SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2004; **34**:653–673 (DOI: 10.1002/spe.585)

A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools

Anthony Sulistio, Chee Shin Yeo and Rajkumar Buyya*,†

Grid Computing and Distributed Systems (GRIDS) Laboratory,
Department of Computer Science and Software Engineering, University of Melbourne,
Carlton, VIC 3053, Australia

SUMMARY

In recent years, extensive research has been conducted in the area of simulation to model large complex systems and understand their behavior, especially in parallel and distributed systems. At the same time, a variety of design principles and approaches for computer-based simulation have evolved. As a result, an increasing number of simulation tools have been designed and developed. Therefore, the aim of this paper is to develop a comprehensive *taxonomy* for design of computer-based simulations, and apply this taxonomy to categorize and analyze various simulation tools for parallel and distributed systems. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: taxonomy; simulation tools; parallel system; distributed system

1. INTRODUCTION

In recent years, *parallel and distributed systems* (*PDSs*) have emerged as viable solutions to meet the ever increasing needs for computational power and data management capability. These systems offer speedup in computational performance that is necessary to support computationally intensive grand challenge applications, such as weather forecasting and earthquake analysis.

Designing a PDS is a complex and challenging task that involves many issues [1,2]. Some of the issues include resource management, network performance, security, heterogeneity, fault tolerance, adaptability, scalability, concurrency and transparency. In short, on designing a PDS one needs to address more complicated issues related to both the parallel execution model and the distributed architecture which do not exist in the sequential execution model and centralized architecture.

*Correspondence to: Rajkumar Buyya, Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, ICT Building, 111 Barry Street, Carlton, VIC 3053, Australia.
†E-mail: raj@cs.mu.oz.au

Received 12 November 2002
Revised 12 September 2003
Accepted 12 September 2003

Published online 16 April 2004

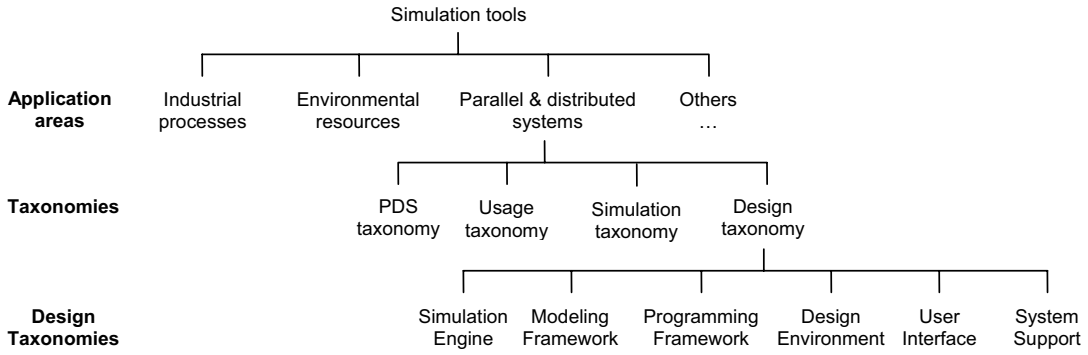

Figure 1. Categorization of simulation tools.

Therefore, researchers need to ensure that their newly designed systems are feasible and can perform as expected before proceeding on with the actual development.

Simulation has been researched and applied successfully to model real world processes, applications and objects. It enables the study of various issues, such as feasibility, behavior and performance without building the actual system, thus saving precious time, cost and effort. A simulation can be adjusted to run at any speed relative to the real world and according to various possible scenarios. The results gathered from the simulation indicate how the real system behaves, thus enabling researchers to understand and improve on their design without the actual implementation.

However, applying simulation to model problems is also a non-trivial task. Good simulation models are difficult to design and maintain. The design and development costs for complex simulation models are sometimes comparable to the costs of building the actual systems. In addition, not all researchers are experts in simulation, thus they have problems creating simulation models successfully. Therefore, there is a need to have effective simulation tools that enable easy and fast creation of simulation models.

Extensive research conducted in the area of simulation over recent years has produced a number of simulation tools for modeling PDSs. Simulation tools support the creation of *repeatable* and *controllable* environments for feasibility study and performance evaluation. These simulation environments facilitate researchers, educators and students to conduct effective research, teaching and learning with ease. For instance, Bricks [3], GridSim [4], MicroGrid [5] and SimGrid [6] are constructed to simulate emerging Grid computing [1] environments since it is difficult to have easy access to readily available Grid testbeds. In addition, it is extremely hard to evaluate their performance under different scenarios since the availability of resources changes with time and the users have different requirements in a Grid computing environment.

Figure 1 provides a categorization of simulation tools. There are many application areas where simulation has been applied, one of which is the design and evaluation of PDSs. Simulation tools for PDSs are then further differentiated by four taxonomies: *PDS taxonomy* identifies the type of target systems to be simulated, *usage taxonomy* illustrates how the tool is used, *simulation taxonomy*

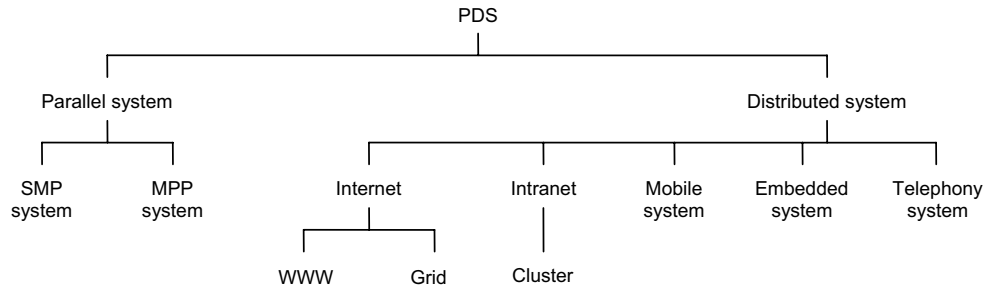

Figure 2. PDS taxonomy.

highlights the characteristics of the simulation, and *design taxonomy* describes the components and features of the simulation tools. The next few sections discuss these taxonomies in greater detail.

A survey of some of the existing research-based simulation tools examine their designs and architectures based on the taxonomy. Only research-based simulation tools are selected since their theoretical concepts, design and architectures are published and readily available, unlike commercially released simulation tools. Therefore, this paper provides researchers with an in-depth understanding of building simulation tools for PDSs and the current research trends. Researchers can extract useful and relevant information and experience encountered in the development of the existing tools to learn, improve and design better tools. There is also a possibility of extending or reusing some of the existing simulation tools to prevent replication of work.

2. TAXONOMY

The proposed taxonomy provides classification of the simulation tools into categories based on different aspects and properties. This section describes four sub-taxonomies: PDS, usage, simulation and design taxonomies, and examples are given of simulation tools to support the taxonomies.

2.1. PDS taxonomy

The categorization of *PDS taxonomy* is shown in Figure 2. A *parallel system* consists of multiple processors in close communication, normally located within the same machine. Multiple processors can share the same memory or access their own local memory. A parallel system provides increased throughput and reliability.

A parallel system can be classified into symmetric multiprocessing (SMP) systems and massively parallel processing (MPP) systems. A SMP system has multiple processors sharing a common memory and operating system. A MPP system has multiple processors accessing their own local memory and operating system. A MPP system is more difficult to program for parallelism compared with a SMP system since each processor accesses each local memory separate from other processors. However, a

MPP system is more scalable in terms of the number of processors, since a SMP system is limited to a maximum number of processors.

A *distributed system* spreads out the computation among autonomous computers, which are physically distributed in general. Each distributed computer accesses its own local memory and communicates with other computers through networks such as the Internet. A distributed system supports both resource sharing and load sharing, and is fault-tolerant through the replication of devices and data in separate physical locations. It also provides a good price/performance ratio.

A distributed system can be further sub-divided into five types: Internet, intranet, mobile systems, embedded systems or telephony systems. The Internet is a heterogeneous network of machines and applications that is implemented using the Internet protocol (IP). The World Wide Web (WWW) is built on top of the Internet architecture to share information, identified through a uniform resource locator (URL). A Grid [1] is a very large-scale distributed system that can scale to Internet-size environments with machines distributed across multiple organizations and administrative domains.

An Intranet is a local area network (LAN) that is usually privately owned by an organization. Access of the intranet is internal within the organization and firewalls guard the interface for external access to the Internet. A cluster [2] is a collection of distributed machines that collaborate to present a single integrated computing resource to the user.

A mobile system is implemented using wireless network protocol so that it can function while it is on the move at different physical locations. Some examples of mobile systems are cellular phone systems, handheld devices and laptop computers with a wireless LAN. An embedded system is a combination of hardware and software designed to function as a type of application device. Some examples of embedded systems are industrial machines, medical equipments, automobiles and aircrafts. A telephony system transmits voice and data using digital transmission through telephone networks. Some examples of telephony systems are asymmetric digital subscriber line (ADSL), integrated services digital network (ISDN), intelligent networks and advanced intelligent networks.

2.2. Usage taxonomy

A simulation tool can be used as a simulator or an emulator as shown in Figure 3. A simulator is a tool that can model and represent the actual system. Simulation runs at any speed relative to the real world and saves information for the entire simulation to facilitate analysis of the simulated behavior of the actual system. Conversely, an emulator is a tool that acts like the actual system. Emulation executes like the actual system itself and is useful for accurate and reliable testing without having the real system. For example, the Grid simulation tools studied in this paper are simulators, except MicroGrid, that emulate Globus-based Grid applications.

2.3. Simulation taxonomy

In general, a *simulation* comprises three properties as shown in Figure 4. Presence of time indicates whether the simulation of a system encompasses the time factor. A static simulation does not have time as part of the simulation, in contrast to a dynamic simulation. Basis of value specifies the values that the simulated entities can contain. A discrete simulation has entities only possessing one of many values within a finite range, but a continuous simulation has entities possessing one of many values within an infinite range. Behavior defines how the simulation proceeds. A deterministic simulation has

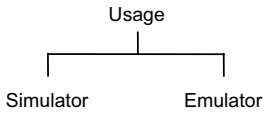

Figure 3. Usage taxonomy.

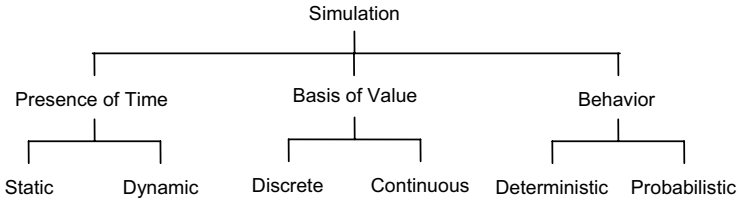

Figure 4. Simulation taxonomy.

no random events occurring, so repeating the same simulation will always return the same simulation results. In contrast, a probabilistic simulation has random events occurring, so repeating the same simulation often returns different simulation results.

Every simulation can be classified based on these three properties. An example of a dynamic, discrete and probabilistic simulation is to generate a path that a data packet moves from a source host to a destination host in a network. However, simulating the path of a missile given an initial firing velocity and fixed wind resistance is an example of a dynamic, continuous and deterministic simulation. A chess game simulation comprises static, discrete and probabilistic properties.

2.4. Design taxonomy

The *design* taxonomy categorizes simulation tools based on desired components and features that are necessary to provide simulation of a PDS. These features and components not only provide a framework for understanding the design of existing simulation tools, but also provide possible ways of improving the design.

2.4.1. Simulation engine

Each simulation tool exploits a *simulation engine* to model and execute the simulation model. Figure 5 shows the taxonomy for the simulation engine. Simulations can be executed in serial or parallel modes. A serial or sequential simulation is executed using a single processor, while a parallel or distributed simulation is executed using multiple processors, located in PDSs. A serial simulation is restricted by limited memory and needs a longer execution time, compared with a parallel simulation. Therefore, there is an increasing interest in using parallel simulation for modeling complex, large-scale systems.

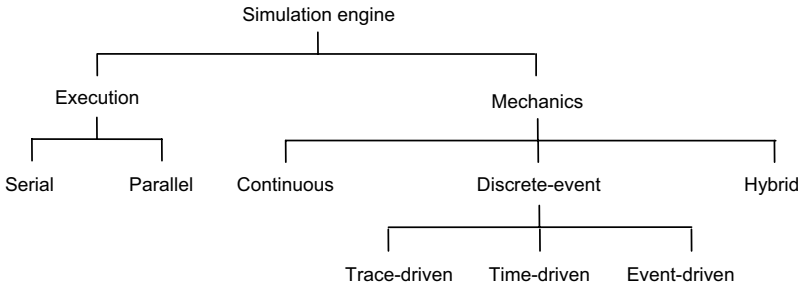

Figure 5. Simulation engine taxonomy.

Simulation models that are created for serial execution cannot be executed using parallel execution. In other words, the user can only design simulation models based on the type of execution mode supported by the simulation tool.

Most existing simulation tools use serial simulation. This is because simulation tools that use a parallel simulation engine are more difficult to implement. However, there is a growing trend to implement tools that exploit parallel simulation given the growing demand for faster simulation and shorter execution time. Some tools such as Parsec [7] and GloMoSim [8] support both serial and parallel simulation, which provides the flexibility for the user to use.

The simulation tool advances the simulation based on the mechanics defined in the simulation engine. In a continuous simulation, state changes occur continuously across time. In a discrete-event simulation (DES), state changes only occur at specific time intervals. A hybrid simulation comprises both continuous and discrete-event simulations.

A DES adopts a queuing system where queues of events wait to be activated. A DES is further subdivided into a trace-driven, time-driven or event-driven simulation. A trace-driven DES proceeds by reading in a set of events that are collected independently from another environment and are suitable for modeling a system that has executed before in another environment. The user can trace and modify the inputs to observe and control the simulation. A time-driven DES advances by fixed time increments and is useful for modeling events that occur at regular time intervals. An event-driven DES advances by irregular time increments and is useful for modeling events that may occur at any time. An event-driven DES is more efficient than a time-driven DES since it does not step through regular time intervals when no event occurs. Most of the simulation tools surveyed in this paper use an event-driven DES because it is relevant for the context of simulating most large-scale PDSs and requires less time. However, hybrid simulation is needed by embedded system simulators such as Ptolemy II [9].

2.4.2. *Modeling framework*

The *modeling framework* depicts how the user models the target PDS in the simulation tool. An incorrect modeling framework results in inaccurate modeling and thus produces useless simulation

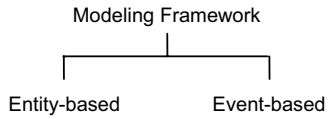

Figure 6. Modeling framework taxonomy.

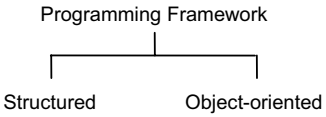

Figure 7. Programming framework taxonomy.

results that differ tremendously from the actual target system. Figure 6 shows the taxonomy for the modeling framework.

An entity-based modeling framework represents processes to be modeled as entities. Each entity performs its own tasks and communicates with other entities via messaging. In an event-based modeling framework, each task in a modeled process is activated via the arrival of some triggering events. It is possible to have a modeling framework that implements both entity and event. All surveyed simulation tools apply both entity-based and event-based modeling frameworks, since both frameworks reflect real-world happenings of actual objects.

2.4.3. Programming framework

The *programming framework* determines the programming paradigm that the user needs to be familiar with in order to use the simulation tool and can thus affect the learning curve of the user. Figure 7 shows the taxonomy for the programming framework.

A structured programming framework implements a top-down structured program design with control passing down the modules in a hierarchy. An object-oriented programming framework expresses the program as a set of objects that communicate with one another to perform tasks. The object-oriented framework is easier to create, maintain and reuse compared with the structured programming framework. However, the structured framework normally incurs less runtime overheads than the object-oriented framework.

A fair share of simulation tools use the structured and object-oriented programming framework. Some examples of tools that use structured programming are Parsec, SimGrid and Ptolemy II, while some examples of tools that use object-oriented programming are SimJava [10], NS-2 [11] and GridSim.

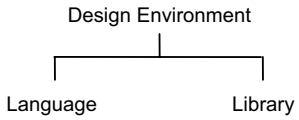

Figure 8. Design environment taxonomy.

2.4.4. *Design environment*

The *design environment* determines how the user uses the tool to design simulation models. A good design environment facilitates easy learning and fast usage. Figure 8 shows the taxonomy for the design environment.

A language provides a set of defined constructs for the user to design simulation models, while a library provides a set of routines to be used with a supporting programming language. A library-based simulation tool normally gives the user more flexibility in creating and controlling the simulation. An experienced user of the supporting programming language may fine-tune and optimize the simulation by exploiting certain libraries. A language-based simulation tool usually hides low-level implementation details from the user and thus provides less flexibility. Therefore, a language-based tool needs to provide a complete set of well-defined constructs to ensure that it supports the required level of flexibility. A language-based tool is also often easier to learn and use since it is more high-level compared with a library-based tool.

Examples of tools that use a language-based design environment are SimOS [12], NS-2 and Bricks, while tools that use a library-based design environment are GridSim, SimJava and GloMoSim. Some tools such as Parsec and Ptolemy II provide both a language-based and library-based design environment.

2.4.5. *User interface*

The *user interface* determines how the user interacts with the simulation tool directly. Figure 9 shows the taxonomy for the user interface. A visual user interface is preferred over a non-visual interface because graphical displays enable better interaction and they are easier to use.

A visual design interface allows the user to create a simulation model much easier and faster compared with a non-visual interface. The user can build the simulation model by dragging and dropping simulation objects and configuring the attributes and values by using forms. In contrast, a typical non-visual design interface requires the user to write program codes which require more time and effort. Examples of tools that provide a visual design interface are Parsec, GridSim and Ptolemy II.

A visual execution interface provides a better representation of the simulation process that enables the user to observe and analyze the simulation better. Animations provide a good visualization and display the flow of the simulation. Graphs give the graphical version of statistical data captured from the simulation. Without a visual execution interface, the user encounters difficulties in analyzing and

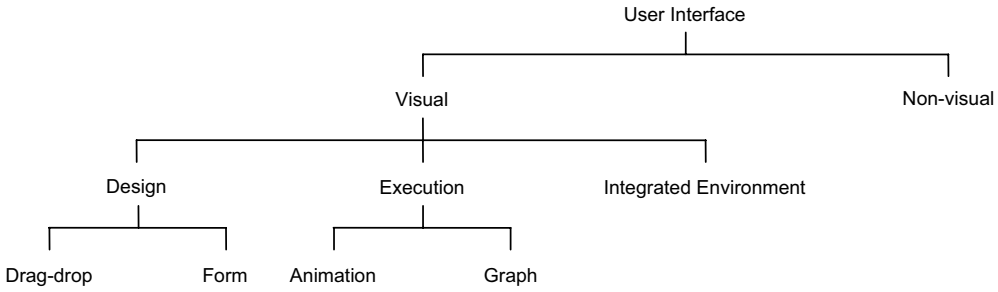

Figure 9. User interface taxonomy.

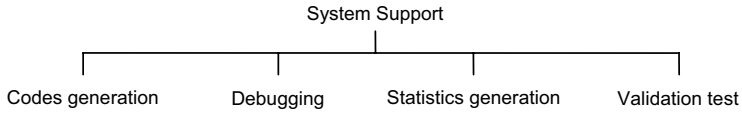

Figure 10. System support taxonomy.

understanding the simulation results based on the huge amount of statistics and events captured in pure text format. For example, NS-2 is able to generate animations; Ptolemy II is able to generate graphs; and SimJava is able to generate both animations and graphs.

A visual integrated environment provides both visual design and execution interfaces. The user conveniently uses the simulation tool to develop simulation models and execute them without exiting the tool. This integrated environment concept is borrowed from integrated development environment (IDE) tools developed for programming. None of the surveyed tools provide a complete visual integrated environment. Most tools have plans to incorporate a visual integrated environment in the future to enable better usability, but implementing a good user interface is not trivial and requires lots of time and effort. This is why most simulation tools are not able to provide a visual interface.

2.4.6. System Support

System support provides useful and ready-to-use features that help the user to build an accurate and successful simulation model. Figure 10 shows the taxonomy for system support.

The codes generation feature produces the resulting simulation program code automatically for the user. This feature normally exists in simulation tools that are language based or use a visual design interface. This allows the user to create the simulation model using a high-level language or simpler visual user interface without worrying about writing complex program code. Tools such as GridSim and Parsec support this functionality.

Table I. Categorization of simulation tools based on their simulated systems.

Category	Tool	Organization	Key similarities and differences, simulated systems, and Web site
Parallel systems	SimOS	Stanford University, U.S.A.	• Models complete computer systems through fast simulation of hardware and levels of abstraction. • Simulates a complete multiprocessor system and studies all various aspects including hardware architecture, operating system and application programs. • http://simos.stanford.edu/
Distributed systems	SimJava	University of Edinburgh, U.K.	• Provides a core set of foundation classes for simulating discrete events. • Simulates distributed hardware systems, communication protocols and computer architectures. • http://www.dcs.ed.ac.uk/home/simjava/
Networks	NS-2	University of California at Berkeley, U.S.A.	• Supports several levels of abstraction to simulate a wide range of network protocols via numerous simulation interfaces, such as using scripting language and/or system language. • Simulates network protocols over wired and wireless networks. • http://www.isi.edu/nsnam/ns/
	Parsec	University of California at Los Angeles, U.S.A.	• Uses a portable runtime kernel that executes simulations on either sequential or parallel architectures enhanced by ready support of numerous parallel simulation protocols. • Simulates very large scale integrated (VLSI) parallel architectures, parallel databases and wireless networks using parallel simulation. • http://pcl.cs.ucla.edu/projects/parsec/
Mobile systems	GloMoSim	University of California at Los Angeles, U.S.A.	• Provides an extensible and modular library that supports implementation of alternative protocols for each layer of the wireless communication protocol stack. • Simulates large-scale wireless mobile networks. • http://pcl.cs.ucla.edu/projects/glomosim/
Grid scheduling systems	Bricks	Tokyo Institute of Technology, Japan	• Provides simulation for resource allocation strategies and policies for multiple clients and servers as in global computing systems in a Grid environment. • Simulates resource scheduling algorithms in Grids. • http://matsu-www.is.titech.ac.jp/~takefusa/bricks/
	GridSim	University of Melbourne, Australia	• Supports simulation of space-based and time-based, large-scale resources in the Grid environment. • Simulates economy-based resource scheduling systems in Grids. • http://www.gridbus.org/gridsim/

Table I. Continued.

Category	Tool	Organization	Key similarities and differences, simulated systems, and Web site
Grid scheduling systems (continued)	MicroGrid	University of California at San Diego, U.S.A.	• Runs emulations by executing actual application code on the virtual Globus Grid and thus requires more time to complete the application. • Emulates the Globus Grid environment for resource management. • http://www-csag.ucsd.edu/projects/grid/
	SimGrid	University of California at San Diego, U.S.A.	• Simulates a single or multiple scheduling entities and time-shared systems operating in a Grid computing environment. • Simulates distributed Grid applications for resource scheduling. • http://grail.sdsc.edu/projects/simgrid/
Embedded systems	Ptolemy II	University of California at Berkeley, U.S.A.	• Builds upon a component-based design methodology that hierarchically integrates multiple models of computation to capture different design perspectives. • Simulates systems that comprise heterogeneous components and sub-components. • http://ptolemy.eecs.berkeley.edu/ptolemyII/

Debugging facilities enable the user to identify abnormalities and problems in the simulation model more easily and correct them. For example, SimOS and Ptolemy II provide some debugging facilities. Statistics related to the simulation generated by the tool allow the user to analyze and justify the simulation model. Without a built-in statistics generation feature, the user will need to write extra separate modules to manually track the simulation execution. Examples of tools that provide statistics generation are SimJava and GloMoSim. The validation test feature is able to help the user identify probable errors or inconsistencies in the simulation model, thus supporting more accurate simulations. For instance, the next release of NS-2 is expected to incorporate validation functionalities.

3. SIMULATION TOOLS SURVEY

A wide variety of tools have been developed to support the simulation of PDSs by various researchers around the world. They include SimOS, SimJava, NS-2, Parsec, GloMoSim, Bricks, GridSim, MicroGrid, SimGrid, and Ptolemy II. This section briefly describes these simulation tools by highlighting their key similarities and differences, and the categorization based on their simulated systems (as noted in Table I). In addition, the proposed taxonomy discussed in the previous section is being applied to the tools to identify their design characteristics and possible innovations needed (as noted in Table II). This section also provides a design comparison for similar tools (that are under the same category as shown in Table I) in order to differentiate them.

Table II. Design of simulation tools based on the proposed taxonomy.

Design	SimOS	SimJava	NS-2	Parsec	GloMoSim
Simulated systems	Parallel systems	Distributed systems, networks	Wired and wireless networks	VLSI circuits, wireless networks, parallel architectures	Wireless mobile networks
Usage	Simulator	Simulator	Simulator	Simulator	Simulator
Simulation	Static, discrete, deterministic	Static, discrete, deterministic	Static, discrete, deterministic	Static, discrete, deterministic	Static, discrete, deterministic
Simulation engine	Parallel, event-driven DES	Serial, event-driven DES	Serial, event-driven DES	Serial & parallel, event-driven DES	Serial & parallel, event-driven DES
Modeling framework	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based
Programming framework	Structured	Object-oriented	Object-oriented	Structured	Structured
Design environment	Language	Library	Language	Language, library	Library
User interface	Non-visual	Animation, graph	Animation	Drag-drop, form	Non-visual
System support	Debugging, statistics generation	Statistics generation	Debugging, statistics generation, validation test	Code generation	Statistics generation

3.1. SimOS

SimOS [12] allows simulation of complete computer systems, including a full operating system and all application programs that run on it. It can be used to study the computer architecture such as the effects of new processors and memories on workloads such as large scientific applications and commercial database systems. It can also be used to study the operating systems like developing, debugging and performance tuning an SMP operating system for multiprocessors.

SimOS runs as a layer between the host machine and the target machine. The host refers to the hardware and software on which SimOS runs, while the target refers to the simulated hardware, its operating system and applications running on it. Each simulated hardware component in the SimOS layer has multiple implementations that vary in speed and detail. SimOS is capable of running large and complex commercial applications available on the SGI platforms by adopting various simulation modes.

SimOS includes a detailed simulator that runs in a loop, fetching, decoding, and simulating the effects of instructions on the machine's register set, caches and main memory. The simulator

Table II. Continued.

Design	Bricks	GridSim	MicroGrid	SimGrid	Ptolemy II
Simulated systems	Grid, resource scheduling systems	Grid, resource scheduling systems	Grid, resource scheduling systems	Grid, resource scheduling systems	Embedded systems
Usage	Simulator	Simulator	Emulator	Simulator	Simulator
Simulation	Static, discrete, deterministic	Static, discrete, deterministic	Dynamic, continuous, deterministic	Static, discrete, deterministic	Dynamic, continuous, deterministic
Simulation engine	Serial, event-driven DES	Multithreaded, event-driven DES	Parallel, event-driven DES	Serial, trace-driven DES	Serial, hybrid
Modeling framework	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based	Entity-based, event-based
Programming framework	Object-oriented	Object-oriented	Structured	Structured	Object-oriented
Design environment	Language	Library	Language	Library	Language, library
User interface	Non-visual	Form	Non-visual	Non-visual	Drag-drop, form, graph
System support	Statistics generation	Code generation, statistics generation	N/A	N/A	Code generation, debugging, statistics generation

uses a parallel pipeline model that provides hardware simulators ranging from very approximate to highly accurate models and records more detailed performance information at the expense of longer simulation time. SimOS maps the actual hardware such as CPU, memory and disk as equivalent entities in the simulation. Interaction events occurring in the simulated hardware are captured and classified to facilitate analysis of results after the simulation.

SimOS is implemented in *C* using a structured programming framework. It is designed as a language with a set of pre-defined constructs that the user can use for simulation. The user uses Tool Command Language (*Tcl*) [13] as a specification language for SimOS for the simulated hardware and controlling the runtime of SimOS execution. Tcl is a powerful and easy to learn scripting language so the user can set up SimOS simulations easily if they are experts in Tcl.

SimOS uses the GNU project debugger (*GDB*) [14] to provide full interactive debugging of the simulated operating system. SimOS has sophisticated statistics collection and classification mechanisms to generate statistics about the simulated behaviors and map these statistics back to the user-defined groups, such as processes, subroutines or transactions. SimOS was last released as version 2.0 in 1998.

3.2. SimJava

SimJava [10] provides a set of foundation classes to enable easy creation and animation of DES models by using *Java*. It can be applied to model distributed systems and networks. Three Java-based libraries are implemented to support the simulation. The first library, called *simjava*, provides the building constructs for text-only Java simulation and incorporates the event-driven DES engine. The users write object-oriented Java programs to call the simjava library constructs to build the simulation. The second library, called *simanim*, provides an applet template that is tightly coupled with the simjava library to create visual animations of the text-only simulation. The third library, called *simdiag*, is a collection of JavaBeans that enables graphical display of simulation results.

SimJava models the simulation objects as entities that communicate by sending passive event objects to other entities via ports. Detailed statistics generation is provided in the simjava library so that the user can indicate which simulation events are collected and written into a trace file. Many other tools use SimJava as the foundation libraries for simulation, one of which, GridSim, that is discussed later in this section. The last release of SimJava is version 2.0, which dates back to September 2002.

3.3. NS-2

NS-2 [11] is a DES that enables simulation of Transport Control Protocol (TCP), routing and multicast protocols over wired and wireless networks. NS-2 allows network researchers to study and evaluate specific network protocols under varying network conditions, which is essential to understand their behaviors and characteristics.

NS-2 [15] is an object-oriented simulator implemented using two languages, *C++* and Object Tcl (*OTcl*). It supports a class hierarchy in C++ called *compiled hierarchy* and a similar class hierarchy for OTcl called *interpreted hierarchy*. The user creates new simulation objects using the OTcl interpreter and each of these objects is mirrored by a corresponding object in the compiled hierarchy.

NS-2 uses two languages to support different purposes. C++ is fast to run but slow to modify, making it suitable for detailed simulated protocol implementation. OTcl runs much more slowly but can be modified more quickly and interactively and is used by the user for setup and configuration. NS-2 operates as a scripting language where simulations are executed by running a simulation script defined in OTcl. The scripts contain calls to predefined topologies provided in a comprehensive library of network protocols.

NS-2 does not have a visual design interface which means that an inexperienced user is likely to encounter difficulties in creating simulations given NS-2's high complexity. NS-2 uses a visual execution interface called *Nam* [15]. Nam is a Tcl/TK based animation tool for viewing network simulation traces and supports topology layout and packet level animation. It also provides various data inspection tools and statistics generation.

NS-2 supports some basic debugging tools. For example, memory debugging is supported using the *dmalloc* library, Tcl-level debugging is supported using a Tcl debugger and C++-level debugging is supported using GDB. The next version of NS-2 is to include automatic validation tests to help NS-2 users to check and verify their simulation models. The current release of NS-2 is version 2.26, which was released in February 2003.

3.4. Parsec

Parsec [7] aims to provide a parallel simulation environment that is able to model large-scale complex systems such as VLSI parallel architectures, parallel databases and wireless networks. The goals of the environment are to provide an easy mapping of simulation models to executable programs, support a wide range of parallel simulation protocols, and enable visual model design. The implemented parallel simulation environment comprises Parsec, the parallel simulation language, its graphical user interface (GUI) called Parsec Visual Environment (*Pave*) and a portable runtime kernel that implements the simulation models.

Parsec adopts the process-interaction approach for event-driven DES. A physical process is modeled as an entity and events passed among the processes are modeled as messages. Parsec provides a structured parallel simulation language that provides a set of simple syntax to define simulation models. The Parsec simulation models are then generated into operational *C*-based simulation codes. It also provides a *C++* library called *Compose* that can interface with native written C++ codes written by the user to execute parallel simulations.

Pave enables the visual design of simulation component libraries and the construction of simulation models from these components through a simple visual framework. Pave allows the user to drop flowchart notations to represent control flow and message operations. In addition, Pave generates the Parsec codes for the simulation models and optimizes them for parallel execution. However, Pave does not provide debugging or statistics generation facilities.

The portable run-time kernel can execute Parsec simulation models on sequential and parallel machine architectures. However, the user has to explicitly program the synchronization required for parallel execution. Parsec has been used to design other parallel simulation libraries such as GloMoSim, which is surveyed in the following section. Parsec was re-released as commercial software in 2001 and is currently maintained by Scalable Network Technologies [16].

3.5. GloMoSim

GloMoSim [8] is designed to support simulation of very large wireless mobile networks with several thousands of nodes. It uses parallel DES to reduce the time required for executing the detailed mobile simulations.

GloMoSim is a library-based simulator that simulates a specific wireless communication protocol in the protocol stack. It is developed using Parsec. Parsec enables new protocols and functions to be programmed and added to the GloMoSim library. GloMoSim is a library built on top of Parsec and thus it is useful for the GloMoSim user to know Parsec in order to build additional protocols and modules. GloMoSim uses Parsec as the simulation engine and is able to support both serial and parallel DES. GloMoSim models mobile network nodes as Parsec entities.

GloMoSim is implemented using a layered approach that enables rapid integration of models developed at different layers by different users. Each layer is treated as a module and is invoked via receiving message events from other layers. This layered design supports modularity and encapsulates the complexity of a specific layer from other layers. Each layer also keeps track of relevant simulation statistics and generates statistics at the end of the simulation. GloMoSim serves as a basic library for wireless simulations that can be customized or extended using Parsec for specific requirements. There is no need to have a visual user interface for GloMoSim since it is easily configured via an input

text file and executes at the command prompt. The last release of GloMoSim, version 2.0, was released in 2000 before Parsec became commercialized.

3.6. Bricks

Bricks [3] is a performance evaluation system that allows analysis and comparison of various scheduling schemes in a high-performance Grid computing environment. Bricks simulates various behaviors of Grid computing systems such as the behavior of networks and resource scheduling algorithms.

Bricks operates as a DES of a job scheduling system in virtual time. It consists of two components: the simulated *Global Computing Environment* that models the Grid and the *Scheduling Unit* that coordinates the simulation behaviors of Grid computing systems. Bricks models the Global Computing Environment to consist of three entities: *clients* that represent user machines that have jobs to run, *servers* that represent resources available to run the jobs, and *networks* that represent the network between clients and servers. Events of the entities form the communication required to drive the simulation.

Bricks is designed using an object-oriented framework and implemented in *Java*. The Scheduling Unit uses Java interfaces to support various scheduling algorithms and components. Bricks provides a Bricks script language that enables the user to set up the configuration and parameters of the Global Computing Environment. The user uses the building 'bricks' within the script to test and evaluate a variety of simulations in a deterministic manner based on the statistics collected. No current work seems to be in progress for Bricks since it is not available for download.

3.7. GridSim

GridSim [4] supports modeling and simulation of heterogeneous Grid resources (both time- and space-shared), users, applications, brokers and schedulers in a Grid computing environment. It provides primitives for creation of application tasks, mapping of tasks to resources, and their management so that resource schedulers can be simulated to study the scheduling algorithms involved.

GridSim adopts the multi-layered design architecture. The first bottom layer is the portable and scalable *Java* interface and runtime environment called Java virtual machine (JVM), whose implementation is available for single and multiprocessor systems including clusters. The second layer is SimJava which provides an event-driven discrete event infrastructure on top of the JVM to drive the simulation for GridSim. The third layer is the GridSim toolkit, which provides the modeling and simulation of core Grid entities such as resources and information services using the discrete event services defined by the second layer. The fourth layer provides the simulation of resource aggregators called Grid brokers or schedulers. The last top layer focuses on application and resource modeling with different scenarios to evaluate scheduling and resource management policies, heuristics, and algorithms.

SimJava provides the event-driven DES engine for GridSim. The Grid resources, users and brokers are modeled as entities, and they communicate via messaging events. To create the simulations, the user writes Java programs that call the GridSim libraries to model the different Grid elements. The Visual Modeler (VM) [17] provides the visual design interface for GridSim. VM is a form-based tool that enables the user to create Grid resources, users and applications and specifies their property values.

VM also generates the Java program codes for executing the simulation. GridSim has a *GridStatistics* library that the user can call to collect simulation statistics automatically. Version 2.1 of GridSim was released in July 2003 and continuing work promises to provide better simulation features and functionalities.

3.8. MicroGrid

MicroGrid [5] implements a virtual grid infrastructure to run a *Globus* [18] application to support systematic design and evaluation of middleware, applications and network services for the computational Grid. MicroGrid models the Globus Grid middleware infrastructure, and aims to support scalable simulation of Grid applications using a wide variety of scalable clustered resources. It provides a realistic Grid software environment that emulates the applications to run with identical Application Program Interfaces (APIs) and allows the MicroGrid user to configure Grid resource performance attributes.

The simulation engine for MicroGrid, called *MaSSF*, is built on top of the parallel DES engine Dartmouth Scalable Simulation Framework (*DaSSF*) [19]. MaSSF is able to provide detailed network emulations that model the behaviors of each IP packet. MicroGrid models Grid resources as computing entities and these computing entities interact via communication entities. MicroGrid adopts the structured programming framework and is implemented in C. The design environment of MicroGrid resembles a language since MicroGrid provides a limited but simple set of user commands that enables easy emulation of Globus applications in a Globus-based simulation environment. No visual user interface is provided for MicroGrid and the user has to manually modify the Makefile of an existing Globus application to add a link to MicroGrid to run the simulation. The latest release of MicroGrid, version 2.2.1 in February 2003, has been successfully tested with Globus 2.2 toolkit.

3.9. SimGrid

SimGrid [6] is a toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational Grid environment. SimGrid aims at providing the right model and level of abstraction for studying Grid-based scheduling algorithms and generates correct and accurate simulation results.

The first version [20] of SimGrid is now known as *SG*. SG is a low-level simulator that enables one to build flexible and general simulations. Using SG to simulate a distributed application where scheduling decisions are taken by different entities may be more difficult. The newest version of SimGrid is called *MSG*, which is a higher-level simulator built using the SG, which is more application-oriented and builds simulation with multiple scheduling agents.

In SG, resources such as processing units and network links are treated as unrelated resources, so no interconnection topology is imposed between the resources. This provides maximum flexibility for SG to simulate a wide range of computing environments as the user can specify topologies based on the application-specific requirements. SimGrid describes scheduling algorithms in terms of agent entities that make scheduling decisions. These agents interact by sending and receiving events via communication channels.

SimGrid is implemented in C and provides predefined SG and MSG API libraries that enable the user to manipulate data structures for Grid resources and tasks. A SG Grid resource comprises a name, a set of defined performance-related metrics and trace values for each metric. SimGrid uses trace-driven simulation based on these trace values gathered from real Grid resources to simulate a more realistic representation of the resources in terms of fluctuating performance rather than constant performance. A SG task comprises a name, a cost, which keeps track of the cost of performing the task, and a state, which reflects the status of the task. These data structures provide maximum flexibility for the SimGrid user, but also require higher user expertise. However, SimGrid does not provide any of the system support facilities as discussed in the taxonomy. The current release of SimGrid is version 2.15, which was released in May 2003.

3.10. Ptolemy II

Ptolemy II [9] emphasizes the modeling and design of embedded systems. It is built upon a component-based design methodology to support hierarchical integration of an extensible range of simulation models. The user uses Ptolemy II to capture, understand and manipulate different design perspectives of an embedded system, which comprises heterogeneous components across different domains. Ptolemy II supports different models of computation [21], consisting of discrete-event (*DE*), continuous-time (*CT*), synchronous dataflow (*SDF*), finite state machines (*FSMs*), communicating sequential processes (*CSPs*), process networks (*PNs*) and distributed discrete-event (*DDE*). These models of computation enable the Ptolemy II user to define different interaction mechanisms between the components.

Ptolemy II uses a serial simulation engine that supports hybrid simulation to simulate the heterogeneous components in an embedded system. Concurrent models of computation, such as CSP, PN and DDE, are thus implemented via threads. Components are modeled as entities called actors and these entities message via interfaces called ports. Ptolemy II uses object-oriented technologies to implement its component-based design.

Ptolemy II serves as both a language and a library. Ptolemy II acts as an architecture design language that focuses on modeling the interaction among a set of components in an embedded system. It also has a comprehensive algebra-based expression language for specifying the values of parameters in components. Ptolemy II is implemented in *Java* and comprises libraries. This enables the user to write Java code that uses Ptolemy libraries directly to create the simulation model.

Ptolemy II provides a GUI called *Vergil* that enables the user to create, view and modify Ptolemy models by dragging and dropping component-based building blocks. Vergil also uses forms to get specific values of each component from the user. Vergil represents Ptolemy II models using an Extensible Markup Language (*XML*) schema called Modeling Markup Language (*MoML*). Ptolemy II uses a plot package that creates applets to plot graphs for the statistical results. The visual design and execution interfaces are not integrated together and have to be invoked separately.

Vergil provides basic debugging facilities to detect errors in the Ptolemy II models. Statistical data of the simulation are collected in Ptolemy II and can be displayed graphically by calling the plot package. A code generation facility [22] for Ptolemy II is currently under development to enable transformation of Ptolemy II models into simulation program codes for more efficient execution. The current release of Ptolemy II, released in May 2003, is version 3.0-beta which provides a more comprehensive actor library.

3.11. Design comparison for similar tools

As seen in Table I, this paper discusses several simulation tools for the same category of PDS: networks and Grid scheduling systems. Therefore, there is a need to highlight the core design differences between these similar tools as follows.

- *Networks.* NS-2 is a discrete event simulator that provides extensive support for simulation of various wired and wireless network protocols, such as TCP, routing and multicast protocols. The user is able to create simulations using NS-2 by writing scripts to access predefined network topologies predefined in a comprehensive library of network protocols. Parsec is able to support both serial and parallel execution of DES models for VLSI circuits, networks and parallel architectures. Parsec provides a set of functions that enable the user to create network simulation. Alternatively, the user can use Parsec to create new simulation libraries suitable for specific applications.
- *Grid scheduling systems*. Bricks is used in simulating client–server global computing systems that provide remote access to scientific libraries running on high-performance computers and is designed based on a centralized global scheduling methodology. GridSim is able to simulate different classes of heterogeneous resources for a Grid environment that includes both time-based and space-based large-scale resources. It can also be used to simulate application schedulers for single or multiple administrative distributed computing domains such as clusters and Grids. MicroGrid is modeled on the Globus Grid computing environment to facilitate execution of Globus applications in a controlled virtual Grid emulated environment. MicroGrid's emulation returns more precise results as actual application code is executed in the virtual Grid and thus requires more execution time. The need to construct Globus applications for emulating new scheduling algorithms also imposes more development overheads. Therefore, the MicroGrid emulator is helpful for testing already built Globus applications. SimGrid supports modeling of resources that are time-shared. The first version of SimGrid, SG, is limited to a single scheduling entity and time-shared system, so it is difficult to simulate multi-user systems in a Grid environment. Therefore, the new version of SimGrid, MSG, was introduced to enhance the low-level SG simulator so that multiple scheduling entities can be simulated.

4. CONCLUSION

This paper has presented a taxonomy for computer simulations and applied the taxonomy on simulation tools for PDSs. The taxonomy comprises PDS, usage, simulation and design taxonomies. The design taxonomy has emphasized respective components and features of a simulation tool, including the simulation engine, modeling framework, programming framework, design environment, user interface and system support. A number of selected simulation tools have been surveyed using the taxonomy. This paper thus helps to identify some approaches for designing simulation tools for PDSs and possible future research directions.

There are many different approaches and methods for developing simulation tools for modeling PDSs. With the increased complexity and scale of simulated systems, there is a growing trend for simulation tools to use parallel simulation engines in order to speed up the simulation process.

Future simulation tools are likely to support both language-based and library-based design environments so that the user has greater flexibility and power to build customized and extensible simulations. Simulation tools are also evolving into visual integrated environments where the user can build, execute, monitor and analyze simulations in a single complete environment. Simulation tools are also incorporating useful system support capabilities to assist the user in building faster and more accurate simulation models.

ACKNOWLEDGEMENTS

The authors would like to acknowledge all researchers of the simulation tools described in this paper and thank them for their outstanding work. We also thank Srikumar Venugopal and anonymous reviewers for their comments on this paper.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.
2. Barker M, Buyya R. Cluster computing at a glance. *High Performance Cluster Computing*, vol. 1, Buyya R (ed.). Prentice-Hall: Upper Saddle River, NJ, 1999; 3–47.
3. Takefusa A, Matsuoka S, Nakada H. Overview of a performance evaluation system for global computing scheduling algorithms. *Proceedings 8th IEEE International Symposium on High Performance Distributing Computing (HPDC8)*, Redondo Beach, CA, August 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999; 97–104.
4. Buyya R, Murshed M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1175–1220.
5. Song HJ, Liu X, Jakobsen D, Bhagwan R, Zhang X, Taura K, Chien A. The MicroGrid: A scientific tool for modeling computational Grids. *IEEE Supercomputing (SC2000)*, Dallas, TX, 4–10 November 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
6. Legrand A, Marchal L, Casanova H. Scheduling distributed applications: The SimGrid simulation framework. *Proceedings 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003)*, Tokyo, Japan, 12–15 May 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003.
7. Bagrodia R, Meyer R, Takai M, Chen Y, Zeng X, Martin J, Park B, Song H. Parsec: A parallel simulation environment for complex systems. *IEEE Computer* 1998; **31**(10):77–85.
8. Zeng X, Bagrodia R, Gerla M. GloMoSim: A library for parallel simulation of large-scale wireless networks. *Proceedings 12th Workshop on Parallel and Distributed Simulations (PADS98)*, Banff, Alberta, Canada, May 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998; 154–161.
9. Liu X, Liu J, Eker J, Lee EA. Heterogeneous modeling and design of control systems. *Software-Enabled Control: Information Technology for Dynamical Systems*, Samad T, Balas G (eds.). Wiley-IEEE Press: New York, 2003.
10. Howell F, McNab R. SimJava: A discrete event simulation package for Java with applications in computer systems modeling. *First International Conference on Web-based Modeling and Simulation*, San Diego, CA, January 1998. SCS Press: San Diego, CA, 1998.
11. Breslau L, Estrin D, Fall K, Floyd S, Heidermann J, Helmy A, Huang P, McCanne S, Varadhan K, Xu Y, Yu H. Advances in network simulation. *IEEE Computer* 2000; **33**(5):59–67.
12. Rosenblum M, Bugnion E, Devine S, Herrod SA. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Society* 1997; **7**(1):78–103.
13. Tcl Developer Site. http://www.tcl.tk/ [July 2003].
14. GNU Project Debugger. http://www.gnu.org/software/gdb/ [July 2003].
15. Fall K, Varadhan K (eds.). The NS manual. *Technical Report*, University of California at Berkeley, June 2003.
16. Scalable Network Technologies. http://www.scalable-networks.com/ [July 2003].
17. Sulistio A, Yeo CS, Buyya R. Visual modeler for Grid modeling and simulation (GridSim) toolkit. *Innovative Solutions for Grid Computing Workshop, Proceedings of the International Conference on Computational Science (ICCS 2003)*, Melbourne, Australia, 2–4 June 2003 (*Lecture Notes in Computer Science*, vol. 2659). Springer: Berlin, 2003; 1123–1132.

18. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(2):115–128.
19. Cowie J, Liu H, Liu J, Nicol D, Ogielski A. Towards realistic million-node internet simulations. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA99)*, Las Vegas, NV, 1999. CSREA Press: Las Vegas, NV, 1999.
20. Casanova H. Simgrid: A toolkit for the simulation of application scheduling. *Proceedings 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2001)*, Brisbane, Australia, May 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
21. Bhattacharyya SS *et al.* Ptolemy II: Heterogeneous concurrent modeling and design in Java. *Technical Report*, University of California, Berkeley, August 2002.
22. Tsay J. A code generation framework for Ptolemy II. *Technical Report*, University of California, Berkeley, May 2000.

