process networks, etc.)

3.3 TASKS

3.3.1 Phase 1 (18 months)

Task 1: Modular deployable design tools. System-level design tools such as our Ptolemy environment tend to be large monolithic software systems, following the VLSI CAD tradition. This makes them difficult to use, maintain, and support, Moreover, increasingly, design tools are expected to perform in an environment where design and evolution of a larger system is ongoing, persisting well beyond initial system deployment. We are exploiting software technologies such as abstract machines, web-based design, migrating processes, client-server architectures, and object-request brokers to break apart the design tools into modular building blocks. One consequence is that certain elements of a design tool (such as schedulers, user-interfaces, displays, controls, and even models) can be deployed as part of system, facilitating maintenance, adaptation, evolution, and documentation. An underlying object-oriented software architecture will provide the modular heterogeneity amenable to such partitioning.

Task 2: Domain-specific design tools. High-level design tools tend to be domain-specific, supporting a

narrow range of design problems. Successful examples include Spice for circuit modeling, digital hardware design tools, and visual dataflow environments for signal processing. Many of these tools share common principles. Most can be viewed in fact as languages in that they have a syntax (either ASCII or pictorial) and a semantics. For electronic systems, the semantics is often concurrent, in that modules conceptually (or physically) operate at the same time, interacting through signals. For MEMS systems, the semantics is always concurrent, reflecting an underlying physical modeling problem, and interaction is through physical effects. We are studying and developing various domain-specific approaches appropriate to composite CAD. These include Spice-level modeling of circuits and mechanical systems, discrete-event modeling of synchronous and asynchronous digital hardware, dataflow modeling of discrete-time systems and embedded software, state-machine modeling of sequential controllers, and various higher-level models that focus on resource allocation and system-level design, applicable for example to hardware/software codesign.

Task 3: Heterogeneous interaction semantics. Frequently, domain-specific design tools and languages need to be combined to design and model heterogeneous systems. One example of great commercial interest is in wireless communication, where Spice-level modeling of RF circuits needs to be combined with functional modeling of signal processing that is often implemented in embedded software. MEMS systems are similar, if a bit more complex, in that they may combine physical models at the level of differential equations, Spice-level circuit models for analog driving and sensing circuitry, functional models of sequential control logic, and functional models of signal processing that deal with sensor data or actuator control signals. Each of these is best supported by a distinct modeling paradigm, language, and tool. These tools tend to grow and evolve in isolation, making it difficult to combine them to achieve system-level design. We are studying and developing theory and techniques for heterogeneous combinations of such tools and languages.

3.3.2 Phase 2 (18 months)

Task 4: Process level type system. Strongly typed programming languages are more robust than weakly typed languages. Static type systems can prevent many common software faults and facilitate compiler optimizations. We will extend this concept to composite system-level design to facilitate heterogeneous modeling and design. We will develop a type system for concurrent processes that regulates the interaction between heterogeneous system components. The notion of types will be adapted beyond that of numerics and data structures to encompass such notions as a continuous-time signal, a discrete-time signal, a set of discrete-events in time, a rendezvous, or a sequence of messages. Instead of defining variables, our types define signals. The forms that signals can take determine the types of the processes that interact with these signals. A type hierarchy will determine how processes can be combined and type resolution and translation will be developed to support heterogeneous modeling.

Task 5: System-level validation. Formal analysis can play a major role in validating designs of embedded systems. Models of computation based on a formal mathematical framework, such as differential equations, synchronous/reactive models, communicating finite-state machines, and dataflow models, have been used to design systems that provably have some desirable property such as safety, stability, or liveness. Such techniques have been applied in hybrid systems, which combine continuous-time differential equation models with discrete finite automata. However, the methods used today mostly do not scale well to practical systems. Key questions frequently become undecidable, and many of those that remain decidable become intractably complex. We will develop systematic techniques that can applied to practical systems. The keys are information hiding and hierarchy. Simple, mathematical models of computation can be used to define the interaction between modules at a large grain level. We

will show that by hiding the internal implementation of modules, they can become large and complex without adversely affecting the ability to answer key validation questions at the system level.

Task 6: System-level visualization. We will construct an object-oriented design visualization environment for complex, evolving, distributed systems. Initially (at least) this will be based on the Tycho syntax manager that we have begun under the Ptolemy project. Tycho supports syntax-directed editing and domain-specific graphical visualization by using a modular, object-oriented software architecture. This architecture will support deployable design tools; visualization modules will be separable from the design environment. We will devise visual representations to illustrate the concurrent and real-time behavior of composite systems. We expect that different visual syntaxes will be required for different models of computation (physical modeling of MEMS components will likely require significantly different representations than the automata that control them). We will devise live, direct-manipulation interfaces for design and modeling composite systems.

4. Summary of Accomplishments

The major accomplishments of the project are summarized in this section. Concrete deliverables included monthly and annual reports, software, demonstrable technology transfer, and 24 publications to date, all of which have been posted on the World Wide Web.

4.1 TASK 1: MODULAR DEPLOYABLE DESIGN TOOLS

4.1.1 Java Exploration

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

The Java language offers a number of significant advantages for use in our context. Like Ada, it is much safer than C or C++ because of its built-in memory management and its lack of pointer arithmetic. Also like Ada, it is a concurrent language, with a built-in thread library. It has also proven portable and distributable, with well-conceived and robust mechanisms for migrating code across platforms. Finally, a huge investment is being made in industry in software development environments, class libraries, and applications. We can leverage this investment.

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

A second key question is whether the Java thread library provides an adequate platform for constructing concurrent applications. The basic mechanism, wait and notify, is too low level for most applications programming. In particular, direct use of such low-level primitives makes validation of

concurrent programs very difficult. For example, there is no direct way of ensuring determinacy and preventing deadlock. Indeed, our experience writing threaded Java code underscores the need for a higher level abstraction above threads.

We have completed two pilot projects that show that higher-level determinate mechanisms can be built on top of Java threads. In the first of these, Dick Stevens, a visiting scholar from the Naval Research Labs, together with two graduate students, Marlene Wan and Peggy Laramie, have designed a set of classes that implement Kahn process networks. KPNs have sequential processes that communicate via unidirectional FIFO queues. Based on previous work at Berkeley and at Lincoln Labs by Tom Parks, they were able to implement a simple algorithm that ensures that the size of the FIFO queues will not grow unbounded unless the program requires them to grow unbounded. We have set out, as detailed below, to construct a Java-based Ptolemy software architecture that will have PN as one of its first domains.

The second project, completed by Jim Young of the CAD group at Berkeley, implements a barrier synchronization mechanism on top of Java threads.

These two pilot projects are very encouraging because the two concurrency models are very different from one another, and yet both appear to be efficiently implementable on top of Java threads.

4.1.2 Java Ptolemy Kernel

We are in the process of designing and implementing a set of Java classes that realize key functionality in the Ptolemy kernel. An image of the more essential classes and their relationships are shown in figure 5. The design of these classes has been painstaking and careful. We have emphasized systematic software engineering over speed of development. One reason for this is that the group of students, staff, and faculty working on the software are all relatively inexperienced with Java, and most of them are also relatively unfamiliar with the Ptolemy kernel. We have instituted careful code review mechanisms, and have rewritten the most critical code several times. The result so far has been to build a small and very well-designed core software infrastructure, and to build a cohesive team of experts.

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

Hierarchical graphs are collection of “entities” and “relations”. Entities have “ports” and relations connect the ports. Entities can contain a graph, and ports in the contained graph can be exposed as ports of the container entity. A port can link to any number of entities, and thus is equivalent to the MultiPort of the C++ Ptolemy kernel. There is no simple port in the Java kernel.

Consistent with theoretical results recently obtained as part of the Ptolemy project [21], we have been designing the Particle class to support tags and values. Particles are explicitly given the notion of tags and values by implementing a tag/value interface hierarchy (in the Java sense). In some cases, say SDF, the tag of a particle may be non-existent and in other cases, say FSM, the value of a particle may be non-existent.

4.1.3 Network Integrated Design

We have made progress on several fronts in making our design tools and methodologies network aware.

John Reekie and Kevin Chang made Tycho internet aware. Tycho is the Itcl side of our software

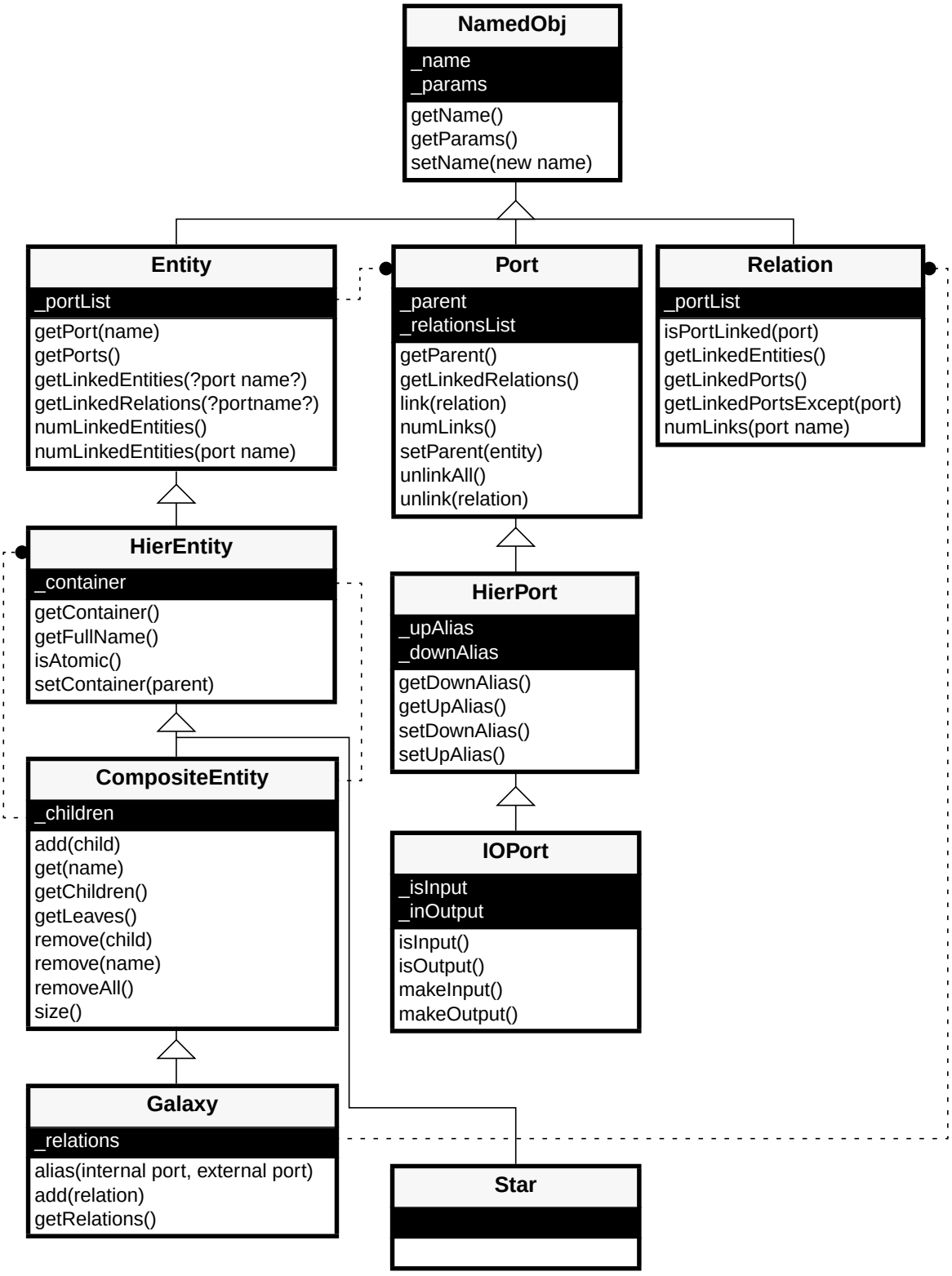

FIGURE 5. Key classes and their methods in the Java Ptolemy kernel.

effort. It provides an infrastructure for data management and user interface design. Their software architecture supports transparent access to URLs, as if they were local files. Every place where Tycho has been able to reference a local file it can now reference a URL. This is done by using Tcl's built in socket mechanism to connect directly to HTTP servers through the network. This mechanism is portable (it works under NT as well as Solaris) and robust.

Kevin Chang has also created a portable interface to electronic mail within Tycho. Thus, Tycho scripts can now be easily written that will send electronic mail, and hyperlink references now support the "mailto:" protocol. The EditMail editor uses the ResourceMail class to contact the mail server on socket port 25. Using sockets instead of UNIX mail, ResourceMail is platform independent. We are experimenting with using this to enhance our own information flow within our software development effort.

We have also collaborated with the WELD project (web-based electronic design), headed by Richard Newton. WELD aims to provide software infrastructure where design tools execute on compute servers in the network and are used by designers anywhere. John Reekie has built into Tycho a number of features to support access to WELD databases and applications. The intent of the integration is to use Tycho to provide more complex user interfaces to WELD design tools that are operated over the network. Tycho provides a significantly higher-level platform for user interface design than Java, on which WELD is based.

John Reekie has demonstrated using Ptolemy remotely via the WELD protocol with Tycho providing the user interface. He gave the demo at DAC (the Design Automation Conference). We learned from this that some refinement of the WELD protocol is needed to support more dynamic simulations. Using the WELD protocol would have a number of advantages like independence from the user interface, ability to monitor just the data of interest from various simulations, perhaps a workflow application that supports dynamic monitoring, and control of any simulation.

This work has contributed some infrastructure to Tycho, including persistent object support classes, which Tycho uses to access the WELD database (written in Java), a client-side interface to a WELD server (Java), and a simple server process that runs Ptolemy under the control of Tycho. This is perhaps the first client-server implementation of Ptolemy.

To make Tycho "WELD-aware", John Reekie wrote a subclass of the Resource class, called ResourceWELD, and added the protocol name "weld" to the existing list ("http", "mailto", "file"). Because of this approach, access to the WELD server is transparent: if you open a file browser (in tycho -java) and enter weld://machinename, you will get a listing of the contents of the WELD server on that machine. John has installed on the main WELD server machine (called "yoyodyne.eecs.berkeley.edu") a graph editor with a simple dataflow graph. There is a menu entry on the graph editor called "Run Remote Ptolemy", which will generate a set of textual commands to Ptolemy and send them to a Tycho server. The Tycho server can also be run from the WELD workflow applet. This infrastructure, however, keeps breaking because WELD and Tycho are both very much work-in-progress and their interfaces keep changing.

We learned a few lessons. First, running Ptolemy in client server mode is easy because of its Tcl-based textual command interface. Second, security is a big problem with this setup, since the server is executing an arbitrary Tcl script. A safe Tcl interpreter is required. Third, most Ptolemy applications need to be modified to work this way because most assume that the user interface coincides on the same machine with the simulation. Thus, for example, output data needs to be returned over the network rather than displayed directly in a plot on the screen. Finally, more development would be needed to fully realize interactive executions, as opposed to executions that put their results in a file. The WELD team is interested in extending their protocol to better support Ptolemy-style interactive simu-

lations.

Finally, inspired by the WELD work, UT Austin (with partial support under our subcontract) has announced Version 1.0.5 of a web-enabled simulation (WEDS) framework for embedded software for DSPs and microcontrollers. The web page is:

http://anchovy.ece.utexas.edu/~arifler/wetics/

The team developing this is led by Prof. Brian L. Evans and includes Dogu Arifler, Chi Duong, Srikanth Gummadi, Saleem Marwat, Chris Moy, Ha Nguyen, Han Nguyen, and Anna Yuan. The framework runs on a Java-enabled Web browser, and consists of the following:

- A set of Java applets that provide a configurable framework for Web-based user interfaces for instruction set architecture simulators.
- A multithreaded TCP/IP Internet server written as a Java application that provides the interface between the Java applets and the instruction set architecture simulators.
- Command-line ISA instruction set architecture simulators written in C/C++ that run under Windows '95, Windows NT, and more than twelve Unix architectures including Solaris 2.5 and Linux for the following processors:

  a. Texas Instruments TMS320C30 floating-point digital signal processor

  b. Motorola MC68HC11 microcontroller.

Acknowledgments: UT based the MC68HC11 simulator on a freely distributable simulator engine written by Ted Dunning at New Mexico State University and Tomaso Poletti from Follina Italy. They based the TMS320C30 simulator on the freely distributable TMS320C30 simulator on the freely distributable C30 DSK tools by Keith Larson at Texas Instruments. The makefile structure is based on Ptolemy.

4.1.4 Java-Tcl Interaction

In the near term, our software environment will mix Java and Itcl. We feel that the combination of a scripted language such as Itcl and a faster compiled language such as Java will be a compelling combination for a wide variety of network-aware applications. Both are platform-independent, network-savvy, and object-oriented. Java is at least 10 times faster that Itcl, so for computationally intensive applications, calling Java methods makes a lot of sense. For high-level control and dynamic reconfiguration, Itcl offers a higher-level interface.

Over the last year, we have been through several Java/Itcl interfaces. The long term solution appears to be TclBlend, recently released by Sun Microsystems. Currently, TclBlend does not work with Itcl, but it does work with Tcl (the non-object-oriented base language). We have been working closely with the SunScript group in configuration management and porting issues for TclBlend, and in fact contributed to its development in a number of ways.

TclBlend is a Tcl/Java interface that uses C code that calls the Java Native Invocation (JNI) module in JDK1.1 (the currently stable Java development kit). We have found problems, for example, with multiprocessor platforms (where SunScript's own test suites fail). We plan on using the Tcl/Java interface to provide a flexible scripting interface to software modules like Ptplot and the new Java Ptolemy kernel. Our first use is to construct a scripted test suite for the new Java Ptolemy kernel that we are building.

Sun has also released Jacl 1.0, which is a "100% pure Java" implementation of a subset of Tcl.

Unfortunately, Jacl cannot currently be used to construct scripted applets, and it throws security exceptions on any attempt. We are not exactly sure what Sun is trying to accomplish with this project, but have been experimenting with it nonetheless.

Prior to TclBlend, we wrote an interface between Tcl and Java that uses the Java Reflect class. This interface was considerably easier to use than Sun's TclJava0.4 interface, Sun's first attempt, making the integration more transparent. We believe that we influenced the design of TclBlend through this work.

4.1.5 Java Signal Plotter

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

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

There are a number of reasons behind our choice to release this module first:

1. It is neither trivial nor excessively complex, and thus represents a manageable amount of software with which to learn about the issues in writing and releasing Java code.
2. The application combines numerical computation, threading, and simple user interfaces. Thus, it hits most of the key facets we expect to find in such modules.
3. It is a clearly circumscribed module that is useful on its own as well as within the larger Ptolemy system.

The plotter is backwards compatible with pxgraph, the signal plotter that has been used in Ptolemy, but adds the interactive animated plotting feature of TkPlot, which is also used in Ptolemy. Thus, it will replace these two facilities. There are a number of things we have learned so far from this experience:

1. Writing nontrivial multithreaded Java programs is very difficult. It is hard to avoid deadlock and starvation conditions. We believe that the Ptolemy process networks domain that we will be reimplementing in Java will make this much easier by providing a much higher level interface to concurrency primitives than Java threads provide.
2. Writing Java code that runs everywhere is much more difficult than would be ideal.

The plotter has the following properties:

- It can be an applet or an application.
- It can read binary or ASCII data off the net or from local files.
- It supports auto-ranging of the scales.
- It does automatic or manual labeling of axes.
- It supports automatic or manual tick marks.
- Live, animated plots.
- Infinite zooming.
- Various plot styles: connected lines, scatter plot, bars, etc.
- Various point styles: none, dots, points, and unique marks.
- Multiple data sets and a legend.
- Color or black and white plotting.

The applet implementation can read data to plot from a URL or construct the data in custom Java code. The application can read data from a file and use command-line arguments to format the plot. The command-line arguments are compatible with pxgraph.

There is a set of demonstrations of these classes. The main class implementing the plotter is Plot. It is derived from PlotBox, which provides only the axes and decorations of the plot. This is implemented in a base class so that it can be reused for different kinds of plots. Live (animated) data plots are supported by the PlotLive class. This class is abstract; a derived class must be created to generate the data to plot (or collect it from some other application). The application is implemented by the Pxgraph class. The Pxgraph class includes support for printing and generating HTML for use with the Plot applet. Unix users can invoke the pxgraph Bourne Shell script to simulate the pxgraph X11 binary on machines where the X Window System is not available. See the Pxgraph class documentation for pxgraph script installation instructions under Unix. Windows users can invoke the pxgraph.bat DOS batch file. Pxgraph is a slight extension to xgraph which reads binary files as well as ASCII. This code owes a heavy debt to David Harrison, the original author of xgraph. The extensions in pxgraph were written by Joe Buck.

The 1.0 release achieved a rating in the top 5% of JARS - The Java Applet Rating Service (http://jars.developer.com/).

The changes between ptplot1.0 and ptplot1.1 were:

- In 1.1, PlotBox extends Panel. In 1.0 PlotBox extended Applet. This makes it much easier to use Plot as an component in either an application or an applet that contains other components
- The PlotApplet class is new in 1.1. PlotApplet can be used for applets that use the Plot class, or the PlotLive class. As a result, the HTML applet tags for the demos now use code="ptplot.PlotApplet". Formerly, they used code="ptplot.Plot".
- The directory structure has been reorganized so that the plot classes are in ptplot, and the demo classes are in ptplot/demo. This means that the ptplot.zip does not contain the demo classes, which is better if anyone actually wants to use the plotter instead of just running the demos.
- There is a new demo that uses real time audio.
- Fixed a bar graph label bug reported by Marc Ellis.
- A few other minor bugs were fixed.

4.1.6 Tycho Information Models (TIM)

John Reekie has defined a simple framework to support exchange of information and cooperation between modular tools. The aim is to make it easy to describe a model, to describe relationships between models, and to define new models.

A Tycho information model (TIM) is the unit of information read and generated by tools. It is conceptually similar to the file representation of a compound document in systems like OpenDoc -- that is, it contains structured data, but it is not an object or a database. Models have attributes, which contain information about the model and its relation to other models, entities, which are conceptually similar to objects in object models, and associations between entities, which are similar to associations or links in object models.

The TIM classes are expected to form the base classes for virtually all domain-specific visual editors and visualization tools that we will be building in Tycho. TIM is based on a “model-view” (MV) paradigm, which is a simplification of the classical “model-view-controller” (MVC) paradigm that was originally elaborated in Smalltalk. In MV, the view and controller are consolidated into a single object

that serves both the visualization and manipulation role. This consolidation is warranted by the huge improvements in user interfaces since MVC was first developed.

The basic idea in MV is that data is represented abstractly in a “model” and one or more “views” render that data concretely. For example, a model may represent a netlist as a mathematical graph and a view may render it as a block diagram. However, a given model may have more than one view. Thus, for example, the same netlist might be rendered as a gantt chart displaying an execution schedule. Either view may include editing, or direct manipulation capabilities, that modify the data stored in the model.

Central to the MV concept is that views subscribe to the model and are automatically notified when changes to the model are made. The model “publishes” data modifications, and the views “subscribe” to the model to be notified of modifications. The base class, called Model, implements the publish-and-subscribe infrastructure. It also implements a reasonably flexible unbounded undo and redo mechanism, something that almost all editable models will require.

DataModel extends the Model class to support data models and the TIM interchange format. TIM is a simple meta-data format that encourages a simple and clean representation of data, both in in-memory objects and in an external file representation. A data model is loaded from a string in TIM format with the *parse* method, and will produce a TIM description of itself with the *describe* method. The DataModel class also provides infrastructure for searching and sorting the objects in the model.

Models are expressed in terms of a simple and flexible syntax. A simple syntax makes it easy to create new models: you do not need to invent new syntax, but just to declare the things in your model. The syntax is based on the observation that there are two common ways of describing data: i) by giving a mapping from names to values; and ii) by saying what a thing is and then giving information about it. TIM calls these attributes and declarations.

TIM is specified at two levels: an abstract syntax states what things TIM can describe and how; a concrete syntax states how you write it. The atomic units of the abstract syntax are types, names, and strings. The concrete syntax used in Tycho is a Tcl-like syntax.

Because TIM is a simple and structured format, it is relatively simple to parse and generate. In Tycho, the Model class provides the support needed to parse and generate TIM files. Subclasses that implement particular models with their own set of entities and associations need only follow a couple of simple rules to inherit this capability. TIM is implemented in the Model class in Tycho, and so far is used in the preferences manager and in the Schedule class. The Model class handles undoing and redoing changes and provides support for making data persistent.

John Reekie has redesigned the preferences manager in Tycho using TIM to make it more modular. Previously, the preferences manager was entirely centralized, providing a significant barrier to modularizing Tycho. The preferences manager needed to be aware of all existing classes. Now, classes register a style sheet with the preferences manager, so classes can be added dynamically and there is no particular fixed set of preferred classes.

The new version is based on the concept of a style sheet. Packages can define their own style-sheets, and each style-sheet can have multiple styles from which the user can choose. This can be overridden on a per-class basis.

4.1.7 Object Modeling

John Reekie has undertaken to lead a study group that is examining current practice in object modeling in general and the UML language in particular. This study is being applied to critically examine the current and future design of Ptolemy. Working documents describing the ongoing work can be

found at:

http://ptolemy.eecs.berkeley.edu/~johnr/tycho/design/

In particular, the following issues have been specifically addressed:

1. The relationship between functional blocks in a netlist and their graphical representation (stars and icons, in the terminology of Ptolemy). One major issue that has been considered is that a given icon may actually represent several possible implementations of the same functionality. Also considered is the relationship between mechanisms for interacting with a functional block (parameters and ports) and their graphical representations.
2. The relationship between ports (logical input and output of a functional block) and terminals (the graphical representation thereof). Questions about how to handle aggregation of indefinite numbers of ports and how to restrict graphical manipulations of ports have been addressed.
3. The representation of logical connections in a netlist. Issues considered include the need to annotate connections, the need to order terminals in a net, and the need to have both point-to-point and multi-way nets. The group has also created a dynamic model for the process of creating and manipulating a connection.

Currently, the group is addresses the uses of design patterns. Patterns seem to be a hot topic in the object-oriented design community, both academic and commercial. The promise of patterns is to provide a means of capturing and reusing object-oriented design expertise that simply cannot be expressed in terms of specific classes or applications.

The object modeling study group has also completed a study of interfaces, a central capability in the Java language. The different purposes for interfaces that we identified are:

Role in a collaboration . An interface provides the syntactic declaration of a role in a collaboration between objects. Design patterns typically contain collaborations suitable for interfaces -- for example, Strategy, Observer, Iterator. Classes can participate in multiple collaborations by implementing multiple interfaces.

Multiple classification . (the Java version of multiple inheritance) A class may naturally be classified in two different ways, and implementing multiple interfaces (or inheriting from one superclass and implementing interfaces as the others) is the only way in Java to directly express this.

Separate interface from implementation . Because interfaces do not have any implementation, interfaces can be used to separate the interface presented by objects (its type) from the implementation. Some people even advise that every class should be split into an interface and a class. Doug Lea's Collections classes use this idea extensively.

Marker interface . Some interfaces are empty -- their sole purpose is as a “marker” that declares that classes that implement it have a certain function. The Java classes Cloneable, Serializable, and Remote are marker interfaces.

In view of these classifications, we studied the details of a number of built-in Java interfaces, including java.lang.Runnable, java.lang.Cloneable, java.rmi.registry.Registry, and a number of interfaces in Doug Lea's Collections classes.

The current prototype of Java Ptolemy contains interfaces for tags and values, both of which must

be implemented by Particle classes. Subclasses of Particle can implement more specific sub-interfaces of Tag and Value, thus providing compile-time type checking on particle usage. We are re-examining this design in view of what we've learned.

4.2 TASK 2: DOMAIN-SPECIFIC DESIGN TOOLS

4.2.1 Process Network Domain in Java and Tycho

Tom Park's Process Network (PN) domain in Ptolemy implements an asynchronous, highly distributable concurrent model of computation that generalizes dataflow. It appears to be a very good candidate for providing a higher-level abstraction in Java (above threads) for concurrent programming. The current Ptolemy implementation uses threads via C++. Tom wrote a Sieve of Eratosthenes demo in Java that uses the Java threads to compute the first N prime numbers, giving an existence proof that Java provided a reasonable substrate.

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

4.2.2 Benchmarking Code Generation Methodologies for Embedded Software

Under subcontract, Brian Evans and his team at UT Austin have completed an in-depth study where they benchmark code generation methodologies for programmable digital signal processors [10]. They evaluate rapid prototyping tools and compilers as code generation methodologies for programmable digital signal processors (DSPs). Code generated by compilers and rapid prototyping tools have been reported as significantly less efficient in memory usage and execution time versus assembly language code written by expert programmers. As complexity of the system increases, however, the scale tips in favor of the automated code generation techniques. They quantify when this trade-off occurs on a Motorola 56002 DSP using the Motorola KCCA56 C compiler version 1.26 (May 22, 1996) and the automated C and 56000 code generators in Ptolemy version 0.7 (June 13, 1997). The Motorola KCCA56 C compiler, which is GNU-based, is available on the Motorola DSP Development Tools CD ROM.

In their evaluation of code generation techniques, they used 3 kernels and 3 stand-alone applications. The kernels were an IIR filter, a 256-point Complex FFT, and Goertzel’s DFT. For the applications, they used Ptolemy demonstrations that obey Synchronous Dataflow semantics: (1) IIR filtering, (2) CD-DAT converter, and (3) Dual-Tone Multiple-Frequency (DTMF) touchtone codec. To isolate the effects of (a) compiler inefficiencies and (b) automatic scheduling algorithms, they coded the kernels and the IIR filtering application (1) manually in 56002 assembly code (2) manually in C and compiled using the KCCA56 C compiler, (3) using Ptolemy’s Code Generation for the 56000 (CG56) domain, and (4) using Ptolemy’s Code Generation in C (CGC) domain to generate code in C which was later compiled using the KCCA C compiler.

They implemented the other two applications using (2)-(4). Comparing assembly language and C implementations exposes compiler deficiencies, whereas comparing hand-written code with Ptolemy generated code emphasizes differences in manual vs. automated scheduling algorithms. In addition, by comparing the results for the same application given by the Ptolemy CG56 domain vs. the Ptolemy CGC domain plus the KCCA56 C compiler with the same scheduler, they isolate the effects of the KCCA56 C compiler.

Table 1 below (from the benchmarking for the CD-DAT converter), shows that Ptolemy performed very well on two fronts. First, with its libraries of optimized code blocks, it was able to come up with

smaller code size. Second, its optimizing scheduler resulted in much shorter execution time for the same application. These same effects were visible to an even larger extent with the DTMF Codec application in which the hand-coded C version resulted in a final object code that was too large to fit onto our 56002 evaluation board's program memory.

Table 11: Memory Usage and Execution Times for the CD-DAT Converter

	Ptolemy CG56	Ptolemy CGC	Hand coding in C
Program Memory	413	586	687
X Data Memory	456	468	398
Y Data Memory	283	280	324
Execution Time	295069	381076	463004

Ptolemy-generated 56000 assembly language programs outperform compiled hand-coded C implementations in program memory usage and execution time, and are comparable in data memory usage. The gap widens as complexity increases. At some point between the complexity of the IIR filtering and CD-DAT converter applications, Ptolemy-generated C programs begin to outperform hand-coded C implementations in the same way. The increase of complexity from a CD-DAT converter to a DTMF codec causes an increase by an order of magnitude of the efficiency of Ptolemy-generated C programs over the hand-coded C implementations. Although not shown, we found that for the IIR filtering demonstrations that Ptolemy-generated assembly language programs were only 2% worse in performance vs. hand-coded assembly implementations.

As far as which code generation methodology to use to create the most efficient implementations in a DSP assembly language, they draw the following conclusions:

1. When the choice is between writing C code manually for compilation and using a synthesis tool to generate assembly language directly, the synthesis tool should be chosen.
2. A key use of a C compiler is to complement a tool that synthesizes assembly language programs by generating efficient implementations of kernels in assembly language when they do not exist.
3. When the choice is between writing assembly language code manually and using a synthesis tool to generate assembly language, the synthesis tool should be chosen for applications of complexity slightly greater than a DTMF codec.

4.2.3 Real-Time Sonar Beamforming using Process Network Models (UT Austin)

Sonar beamforming algorithms, which require on the order of one billion operations per second, have traditionally been limited to custom hardware implementations to meet real-time requirements. Real-time sonar beamformers in the field today are based on custom application-specific integrated circuits (ASICs). A typical sonar beamformer processor costs $1 million. At present, only one company manufacturers real-time sonar beamformer processors. However, recent developments in both hardware and software present the possibility of accomplishing real-time beamforming on UNIX workstations at a fraction of the cost of a custom hardware solution.

Prof. Brian Evans (on a subcontract to UT Austin) and Mr. Gregory Allen (Advanced Sonar Group, Applied Research Laboratories, UT Austin) have been developing scalable software to implement real-time sonar beamformers on UltraSparc workstations using the Solaris operating system [12].

They have been implementing digital interpolation beamforming algorithms specified using the Process Network computation model [25] and implemented using POSIX threads. Being a superset of dataflow models, the process network model captures the parallelism in data-intensive digital signal processing algorithms such as sonar beamformers.

The software beamformer gained near linear speedup over one, two, and four processors. On two processors, the processor utilization was over 95%. At the present time, real-time sonar beamforming is feasible on a workstation containing 12 high-end UltraSPARC-II processors. Such as workstation costs on the order of $250,000, which is one-fourth the cost of the ASIC sonar beamformer.

4.2.4 Real-Time Smart Antennas for Wireless Communications (UT Austin)

The proliferation of wireless communication services have been stimulating unprecedented demands for scarce radio spectrum. A key emerging technique for increasing the number of users in a cell site is to add spatial diversity to existing standards, such as the IS-95 CDMA standard. Spatial diversity among the users is incorporated by means of an antenna array with embedded processing. As the number of standards continue to increase, a significant need is arising for software base stations that can support multiple standards by means of deployable, configurable software. The reason is that base stations cost on the order of $150,000 to build, and it is becoming more cost effective to reuse the existing stations than to build new ones for each new standard.

Prof. Brian Evans' research team at UT Austin (under subcontract) has developed three new approaches for implementing smart antenna systems in real-time:

1. optimal weight vectors for spatial broadcast channels [13][4]
2. blind estimation of FIR channels in CDMA systems [14]
3. modified constant modulus antenna arrays [15]

The first two approaches can be modeled using a mixture of Boolean Dataflow and Synchronous Dataflow, and the third approach is completely Synchronous Dataflow. All three approaches lead to statically scheduled implementations. The first two approaches have been implemented in the Real-time Wireless Communications Testbed run by Prof. Guanghan Xu (UT Austin). Ptolemy demonstrations and real-time implementations of the third approach are forthcoming.

In the first approach, the UT Austin team improves capacity of broadcast channels by employing multiple transmitters and exploiting the spatial diversity among the users. They derive fast algorithms to compute orthogonal, near-optimal, and optimal weight vectors for broadcasting message signals to two and three users. The key innovation is that we decouple the weight vectors in the measure of channel capacity to (1) simplify the optimization problem to a search for the maxima of a smooth multidimensional function and (2) derive closed-form expressions for the orthogonal and near-optimal algorithms. All three methods are well-suited for implementations on embedded digital signal processors or workstations.

The second approach takes advantage of the structure in CDMA signals. CDMA systems commonly use aperiodic spreading codes to distribute a signal spectrum uniformly over the channel bandwidth and differentiate neighboring cell sites. CDMA receivers often suffer from interference due to multipath fading. Blind signal estimation schemes cannot be used because they require access to one or more periods of the spreading sequences, and the period of aperiodic spreading codes in the IS-95 standard is on the order of 1800 days. RAKE receivers are often used, but they cannot fully exploit the rich structure of CDMA signals to minimize interference. This paper presents an iterative technique to estimate multipath parameters which can serve as a preprocessing step in a receiver to increase signal-to-interference ratio. They investigate the performance of the proposed method using computer simu-

lations. Preliminary simulation results show an average of 10 dB gain on channel parameter estimation.

The third approach takes advantage of those digital modulation schemes whose constellation either has constant amplitude (such as BPSK) or is composed of a union of constant amplitude codes (such as QAM). The multistage constant modulus array is capable of separating interference from other users (known as cochannel signals) [26]. They convert a constant modulus (CM) array into a robust smart antenna by modifying the error criterion to be a weighted sum of conventional CM array error and decision-directed equalization error. The new error criterion enables the CM array to (1) separate digital cochannel signals with multipath and inter-symbol interference and (2) track fading signals. Fading signals result from the regions of high and low amplitudes of the standing waves caused by the Doppler shift in received frequencies from a moving transmitter. The key contribution is that the modified error criterion adds phase sensitivity to the otherwise phase insensitive CM error criterion. The UT team presents computer simulations to show the signal tracking properties of the CM array using the modified error criterion in a fading environment.

4.2.5 Filter Design

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

William also plans to construct a static library in that has many useful math functions needed in filter design computations, like an FFT, polynomial expansion, roots finding etc. Eventually this will expand beyond this class of applications. It will for another modular package in the Java Ptolemy ensemble.

4.3 TASK 3: HETEROGENEOUS INTERACTION SEMANTICS

4.3.1 Generalized Hybrid Systems

We have completed a paper [20] that develops the semantics of hierarchical finite-state machines that are composed using various concurrency models, particularly dataflow, discrete-events, and synchronous/reactive modeling. It is argued that all three combinations are useful, and that therefore the concurrency model should be selected independently of the decision to use hierarchical FSMs. In contrast, most formalisms that combine FSMs with concurrency models, such as Statecharts (and its variants) and Hybrid systems, tightly integrate the FSM semantics with the concurrency semantics. We have prototyped an implementation of two of the three combinations described.

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

4.3.2 Type Systems

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

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

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

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

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

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

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

4.3.3 A Partial Order of Models of Computation

Heterogeneity has always been a core principle of the Ptolemy project. Ptolemy supports hierarchically nesting multiple models of computation using a software architecture that has come to be

known as the Wormhole architecture. The strict hierarchy implied by Wormholes and the information hiding between them, however, preclude certain optimization possibilities. We have developed an approach that permits both arbitrary combination of models of computation using wormholes and tighter integration of models of computation. We are implementing this as part of the new Ptolemy kernel written in Java.

Partially ordered sets (posets) of models of computation are specified by the designer of a particular modeling tool (formerly “domain”). At runtime, the model of computation of an application is resolved by selecting the least upper bound of the models of computation of the modules that compose the design.

This process has been implicit in Ptolemy. For example, the dataflow domains are related according to the following order: SDF < BDF < DDF < PN. Thus, an SDF block can run in a PN simulation. In our current design, we take this idea beyond theory and explicitly set a policy for implementation. As designers create and implement new models of computation (both control and additional dataflow), relationships between these models of computation can be specified. However, in the current Ptolemy, there is no systematic way to mix in models of computation that do not follow such a neat total order, such as FSMs.

4.4 GENERAL INFRASTRUCTURE

We have done quite a bit of work that contributes to each of the three tasks without being specifically part of any one of them. This section summarizes that work.

4.4.1 Software Engineering

Christopher Hylands has been exploring using JavaScope, a code coverage tool. Unfortunately, it fails to work with TclBlend, which we are using to construct the test suite. Thus, we cannot perform code coverage tests on the test suite for the Java Ptolemy kernel. We have submitted detailed bug reports.

We downloaded and tested a beta version of Parasoft's CodeWizard for Java. The web site, www.parasoft.com says:

> CodeWizard - Prevent bugs automatically with this easy-to-use source code analysis tool.
> CodeWizard can help you correct poor design strategies, improve software reliability, and enforce coding standards. Available now for C++ and for Java!

Unfortunately, CodeWizard for Java did not install properly, and it cannot deal with Java packages. Christopher Hylands has submitted a bug report.

4.4.2 Ptolemy and Tycho Software

We have made a large number of changes to the Ptolemy and Tycho software supporting the research in this project. This section list a few of these changes.

- Christopher Hylands built a simple syntax-sensitive Java editor in Tycho and an interface between Tycho and a Java applet viewer.
- Cliff Cordeiro has developed a tool for generating and browsing documentation for object-oriented languages, and has built implementations for Itcl and Java. The tool analyzes the source code syntax and relationships between classes and constructs a Tycho information model (TIM) that represents the components and their interrelationships. A viewer compactly displays these relationships, allowing simultaneous multi-resolution views of a set of classes.

- Christopher Hylands has developed support for embedded Tcl/Java code in HTML documents.
- Christopher Hylands fixed a number of portability problems in Tycho so that it now runs reliably on NT machines and some Macintoshes.
- Christopher Hylands wrote oct2tim, a tool for converting the old-style Oct database information for Ptolemy designs into Tycho Information Model (TIM) for use in Tycho. This is a first step in porting existing Ptolemy applications to the new infrastructure.
- Christopher Hylands improved the handling Java files within Tycho under Windows NT so that Java files can now be compiled and executed entirely from within Tycho. A major part of the challenge here was to develop a rational and robust handling the CLASSPATH environment variable that Java depends on.
- Christopher Hylands created a facility in Tycho for generating class inheritance diagrams from C++ source code. This complements the facility we previously wrote in Tycho that generates class diagrams from Itcl source code.
- Mudit Goel completed a modification of Ptolemy that improves the modeling of time in combined discrete-event and dataflow applications. In particular, previously, a dataflow subsystem within a discrete-event simulation had to be supplied with clocking events to trigger its firings. With Mudit's change, the dataflow subsystem can directly specify that it requires repeated periodic firing. This allows it to more naturally model real-time signal sources using dataflow.
- Christopher Hylands added a mechanism to Tycho that will convert a Unix style makefile to a Microsoft Visual C++ makefile. This will help in the construction of a more complete port of Ptolemy to Windows.
- Edward Lee and Seehyun Kim implemented a gridded, multi-column user query widget in Tycho. The first application is a dialog whereby the user guides the selection of fixed-point properties of a signal processing system described with a dataflow graph.
- Brian Evans and Guy Maor (UT Austin, under subcontract) installed a new Matlab interface that removes dependencies on the Matlab 5 shared library in the Ptolemy executables. It also works with Matlab 4.2 and works when Matlab is not present.
- Seehyun Kim wrote some C++ code to collect statistics of inputs, outputs, and states of a functional block in the dataflow domains. Statistics might be used for estimating signal ranges to help in generating a fixed-point implementation of a system. He modified the Tycho editor class EditPtlang to translate a block definition without range estimation into one with range estimation. He has also modified the Ptolemy preprocessor ptlang to support the C++ classes implementing range estimation.
- Christopher Hylands was able to compile and run the DE domain under Tcl8.0 under NT4.0. He used Cygwin18 to compile Tcl8.0b2. He has made some progress using Microsoft Visual C++, but this work remains incomplete.
- Tom Lane (of Structured Software Systems) completed a major redesign of the higher-order functions (HOF) mechanism in Ptolemy. This mechanism is central the ability to specify scalable applications compactly, and may turn out to be central to supporting mutable systems in the future. The previous mechanism was full of bugs concerning its interaction with hierarchy and with the notion of delays on arcs.
- John Reekie added a simple animation facility to the Slate, one that supports linear moving and reshaping of graphical items. The idea is to use this infrastructure in design visualization. An animation is driven by an instance of a new class called Animator; Animator is a sub-class of Interactor, but generates interaction events by itself instead of in response to mouse events. The

animate{} method of the Slate class provides a simple interface to objects of this class.

- Seehyun Kim has created a flexible facility for synthesizing fixed-point implementations of blocks that are specified more generically. This will support more intensive exploration of fixed point designs.
- Christopher Hylands modified the HTML browser in Tycho so that if it is running within Java and the HTML file it is parsing contains an applet, then the applet is displayed. This is done in a separate window because we do not yet know how to map Java frames into Tk windows in a portable way. However, it is done within a single process, not with an external invocation of an appletviewer, and so it provides most of the infrastructure for a tighter integration.
- Christopher Hylands completed the first cut at the Tycho test suite. It is possible to run all the tests with one command.
- Alain Girault has written an automatic star generator for the PN domain that takes as input OC specifications, which are the output from several synchronous languages (Esterel, Lustre, Argos). This provides an interesting experimental infrastructure for exploring the "desynchronization" of synchronous specifications, allowing for distributed implementations. Details are contained in:

  http://ptolemy.eecs.berkeley.edu/~girault/Ocpn/ocpn.html
- Christopher Hylands has constructed a "builder dialog" that can be used to manage installation of Tycho. One consequence is that it becomes unnecessary to bloat the software distribution with automatically generated documentation because the builder dialog will manage generation of the documentation at installation time.
- John Reekie and Sunil Bhahe have constructed a "target" in Ptolemy that generates stand-alone applications that leverage the Tycho user interface infrastructure. The objective is to be able to collect parts of the design infrastructure to create a customized deployed system that includes part of the design software. In the current implementation, a shared library file is created that encapsulates the signal processing part of the system, and the shared library is dynamically linked to running Tycho executable, which then provides the user interface infrastructure. We are working next on mechanisms for extracting the relevant parts of Tycho to provide a minimal, deployable user interface containing exactly what is needed and no more.
- John Reekie and Christopher Hylands have interfaced Tycho to the Java debugger, jdb.
- Bilung Lee has modified Ptolemy to animate finite-state machines models. This should help with debugging and visualization.
- Kevin Chang added a 'diff' capability to Tycho, whereby files can be easily compared. A major use for this is within the version control system.
- The group worked on the compilation interface, whereby Tycho manages Java and C++ compilations.
- Christopher Hylands created mechanisms in Tycho for constructing a graph describing hyperlinks in HTML documents and for exporting postscript from HTML documents. Tycho already has a mechanism for viewing such graphs and exporting gif images and imagemap files that can be used together to create navigation aids on the web.

4.4.3 Support Software

We have also worked with third party software in ways that generically contribute to our project.

- Christopher Hylands ported Glimpse to Windows NT and interfaced it to Tycho. Glimpse is a search engine that efficiently handles large numbers of files. We use in Tycho for searching a source code tree.

- Christopher Hylands set up the htdig search engine so that it will search most of the pages on the http://ptolemy.eecs.berkeley.edu web site. This search mechanism is for our own internal use and has already proved extremely useful for searching the wealth of information we maintain locally.
- We continue to explore alternative Java packages that are available on the net that might provide generic infrastructure for our software development. We have recently discovered JAL from Silicon Graphics, which is like the C++ Standard Template Library. The following is from the SGI page (http://reality.sgi.com/austern/java/index.html):
- "The Java Algorithm Library (JAL), is a collection of generic algorithms for the Java (TM) language; it is modeled on the STL as closely as Java makes possible. The only data structure in Java that can contain both built-in types and user-defined classes is the array, so the JAL is a set of algorithms that operate on one-dimensional arrays."
- We are investigating a package from Bell Labs, called NSBD, Not-So-Bad Distribution, that might help maintain security in distributed Tycho and Ptolemy applications. The home page of NSBD is http://www.bell-labs.com/nsbd
- We obtained an evaluation version of the ICEM CFD Tcl Lint program. This is a modified version of the ICE Tcl compiler. Instead of creating C or bytecode the lint program checks for probable or possible errors in the Tcl code and prints warning and error messages. Unfortunately, this promising utility appears unable to test Tk, Itcl, or Itk code, greatly limiting its usefulness to us.
- We have obtained and evaluated ssh (secure shell) for Windows NT, and believe that it provides adequate security for remote access to Windows NT systems. This means that our Windows NT machines can now be safely integrated into the cluster.

5. Software

Software in the Ptolemy project serves as both a laboratory for experimentation and a mechanism for disseminating results. A new feature of this project is that we expect to be distributing software in the form of smaller packages rather than large monolithic software systems. During the first year of the project we completed two small software releases, Tycho 0.2 and PtPlot 1.0. The first is written in Itcl and the second in Java.

5.1 INFORMATION DISSEMINATION POLICY

The Ptolemy web site, http://ptolemy.eecs.berkeley.edu, is used to distribute all software (including source code) and documentation, together with updated summary sheets, answers to frequently asked questions, and tutorials. We use the most liberal copyright permitted by the University of California, one which has proven effective in promoting technology transfer. A Usenet news group called comp.soft-sys.ptolemy and a mailing list ptolemy-hackers@ptolemy.eecs.berkeley.edu are used to communicate with outside users. Postings to the mailing list are cross-posted to the news group. Postings are archived and searchable from our web site.

5.2 TYCHO

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

using HTML, with an integrated and network capable simplified browser.

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

5.2.1 Tycho 0.2 Release (June 1997)

Tycho 0.2 was released with Ptolemy 0.7. It included some capabilities developed with prior funding, and some developed under this project. In particular, the Java interface is central to this project. Significant new features in this version:

- Java/Tycho interface
- Compilation and dynamic loading of C modules at runtime.
- Improved preferences manager.
- An interface to C,C++ and Java compilers.
- Interfaces to SCCS and RCS revision control systems.
- An interface to the Glimpse index browser, which can rapidly search large directory trees.
- A graphical Tcl profiler.
- A source code documentation system and browser.
- A Tycho Information Model (TIM) architecture.
- A time-slice scheduler for dynamically linked C modules.

5.3 PTPLOT 1.0 AND 1.1

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

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

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

6. Plans for the next year

6.1 MODULAR DEPLOYABLE DESIGN TOOLS

A major part of our work over the next year will be completing and releasing the Java implementation of Ptolemy, with at least a process networks and synchronous dataflow domain. Further efforts will emphasize cleanly incorporating finite-state machine semantics into these domains, following the theoretical developments over the last year [20]. A major still-open question, however, is how to modularize the software. We discuss some of the issues here.

We have had extensive discussions within the group about how to achieve a software architecture

for modular deployable design tools. A key part of the problem is to determine how to reliably create a subset of the developer's software infrastructure to deploy. There is prior art on which to base our design. For example, many Lisp systems include a way to dump the state of the system to disk for restarting later. For example, GNU Emacs includes the function 'dump-emacs'. In Lucid Common Lisp, there was a mechanism called 'tree shaking' which would allow a developer to 'shake' a code tree and get the 'fruit', or the code that was actually used. Franz' Allegro Common Lisp has a similar thing, in which the developer 'trains' the application and then can build a file that contains only the necessary parts of the system. Presumably we could do a similar thing with Tycho, using 'namespace::tycho {info classes}' to see what classes are loaded.

A possible visual aid would be to enhance the class diagram so that the classes that are actually loaded are highlighted. This might help the developer understand the ramifications of particular choices. To do this, we would need to extend class diagram viewer in Tycho so that it would add the proper functionality.

In Ptolemy, the CGC domain (code generation in C) comes fairly close to building deployable stand-alone binaries. Clearly, CGC can build simple non-graphical deployable applications. Two key stumbling blocks to making more complex applications stand-alone are the interactive graphical infrastructure and the makefile infrastructure for compiling code. Also problematic are applications that use shared libraries that might be on the user's system and applications that use proprietary formats, such as Sun .au files for audio.

One question to deal with throughout is what platform we assume a deployable system will be deployed on. How much software infrastructure can be assumed? A Java virtual machine interpreter? A Tcl interpreter? Where processor-specific binaries are required, we need a mechanism to package the minimal set of binaries. Software component technology such as Corba may provide a solution here.

A Java-specific question is whether to focus on creating applets or to focus on finding architectures that support Java applications. The security model built in to applets is essential for certain uses, but it can also be a significant hindrance. Java Beans, another software component technology, provides an alternative that we are investigating.

Concerning the software architecture issues around modular deployable tools, John Reekie contributes the following observations:

- There is always a tension between reusing code by inheritance (inherit and extend the functionality) or by composition (compose objects to obtain more complex functionality). Our designs tend to use a lot of inheritance; deployability implies that the emphasis should shift to composition -- which in turn implies shallower, broader, inheritance trees.
- Java encourages the separation needed for deployability with its interfaces. Using interfaces could allow alternative "lighter" classes to be deployed instead of heavier ones.
- Cleanly staging the "evolution" of a design from UI to execution will aid deployment at one of a number of stages. At each, it should be clear what is required to deploy and what is not. For example, it should be possible to deploy and execute a system that contains no scheduler; implication: the run-time data structures contain a reference to a schedule, but not a reference to a scheduler.
- Incremental changes need to be supported at each stage -- the amount of "stuff" required at each deployable stage depends on degree of mutability needed. It should, for example, be possible to say "I want, at run-time, to substitute this subgraph with this other subgraph" without having to deploy an SDF scheduler. The semantics of mutability at run-time should therefore not be defined as graph mutations.

- A deployed executable can be as light as the preceding stages can make it. It does not, for example, have to actually contain a graph. The framework for deployed applications needs to support the concepts of execution, control, monitoring, and debugging without being dependent on a graph structure.

6.2 DOMAIN-SPECIFIC DESIGN TOOLS

We are focusing here on domain-specific techniques that are relevant to analog circuit and MEMS, particularly lumped parameter modeling. We are particularly interested in the interaction of such models with foreign models, such as finite-state machines that may be modeling discrete controllers. Standard methods for numerical integration, such as Runge-Kutta, appear at first glance to be incompatible with such foreign interaction. We are investigating both the mathematics of simulation and the software architecture questions, and hope to be able to provide within the Java Ptolemy infrastructure continuous models.

We have also had extensive discussions within the group about the implications of dynamically modifying the topology of applications that are specified using the various models of computation that we use that are based on graphs representing the interconnection of modules. The cost of such mutations depends on the amount of static analysis done prior to execution of the application. Thus, the cost savings (in overhead) of static analysis are offset by the cost of mutations for dynamically changing topologies. There are, however, mechanisms for supporting limited graph mutations without incurring extensive run-time cost. Hierarchically combining FSMs (finite-state machines) with synchronous dataflow can (usually) be done in such a way that the FSMs effect mutations without demanding reanalysis of the dataflow graph. We are further evaluating the expressiveness of this particular combination, since it seems to be a promising approach for adaptive signal processing applications.

6.3 HETEROGENEOUS INTERACTION SEMANTICS

A major part of the effort here will be to bring into practice (via the Java Ptolemy implementation) the considerable theory that we have developed for integrating finite-state controllers hierarchically with concurrency models [20]. In addition, we plan to continue work on the type-system concepts, and also implement a process-level type system in Java Ptolemy.

7. Technology Transfer

One of the notable properties of the Ptolemy project is its track record of demonstrable transfer of technology to industry leaders in the computer-aided design and defense industries. This is accomplished via a careful cultivation of industry contacts and a strategy of wide open, very liberal distribution of software and publications. All software is made available on the Web with the most liberal copyright notice permitted by the University of California. This notice retains ownership of the copyright, but expressly grants permission to use the software for any purpose, including development of commercial products. It is distinctly more liberal than the GNU public license, and thus better represents “free software.”

Although it is still too early in this project for major results of the project to have been transferred, there are some ongoing interactions that we wish to highlight.

7.1 HEWLETT-PACKARD INTEGRATES PTOLEMY WITH ANALOG SIMULATION

On June 2, 1997, Hewlett-Packard’s EEsof Division announced plans to deliver a comprehensive digital signal processing (DSP) design system as part of its effort to broaden its solutions for the elec-

tronic design automation (EDA) industry. In their June 2, 1997, press release, HP EEsof states:

> "Built into the HP DSP Designer software is a new simulation technology developed by merging HP research and technology with the University of California at Berkeley Ptolemy project. This new simulation engine facilitates cosimulation of time, frequency and data flow technologies and significantly expands the DSP development capability for mixed RF/analog/DSP communications projects."

The software is comprised of two new DSP tools - DSP Designer and DSP Synthesis. It is part of HP's newly introduced HP Advanced Design System, which includes the latest versions of its highly regarded RF and analog circuit simulation technology. The complete press release and a related article from *EE Times* are available on the Ptolemy Web site. Some of the major additions to Ptolemy are:

* Runs On NT 4.0, NT 3.51 and Win 95
* Added time and frequency simulation and modeling
* Added DSP filter tool
* Added VHDL modeling and simulation
* Added Verilog modeling and simulation
* Added 300-400 time and frequency models
* Added Spice and Harmonic Balance cosimulation....

HP has integrated their HF Spice, Harmonic Balance and Circuit Envelope simulators into HP Ptolemy. Their simulation executable links two distinct simulator software architectures: one based on Ptolemy (HP Ptolemy) and the other incorporating the HP EEsof analog simulators (Gemini). In HP Ptolemy there currently are two simulation domains: SDF and TSDF. In TSDF, HP has modified the semantics of SDF to introduce a notion of time to apparently make it easier to combine DSP simulations with analog simulations. Using the TSDF domain a user can embed a circuit simulator into a dataflow simulation using an interface very similar to a Ptolemy WormHole.

This interaction is particularly relevant because of its integration of Ptolemy's dataflow modeling with HP's continuous-time modeling of analog and RF circuits. The fundamental theory techniques behind this kind of mixture are a major part of this project.

7.2 LOCKHEED-MARTIN AND BERKELEY TARGET CONFIGURABLE HARDWARE

Sanders, a Lockheed-Martin company, has joined forces with the Ptolemy project to develop a capability in Ptolemy to synthesize application for configurable logic (such as FPGAs). This project is just starting.

7.3 POLIS — A CODESIGN SYSTEM BASED ON PTOLEMY

The POLIS team at Berkeley and Cadence (headed by Professor Alberto Sangionvanni-Vincentelli) has been cooperating with the Ptolemy project to develop hardware/software codesign technology. They recently announced the public availability of the POLIS-0.3 co-design environment for control-dominated embedded systems. POLIS, which is based on Ptolemy, offers an integrated interactive environment for specification, co-simulation, formal verification, and synthesis of embedded systems implemented as a mix of hardware and software components.

Most of the information about POLIS, including pointers to source and object code (for various CPUs and OSes) is available at the WEB site:

http://www-cad.eecs.berkeley.edu/~polis

The POLIS team consists of Felice Balarin, Massimiliano Chiodo, Alberto Ferrari, Paolo Giusto, Harry Hsieh, Attila Jurecska, Marcello Lajolo, Luciano Lavagno, Claudio Passerone, Claudio Sansoe', Ellen Sentovich, Marco Sgroi, Kei Suzuki, Bassam Tabbara, Reinhard von Hanxleden, and Alberto Sangiovanni-Vincentelli.

7.4 PTOLEMY MINICONFERENCE

We held a miniconference at Berkeley that reviewed major accomplishments of the Ptolemy project and introduced this new effort. The objectives of the conference were primarily to report to and solicit advice from the industrial sponsors and friends of the project. The miniconferences was held in conjunction with Berkeley's annual Industrial Liaison Program (ILP) conference.

7.4.1 Second Ptolemy Miniconference — March 14, 1997

This miniconference reviewed the previous DARPA-funded effort in the Ptolemy project, which was at the stage of wrapping up, and future plans and preliminary results under this new DARPA effort. The conference included several outside speakers reporting on uses of Ptolemy software and techniques plus ongoing interactions. We had 58 attendees from the following organizations:

- Adaptec
- Advanced Fibre Communications
- Advantest
- Alta Group of Cadence Design Systems
- Angeles Design Systems
- Berkeley Design Technology
- Data Flow Systems
- Ericsson Radio Systems AB
- Hewlett Packard
- Hughes Aircraft
- Hughes Space and Communications
- LG Electronics
- Lockheed-Martin
- Motorola
- National Semiconductor
- NEC
- Nortel
- Rockwell International
- Sanders, a Lockheed Martin Company
- Seiko Epson Corp.
- Semiconductor Research Corporation
- Seoul National University
- Sony
- Structured Software Systems
- Sun Microsystems
- Synopsys

- Tektronix
- Thomson-CSF
- University of Pittsburg
- University of Texas, Austin
- University of Washington
- White Eagle Systems

The highlights of the conference included:

- The first public demonstration of hierarchical finite-state machines combined with dataflow and discrete-event concurrency models.
- The first public demonstration of a synchronous/reactive modeling environment that supports hierarchical heterogeneity.
- The first public demonstration of Tycho, our user-interface development environment, interacting with Java and with Ptolemy.
- The first public demonstration of Web-based simulators for programmable DSPs, from UT Austin.
- The first public description of an investment analysis tool from Structured Software Systems, based on Ptolemy.

The miniconference also included descriptions of the use of Ptolemy in modeling free-space optoelectronic systems (from the University of Pittsburg), a description of Myrnet network simulations in Ptolemy (from Sanders), the use of Ptolemy for VHDL-based circuit design, research on multidimensional signal processing models, and theory that we have developed to help us understand interacting models of computation. In addition, we outlined plans for future work including a strategy for supporting fixed-point design and our plans for Java-based design. The proceedings of the conference are at:

http://ptolemy.eecs.berkeley.edu/papers/viewgraphs/miniconf97/

8. Acknowledgments

8.1 PARTICIPANTS AT BERKELEY

8.1.1 Principal investigator

- Edward A. Lee

8.1.2 Professional staff

- Christopher Hylands
- Fiona Sinclair
- Mary Stewart

8.1.3 Post-doctoral researchers

- Praveen Murthy
- John Reekie
- Dick Stevens (from NRL)

8.1.4 Graduate students

- Cliff Cordeiro
- John Davis, II
- Stephen Edwards
- Ron Galicia
- Mudit Goël
- Michael Goodwin
- Bilung Lee
- Jie Liu
- Praveen K. Murthy
- Neil Smyth
- Michael C. Williamson
- William Wu
- Yuhong Xiong

8.2 CORPORATE SUPPORT

8.2.1 Sponsors

The following organizations have contributed additional financial support for the Ptolemy project:

- the State of California MICRO program
- The Alta Group of Cadence Design Systems
- Hitachi
- Lockheed Martin ATL
- NEC
- Philips
- Rockwell
- the Semiconductor Research Corporation (SRC)

9. Publications

Following is a list of publications with significant content developed under this project (or in a couple of cases, content developed largely during the proposal phase of this project).

9.1 JOURNAL ARTICLES

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

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

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

[4] M. Torlak, G. Xu, B. L. Evans, and H. Liu, "Fast Estimation of Weight Vectors to Optimize Multi-Transmitter Broadcast Channel Capacity," IEEE Transactions on Signal Processing, Jan. 1998.

[5] M. Torlak, G. Xu, B. L. Evans, and H. Liu, "Fast Estimation of Weight Vectors to Optimize Multi-Transmitter Broadcast Channel Capacity," *IEEE Transactions on Signal Processing*, to appear, Jan. 1998.

9.2 CONFERENCE PAPERS

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

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

[8] S. Kim and E. A. Lee, “Infrastructure for Numeric Precision Control in the Ptolemy Environment”, *Proceedings of the 40th Midwest Symposium on Circuits and Systems*, August 3-6, 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/fixedpointInfra)

[9] D. Arifler, C. Duong, B. L. Evans, S. K. Marwat, C. M. Moy, and A. Yuan, ‘‘A Configurable, Portable, Extensible Framework for Web-Enabled Interactive Simulation of Software for Embedded Programmable Processors,’’ *submitted*.

[10] A. K. Kulkarni, A. Dube, and B. L. Evans, ‘‘Benchmarking Code Generation Methodologies for Programmable Digital Signal Processors,’’ *submitted*.

[11] B. Lu, B. L. Evans, and D. V. Tosic, ‘‘Simulation and Synthesis of Artificial Neural Networks Using Dataflow Models in Ptolemy,’’ Invited Paper, *Proc. IEEE Conf. on Neural Network Applications in Engineering*, Sep. 8-9, 1997.

[12] G. E. Allen, D. C. Schanbacher, and B. L. Evans, "Real-Time Sonar Beamforming on a Unix workstation Using Process Networks and Pthreads", Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc., submitted.

[13] M. Torlak, G. Xu, B. L. Evans, and H. Liu, "Estimation of Optimal Weight Vectors for Broadcast Channels", Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc., Apr. 17-20, 1997, Munich, Germany.

[14] M. Torlak, B. L. Evans, and G. Xu, "Blind Channel Estimation in CDMA Systems with Aperiodic Spreading Sequences," Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, Nov. 3-5, 1997.

[15] S. Gummadi and B. L. Evans, "Cochannel Signal Separation in Fading Channels Using a Modified Constant Modulus Array", Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc., submitted.

[16] G. E. Allen, D. C. Schanbacher, and B. L. Evans, "Real-Time Sonar Beamforming on a Unix workstation Using Process Networks and Pthreads", *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.*, submitted.

[17] 4. M. Torlak, G. Xu, B. L. Evans, and H. Liu, "Estimation of Optimal Weight Vectors for Broadcast Channels", *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.*, Apr. 17-20, 1997, Munich, Germany.

[18] M. Torlak, B. L. Evans, and G. Xu, "Blind Channel Estimation in CDMA Systems with Aperiodic Spreading Sequences," Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, Nov. 3-5, 1997.

[19] S. Gummadi and B. L. Evans, "Cochannel Signal Separation in Fading Channels Using a Modified Constant Modulus Array", *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.*, submitted.

9.3 TECHNICAL REPORTS

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

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

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

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

9.4 PHD THESES

[24] S. A. Edwards, “The Specification and Execution of Heterogeneous Synchronous Reactive Systems,” **Ph.D. thesis**, University of California, Berkeley, May 1997. Available as UCB/ERL M97/31. (http://ptolemy.eecs.berkeley.edu/papers/97/sedwardsThesis/)

10. References

[25] T. W. Parks, "Bounded Scheduling of Process Networks," Technical Report UCB/ERL-95-105, Ph.D. Dissertation, EECS Department, University of California, Berkeley, CA 94720, Dec. 1995.

[26] J. J. Shynk, A. V. Keerthi, and A. Mathur, "Steady state analysis of the multistage CM array," IEEE Trans. on Signal Processing, vol. 44, pp. 948-962, 1996.