

USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

2

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES
(AIRMICS)

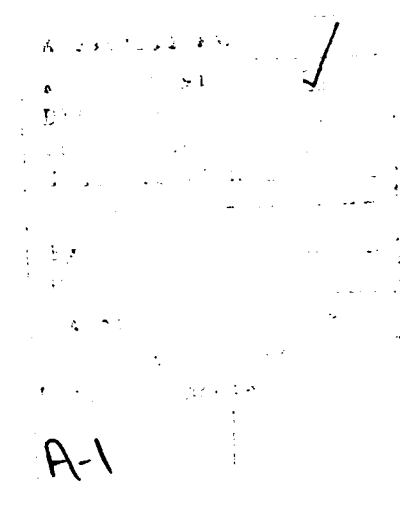
AD-A236 544



Specification and Analysis of Parallel Machine Architecture

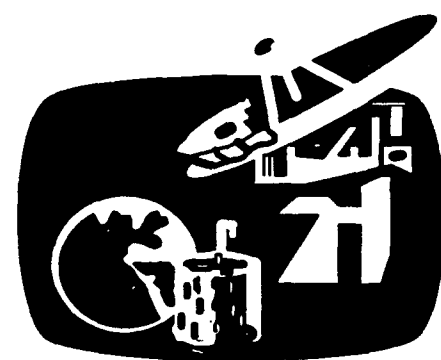
(ASQB-GC-90-011)

17 March 1990



A-1

115 O'Keefe Bldg
Georgia Institute of Technology
Atlanta, GA 30332-0800



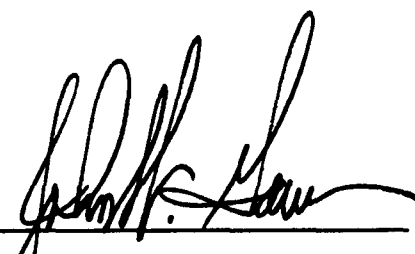
91-01900



91 011 178

This research was sponsored by the Army Institute for Research in Management Information, Communications, and Computer Science (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). The objective of this research was to develop an environment in which a software application, target hardware architecture, and resource allocation strategy can be defined, simulated and evaluated. The motivation for this research was to provide ISC, particularly ISSC, methods for testing the performance of a given application or set of applications on a set of candidate host computers. The tools designed under this project can be used to evaluate the performance of large Ada programs running on POSIX compliant operating systems, as well as less conventional computer architectures. The first year results reported herein include a survey of existing methods/tools and the design of software and parallel architecture representation language. Based on the survey, the Berkeley Requirements Statement Language (BRSL) was extended making it suitable for the high level specification and analysis for both software and hardware of parallel machine architecture. This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

SI/ 
John W. Gowens
Division Chief
CNSD

SI/ 
John R. Mitchell
Director
AIRMICS

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS NONE	
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT N/A	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A				
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A	
6a. NAME OF PERFORMING ORGANIZATION AIRMICS		6b. OFFICE SYMBOL (If applicable) ASQB-GC	7a. NAME OF MONITORING ORGANIZATION N/A	
6c. ADDRESS (City, State, and Zip Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800			7b. ADDRESS (City, State, and ZIP Code) N/A	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS		8b. OFFICE SYMBOL (If applicable) ASQB-GC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N/A	
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	PROJECT NO. DY10-01-01
			TASK NO. 01	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Specification and Analysis of Parallel Machine Architecture				
12. PERSONAL AUTHOR(S) C. V. Ramamoorthy				
13a. TYPE OF REPORT Research		13b. TIME COVERED FROM 4/89 TO 3/90		15. PAGE COUNT 20
14. DATE OF REPORT (Year, Month, Day) 90,03,17				
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUBGROUP		
			Parallel Architecture Software Specification	
			Parallel Machines Hardware Specification	
			Specification Languages Berkeley Requirements Statement Language (BRSL)	
			Resource Allocation Strategies	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
The objective of this research was to develop an environment in which a software application, target hardware architecture, and resource allocation strategy can be defined, simulated and evaluated. The motivation for this research was to provide ISC, particularly ISSC, methods for testing the performance of a given application or set of applications on a set of candidate host computers. The tools designed under this project can be used to evaluate the performance of large Ada programs running on POSIX compliant operating systems, as well as less conventional computer architectures. The first year results reported herein include a survey of existing methods/tools and the design of software and parallel architecture representation language. Based on the survey, the Berkeley Requirements Statement Language (BRSL) was extended making it suitable for the high level specification and analysis for both software and hardware of parallel machine architecture.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL LTC Joseph M. Hanratty			22b. TELEPHONE (Include Area Code) (404) 894-3136	22c. OFFICE SYMBOL ASQB-GC

FINAL REPORT

C.V. Ramamoorthy

Computer Science Division

Department of Electrical Engineering and Computer Science

University of California

Berkeley, CA 94720

(Term: April 1989 - March 1990)

(Contract No. DAKF11-89-C-0016)

Summary of the report

The main tasks for the first year of research (April 1989 - March 1990) consist of the survey of existing methods/tools and the design of software and parallel architecture representation language. Based on the survey, we extended our BRS� system so that it is suitable for the high level specification and analysis for both software and hardware of parallel machine architecture. Our approach uses one common specification language incorporating both of software and hardware aspects in order to make the specification and analysis consistent and easy.

During this fiscal year, we developed BRS� to be used as specification language. Hardware description languages were fully surveyed to extend our BRS� to accommodate the specification of hardware aspects of parallel machines including task allocation strategy. The prototype has been developed and is now being extended to be fully expressible all the aspects of software and hardware of parallel systems. At this stage, we are concentrating on the high level hardware description which can be used to describe both types of shared memory and message passing parallel machines. More detailed language description capability that discriminate the tightly coupled vs. loosely coupled system is also being considered. For this purpose, basic research for the classification of various parallel architectures is done.

In this report, we describe BRS� specification language, Distributed simulation which can be used during simulation to speed up the simulation process, and Network Event Manager which is to be used for monitoring the simulation process and for analysis of the simulation.

Report on Specification and Analysis of Parallel Machine Architecture

C.V. Ramamoorthy

Computer Science Division

Dept. of Electrical Engineering and Computer Science

University of California, Berkeley

1. Introduction

Recent years we have seen the introduction of many parallel machines in the research field and the commercial market. This was made possible by the remarkable advancement in the device technology, and was also needed to provide the enough processing power required by the application programs which are becoming larger and more complex. But we also have seen the proliferation of many different architectures without a single dominant architecture. Unfortunately each architecture has some advantages and some disadvantages and our experience with these parallel machines is very limited. This situation puzzles not only a designer when he has to make decisions on how to design a parallel machine but also a customer or user when he has to decide which parallel machine to purchase or wants to know how to use it more efficiently. Therefore we need a method to evaluate and compare many different parallel architectures.

One approach is to analyze and operate existing parallel machines and compare the data collected. But this approach is not practical to most users with limited resources and time, let alone a designer who does not even have any real machines yet. More realistic and economic way will be to describe the parallel architectures under consideration with a language or other description tools, analyze and simulate them to gather information on their static and dynamic characteristics. For the conventional machines which have a single processor mostly, it suffices to describe the architecture, analyze it, and simulate some synthetic benchmark programs on it. But when we execute the same task on parallel machines which have many processors and many memory modules, we face new problems which did not exist in a single processor machine. One problem is the resource allocation, deciding which processor gets which process and which memory module stores which data fragment. Depending upon this resource allocation policy, the parallel machines exhibit a wide range of characteristics. Another problem stems from the

complexity of the application programs running on the parallel machines. Those programs consist of many processes and the interactions among these processes through communication are complicated. Therefore although it may be an immediate goal to specify, analyze, and simulate the parallel architectures, the ultimate goal should be a design of an integrated tool with which we can specify, analyze, and simulate parallel architectures, resource allocation policies, and application programs altogether. Our environment is described in Figure 1.

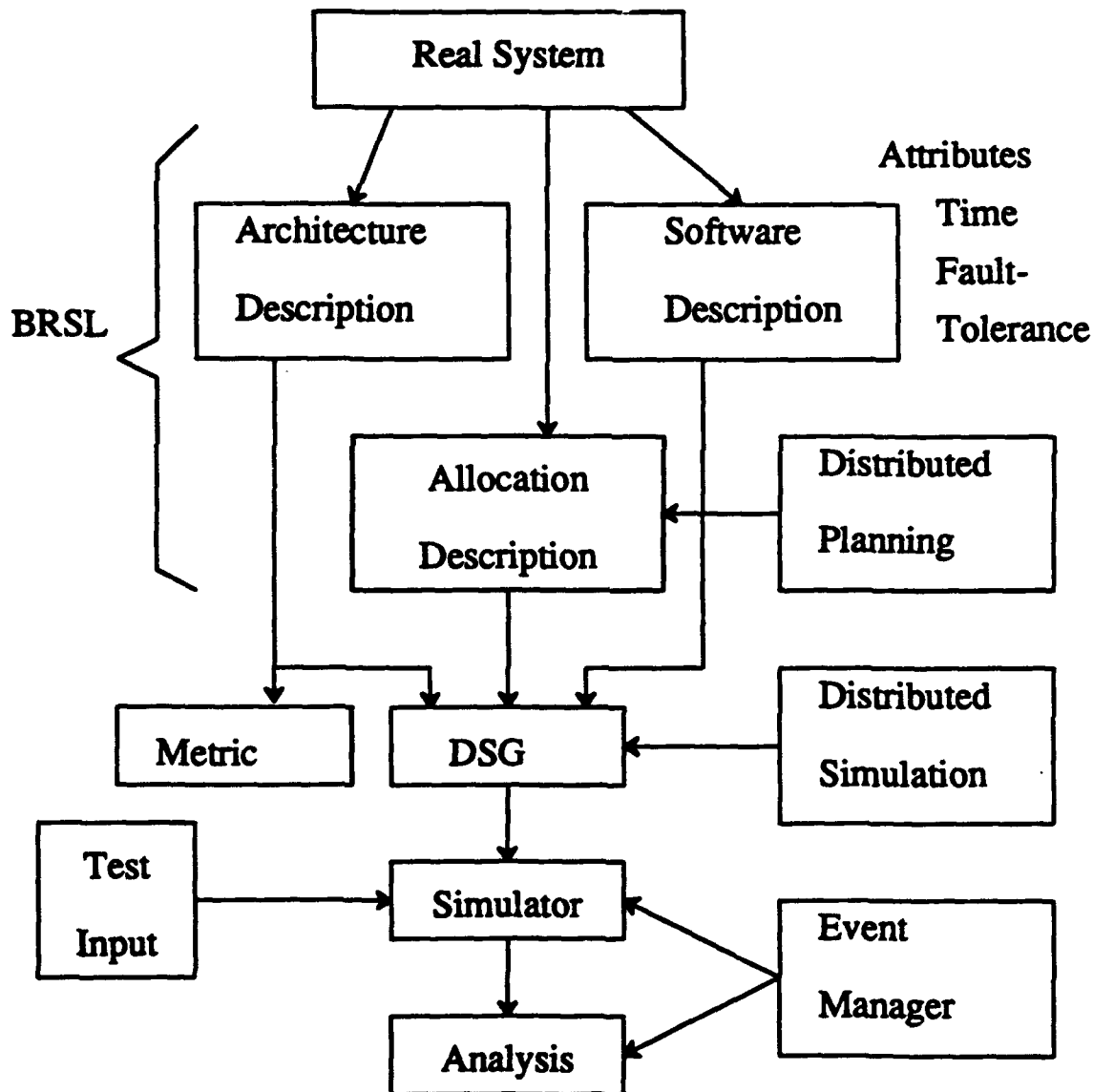


Figure 1. Specification, analysis and simulation environment

With this integrated environment we first specify and analyze hardware and software separately, then evaluate the whole system with the resource allocation strategy specified. Distributed simulation can be used to speed up simulation process during simulation. In our environment, we design the use of Network Event Manager to monitor the simulation process and to analyze the simulation result.

There are already many hardware description languages and analysis tools available but most of them fall in one of following two categories. One category is languages and tools for integrated circuit synthesis and its level of description can be instructional level, register transfer level, switching circuit level, circuit level. Some examples are Instruction Set Processor (ISP) [BAR77], Computer Design Language (CDL) [BAR75], A Hardware Programming Language (AHPL) [HIL75], ESIM, VIESIM, SPICE, PISCES. But none of them are suitable for our purpose because the level of detail is too low, hence it is very difficult and takes too much effort to specify and analyze parallel architectures with them. The other category comprises the languages and tools for the description of distributed environments [PON86]. Here each host is represented by an abstract single processor machine. So many important features of parallel machines cannot be explained.

Realizing inappropriateness of existing languages and tools, we have developed new languages and tools suitable for our purpose by extending Berkeley Requirement Statement Language (BRSL) to specify and analyze parallel architectures. The level of specification detail is very flexible, and there exist analysis and simulation tools.

The organization of the report is as follows. Section 2 describes the survey of existing tools and compare with our environment. In section 3 we introduce BRSL, explains how to specify parallel architectures with BRSL, and describe how and what to analyze with the specification. Section 4 describes the distributed simulation which is to speed up the simulation process. Network Event Manager to be used for monitoring and analysis of simulation is described in section 5. We conclude the report by identifying future work.

2. Survey of existing tools

In this section, we describe the survey of tools developed or proposed. Along with the survey of existing tools, we put much efforts on identifying important metrics in evaluation of parallel systems.

2.1. Tools

We briefly describe the survey on hardware description languages for detailed specification of integrated circuits and 6 simulation tools for performance analysis.

2.1.1. Hardware description languages

There have been many hardware description languages and analysis tools mainly for the purpose of synthesis of integrated circuits. In the order of degree of abstraction level, they can be categorized into circuit level, gate level, register transfer level, instruction level, and finally system level. The VHSIC Hardware Description Language (VHDL) [1] allows design and documentation of digital circuit from system level down to the gate level. VHDL system uses event-driven model to simulate MOS circuits. One example of instruction level language is Instruction Set Processor (ISP)[2]. Computer Design Language (CDL)[3], A Hardware Programming Language (AHPL)[4], and Digital Systems Design Language (DDL)[5] describe at the level of register transfer logic. AHPL and DDL also describe hardware at circuit level.

Although the above mentioned languages provide strong capability to describe hardware environment, their level of description is too low to be used to specify and analyze parallel architectures especially in conjunction with software description. Therefore, we don't consider this category of systems.

2.1.2. Distributed Execution Analysis Suite (DEA-suite)[6]

The system is to be used as a design tool in the early phase of the development process for performance analysis. Basically, it requires two inputs, high level description on application based on dataflow model and high level description on underlying hardware system such as connectivity of each node, etc. Task allocation and scheduling policies on each node are automatically selected from the provided library. However, the system proposed has several weaknesses as follows.

- (1) Main focus on analysis is to find the cost related to communication delay in each link.
- (2) Task allocation requires extensive interaction with designers even though they claimed it is done automatically.
- (3) They assumed that scheduling policy is chosen automatically. Since the cost of communication delay depends on the communication protocol and scheduling policy on each node, it is questionable whether designers are able to give the cost of communication delay without knowing how DEA-suite select scheduling policy and what kind of protocol is to be used in application.

2.1.3. EUCLID[7]

Recently, most research on parallel computing system are oriented at development of computational models. Unlike recent trade, this system is to study operational behavior and performance of parallel computer systems. Although this kind of system can be used at the testing phase after design, it would be hard to be used in the initial phase of design or evaluation because of the following.

- (1) It requires detailed description on application and underlying systems. Therefore, users have to put much efforts to use the system, especially in the case of many candidate underlying systems. Also, description on application might be dependent on underlying systems.
- (2) Description depends on assembly language level. Therefore, it is very difficult to describe the system without knowing low level details of the system.

2.1.4. Massive Multiprocessor Simulator (MMS)[8]

It is to support the design of the system whose application is specific. First, the design begins by considering application itself, and decomposes its requirement based on the parallel flow graph (PFG) model which is similar to the AND-OR graph. Next, the system uses PFG as an input to MMS for checking response time, communication delay and so on. However, the system has inherent disadvantages as follows.

- (1) PFG is a subset of queueing network models. Therefore, what MMS can do is done by systems based on queueing network models.

- (2) The tool is to support the development of custom-made hardware. Therefore, it can hardly be used for evaluation of existing systems.

2.1.5. Simulator at Global Level (SiGLe)[9]

SiGLe is a development tool which can simulate the execution of distributed algorithm on a user-defined architecture. Three inputs to the SiGLe consists of the definition of the machine architecture, the specification of the distributed algorithm, and implementation scheme of the algorithm on the architecture. The LAR (Language for ARchitecture) specifies the MIMD architecture. The basic types of this language are processors, memories, and buses. A complex type can be defined as a record structure of component types and connections among them. The LAL (Language for ALgorithms) allows the description of the creation of processes and the communication among them. The sequential part of each process can be given in any language, at the convenience of the user (e.g., C or Pascal). The LI (Language for Implementation) specifies the mapping between processes and processors. And the simulation produces results such as the actual results of the computation, the total execution time, the execution time of each process, the utilization of each processor, the load on each bus, and the access conflicts to the shared resources and the corresponding waiting time for processes.

2.1.6. Parallel Architecture Research and Evaluation Tool (PARET)[10]

The PARET is a tool which simulates the execution of a distributed algorithm on a nonshared memory multiprocessor. Based upon the user provided specification of the program, operating system, and architecture, the PARET performs the high-level behavioral simulation, displays the result on the graphic workstation, and saves the summary statistics in a file. The input to the PARET system consists of the description of three subsystems: the user software, the system functions, and the interconnection. These subsystems share the same model - a directed flow graph where nodes represent units of computation and are connected by arcs for both data and control flow. And objects like elements (processors), tokens (units of data and control flow), buffers (storage of tokens), and threads (objects used to connect the buffers of different subsystems) are additionally required to define the subsystems and the interaction among them. In addition to the run-time visual display of the execution, the PARET collects statistics such as average

execution time of each node, the average occupancy of each buffer, the average transit time of a token, and the amount of time an element spends idle.

2.1.7. Architecture Description Language (ADL)[11]

ADL - in conjunction with its support software system, the ADS (Architecture Development System) - provides capabilities to describe and experiment with the hardware, system software, and application software of distributed systems. The structure of hardware is described as a collection of objects like processor, memory, switch, etc., and an interconnection among them. The structure of software is also specified as a collection of objects like task, file, message, etc., and an interconnection among them. Some system behaviors like scheduling, load balancing, communication protocol, fault management, etc. can be specified and the tasks of the application software can be provided as either real software or a simulated model of the real software.

2.2. Metrics

We define two types of metrics for parallel system evaluation: application independent metrics and application dependent metrics.

2.2.1. Application independent metrics

This category of metrics can be obtained by analyzing the structure of the system and from information provided by manufacturers without considering applications. In the following, we describe metrics belonging to this category.

- (1) Granularity of parallelism: fine grain, coarse grain, or medium grain.
- (2) Machine type: SIMD or MIMD.
- (3) Reliability: mean time between failure and inherent redundancy (e.g., the number of CPUs, the number of memory modules, the number of available data paths between components, etc.)
- (4) Cycle time and word size.
- (5) Scalability: cost of adding more components.
- (6) Communication scheme: dynamic or static.

(7) Intended application area (e.g., Cray 2 for scientific problems)

These metrics provide basic ideas on the power of the system. Based on these metrics, we examined four systems: Connection machine (CM-1), Cray 2, Parallel Inference Machine being developed in Japan (PIM-D), and Butterfly multiprocessor system. The result is shown in Table 1.

Metric	CM-1	Cray 2	PIM-D	Butterfly
Granularity of parallelism	fine grain	coarse-medium grain	coarse-medium grain	coarse grain
Machine type	SIMD	MIMD (basic level)	SIMD/MIMD	MIMD
Cooling System	Air-cooled	Liquid Immersion	Unknown	None
Operations/ns	10.8	15.8	1 logical inference (1G LIPS is assumed)	
Bottleneck	Memory/communication	Processor/memory	Communication	Memory
Scalability	High	Low	Very high	Extremely high
Communication scheme	dynamic	roughly fixed	fixed	fixed
Application area	modeling the problems in real worlds.	Scientific problems	Knowledge information processing	Unknown

Table 1. Comparison by application independent metrics

2.2.2. Application dependent metrics

This category of metrics can be obtained by simulation. In the following, we describe metrics belonging to this category.

- (1) Performance metrics: response time, communication delay, size of message on data path, processor utilization, average queue length, or delay caused by accessing shared resources.
- (2) Reliability metrics: performance degradation when a component fails.

(3) Scalability metrics: performance improvement by adding more components.

Based on these metrics, we compare the tools being surveyed and our proposed approach as shown in Table 2.

tool	performance	reliability	scalability	specification
DEA-suite	- communication cost	none	-	abstract data type
EUCLID	- response time - processor & memory utilization	none	variable number of processor & memory modules	instruction set for SW network model for HW
MMS	- response time - communication delay	none	none	parallel flow graph
SiGLe	- response time - processor utilization - shared resource access pattern	none	-	abstract data type + real code for SW abstract data type for HW
PARET	- response time - processor utilization - queue length - communication delay	none	-	directed graph
ADL	not available	specification possible	-	entity-relation-attribute model + real code
Our approach	described in the text	sensitivity analysis	sensitivity analysis	entity-relation-attribute model (BRSL)

Table 2. Comparison by application dependent metrics

3. Berkeley RSL

BRS language is based on entity-attribute-relation model. The components of a parallel machine hardware (e.g. processor, node, communication link) and software (e.g. execution block, interface, process) are represented as *entities* or in E-R-A model as shown in Figure 2.

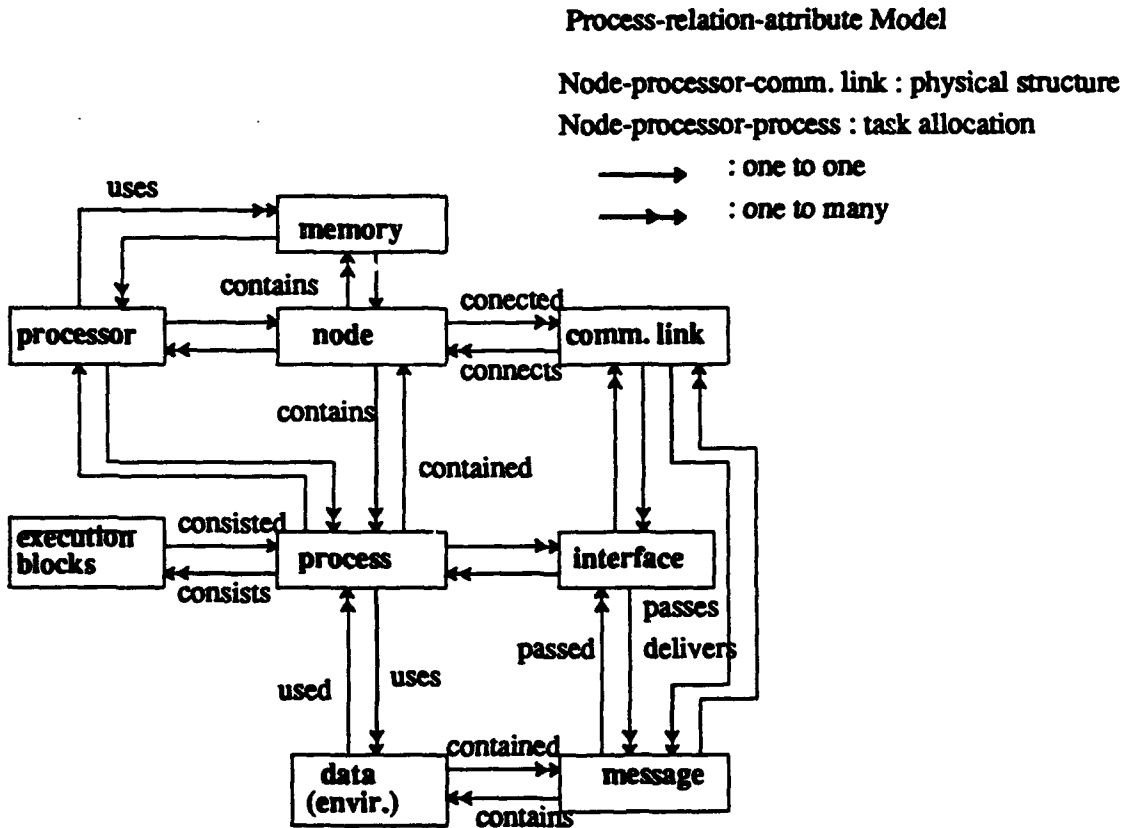


Figure 2: E-R-A Model

The entities have *attributes*. For example the processor can be attributes like Computation power (in MIPS) to describes properties of the component. Some of the attributes are basic and provided by the design. Other attributes are derived and should computed during the evaluation procedure and simulation. The properties related to more than one component-type are represented by *relationships*. For example nodes are connected via communication links.

The language is described by Backus Naur Form grammar rules as shown in the illustrative subset of rules listed in the following.

3.1. BNF grammar for BRSL

SYSTEM: <entity_definition>.
| <entity_definition> <entity_definitions>.
<entity_definition> : <node_entity_definition>
| <process_entity_definition>
| <block_entity_definition>
| <interface_entity_definition>
| <link_entity_definition>
| <message_entity_definition>
| <data_entity_definition>
<entity_definitions>: <entity_definition>
| <entity_definition> <entity_definitions>

% task allocation specification

<node_entity_definition> :
NODE <node_name> CONTAINS PROCESS <process_names>.
| NODE <node_name> IS CONNECTED TO LINK <link_names>.

% process related timing information specification

<process_entity_definition>: <process_relations> <process_attributes>.
<process_relations>: <process_relation>.
| <process_relation> <process_relations>.
<process_relation>:
PROCESS <process_name> CONSISTS OF BLOCK <block_names>.
| PROCESS <process_name> CONSISTS OF PROCESS <process_names>.
| PROCESS <process_name> USES DATA <data_names>.
| PROCESS <process_name> INPUTS FROM INTERFACE <interface_name>.
| PROCESS <process_name> OUTPUTS TO INTERFACE <interface_name>.
| PROCESS <process_name> IS CONTAINED IN NODE <node_name>.
| PROCESS <process_name> STRUCTURE <block_entity_definition> END.
<process_attributes>: <process_attribute>.
| <process_attribute> <process_attributes>.
<process_attribute>: DEADLINE <number>.
| MAXIMUM_EXECUTION_TIME <number>.
| MINIMUM_ARRIVAL_INTERVAL <number>..

% message related timing information specifications

<message_entity_definition>: <message_relations> <message_attributes>.
<message_relations>: <message_relation>.
| <message_relation> <message_relations>.
<message_relation>: MESSAGE <message_name> CONTAINS DATA <data_name>.
| MESSAGE <message_name> IS PASSED THROUGH INTERFACE <interface_name>.
| MESSAGE <message_name> IS DELIVERED BY LINK <link_name>.
<message_attributes>: <message_attribute>.

```
| <message_attribute> <message_attributes>.  
<message_attribute>: AMOUNT_DATA <number>.  
| MINIMAL_ARRIVAL_INTERVAL <number>.  
| VALIDITY <number>.
```

4. Distributed Simulation

Simulation of a parallel architecture along with a benchmark program would be a very time consuming job because we have too many components to simulate such as hundreds or thousands of processors, many memory modules, and many switching components and also both the behavior of each component and structural relationships among them are very complex. To cope with this performance problem of the simulator itself, many researchers have worked on the distributed simulation in which the simulator itself is running on a distributed environment.

There are two major approaches to distributed simulation: *synchronous* method and *asynchronous* method. In the synchronous method, the computation consists of a sequence of phases, where in each phase all the processes simulate the behavior of a component of the target system for time Δt and then synchronize. The primary issue in this approach is the selection of the parameter Δt which affects the numerical accuracy of the results. Another issue, which is application-dependent, is finding parallelism for the simulation.

In the asynchronous method, the target system is modeled as a set of *asynchronous* computations running concurrently on multiple processors. The simulation time advances asynchronously for each process. One problem which arises in structuring the simulation as a set of asynchronous communicating computations is that each process has to determine without access to global state when it is safe for it to advance its local simulation time without causing deadlock. This problem is called *time-advancement* problem. And work in the area of distributed simulation has primarily concentrated on distributed time-advancement. The following are some of the properties that are desirable in a distributed control (special case of which is time advancement) algorithms:

- (1) *Global knowledge*: use global knowledge to make decisions if it allows more optimality in reaching towards a good solution.
- (2) *Scalability to large problems*: If there is any locality in the problem, the algorithm should exploit it.

- (3) *Collection of knowledge*: The scheme for collection of knowledge should be efficient with low latency and overhead. In particular, it is desirable that it does not require flushing of channels or make unrealistic demands on resources, such as unlimited buffering capacity.
- (4) *Adaptive*: The overhead in resolution of deadlocks, etc. should be in proportion to their frequency.
- (5) *Avoid rollbacks*: Rollbacks can be expensive during run-time. Furthermore, the schemes using rollbacks can be difficult to implement. Therefore, one must avoid rollbacks, and if advantages are there in using a scheme with rollback, reduce their frequency to optimize performance.
- (6) *Dynamic topologies*: The solutions must be able to deal with situations where processes or communication channels can get created or terminated dynamically.
- (7) *Finite memory*: The algorithms should work if the processors have only limited memory.

The Null-message scheme[12] uses only very localized information and thus advances time very slowly, and can even fail to work in some cases. The algorithm proposed by Bryant[13] uses global information to detect and resolve deadlocks but requires flushing of channels during status collection and thus may fail if sufficient memory is not available for flushed messages. The time-warp method[14] does not use any global information and relies on rollback and is thus expensive to implement in general. The circulating token algorithm[15] takes advantages of locality in deadlock resolution; however there is no mechanism for dealing with dynamic topologies and changing the token assignment dynamically based on the system state. Another algorithm by Chandy and Misra[16] uses full global state for time advancement; however, it requires that all processes be deadlocked before resolution is carried out and thus cannot take advantage of locality. Finally the algorithm in [17] is based on the hierarchical algorithm and attempts to meet all the above criteria.

The main idea in the hierarchical algorithm is to divide the process network into *clusters*, according to the locality of frequency of deadlocks. Clusters appear as single processes at higher levels of the hierarchy. Time advancement is carried out locally within each cluster if possible. If time advancement is not possible locally, then an attempt is made to advance time at higher levels of the hierarchy. It is guaranteed that at

least one message will be sent or received between the initiation of the algorithm and its termination. This algorithm has the following features:

- (1) The algorithm is hierarchical. Therefore time increment in local deadlocks is carried out rapidly before the deadlock spreads to other processes. Also, because of the hierarchical nature of the algorithm, communication and processing overhead in time advancement is localized.
- (2) The algorithm places emphasis on time advancement rather than deadlock detection.
- (3) Many time advancements may be going on at the same time in different clusters of the hierarchy.
- (4) The algorithm can be used in parallel with other time advancement schemes.
- (5) The algorithm is conservative, i.e., no rollbacks occur during its execution. However, it uses any look-ahead information that the programmer provides in the simulation program to speed up time advancement.

5. Network Event Manager

It is very important to be able to track events that occur in a distributed computing environment, to take actions for analyzing the events, and to improve the behavior of the system. During a distributed simulation, we are often interested in determining the global state of the system with respect to simulation time. For example, it may be desirable to display snapshots of system state graphically at a rate proportional to simulation time. Some of the examples are as follow:

- (1) When the simulation clock of all the processes exceeds a certain maximum, report to the user.
- (2) Report the mean and variance of queue length of all the channels to the user after every 5 units of simulation time.
- (3) When the number of events executed by all the processes exceeds a certain maximum, initiate a simulation termination algorithm.

In the domain of distributed computing, specifically distributed simulation, the following services are necessary.

- (1) *Collect/stored data*: Monitor the system continuously and collect data required for detection, analysis, and action as explained next.
- (2) *Detect event/state*: Detect undesirable or interesting events/states. We need a language to specify those events/states and efficient algorithm to detect them.
- (3) *Analysis*: Find the most possible cause for the detected event/state by analyzing the stored data. For this we need inferencing capability with the management of uncertainty. If more data are required for the analysis in addition to normally collected data, we should conduct experiments on remote machines to acquire those required data.
- (4) *Plan/execute action*: Plan and execute actions to control the functioning of distributed objects to take the system to a more desirable global state.

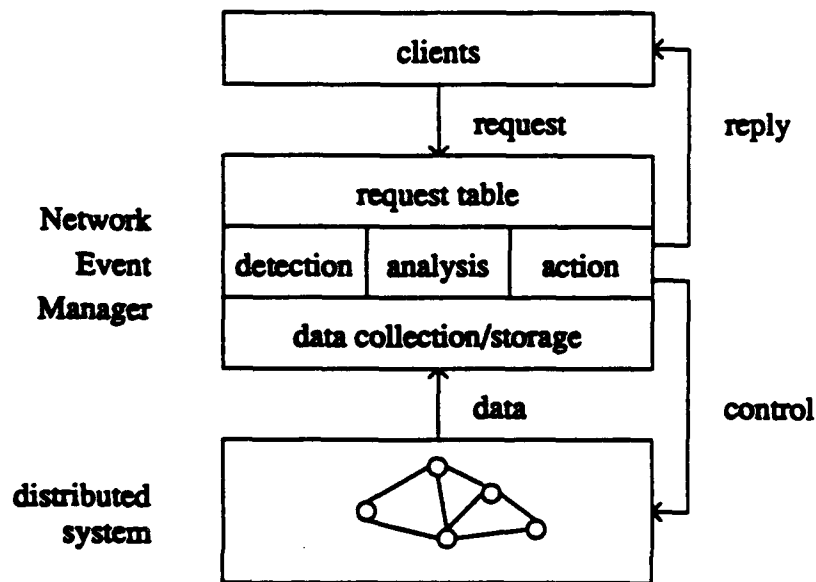


Figure 3 Block diagram of the Network Event Manager

For the network environment, in [18] Shim described a system called the *Network Event Manager* (NEM) which provides the above services. The basic structure of the NEM is shown in Figure 3. It accepts request of the form

$\text{event/state} \rightarrow (\text{hypothesis} \rightarrow \text{action}).$

Upon receiving this request the NEM detects the event/state, analyzes data to prove that

the given hypothesis is the cause for the event/state, and plans/executes the action if the hypothesis is proven to be the cause. If there can be several different causes for an event/state, then the request is given as follow:

$$\begin{aligned} \text{event/state} &\rightarrow (\text{hypothesis-1} \rightarrow \text{action-1}) \\ &\text{or } (\text{hypothesis-2} \rightarrow \text{action-2}) \\ &\dots \\ &\text{or } (\text{hypothesis-n} \rightarrow \text{action-n}) \end{aligned}$$

After detecting the given event/state, the NEM analyzes data to decide which is the most possible cause among the n hypotheses, and then plan/execute the corresponding action. Or the basic form of the request can be simplified as following two forms:

$$\begin{aligned} \text{event/state} &\rightarrow (\rightarrow \text{action}) \\ &\rightarrow (\rightarrow \text{action}) \end{aligned}$$

In the first case the action is planned/executed as soon as the event/state is detected and in the second case the action is planned/executed without any detection or analysis.

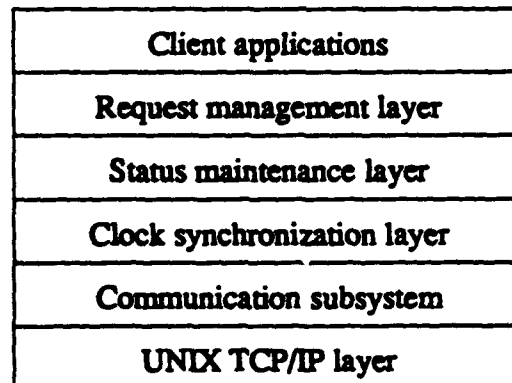


Figure 4 Layers of the Network Event Manager

The NEM is designed in a layered manner as shown in Figure 5. The communication layer provides facilities for interprocess communication across a network. The clock synchronization layer provides a method of determining ordering and duration of events/states in the system.

The status maintenance layer collects data from the distributed system and stores them in the database. We view the distributed system as a collection of various kinds of objects such as host, process, message, file, etc as in Figure 4. An object is characterized by two lists: attributes and operations. Attributes represent states of an object and can be either constant or time-varying. An object operation is a function which can be operated on that object. For example, a process object can have constant attributes such as *process-identifier*, and *command*, a time-varying attribute such as *process-status*, and operations such as *create*, *destroy*, *start*, *stop*. Relationships exist among objects. *Part-of* relationship exists from a process to a host, *connect* relationship exists among hosts, etc. With this conceptual view of the distributed system, the collected data consists of *object attribute values*, *occurrence of object operations*, and *relationships among objects*. Some of these data are collected by either *sampling probes* or *tracing probes*. A sampling probe is periodically or aperiodically invoked and collects values for time-varying attributes. A tracing probe is invoked when an object operation occurs and collects data regarding that object operation.

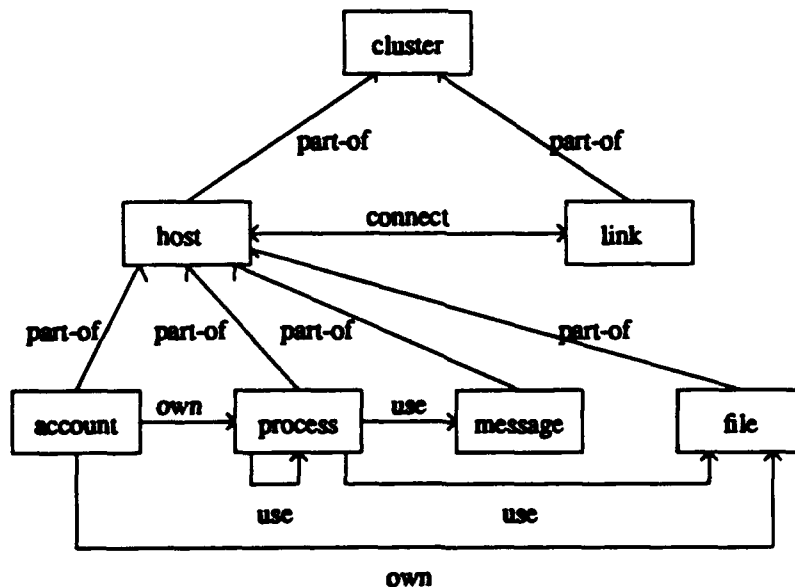


Figure 5 Conceptual view of a distributed system

Request management layer consists of 3 modules to detect events/states, analyze data, and plan/execute actions. The module to detect events/states, in real-time, detects events/states specified in a classical temporal logic-based language. The modules to analyze data and plan/execute actions consists of a prolog process which can communicate with remote processes and use data in the database. These two modules have the capabilities to reason with uncertainty management and experiment on remote machines and execute planned actions.

6. Conclusion

We conclude this report by itemizing our future works in the following.

- (1) Classification of parallel machines : More research will be done to specify and classify various parallel machines.
- (2) Evaluation of specification language : Feasibility, descriptive power, and capability to represent various parallel architectures will be studied.
- (3) Identification of metrics : We need to define important metrics such as performance measure, scalability, reliability, fault-tolerance, concurrency, granularity, etc.
- (4) Benchmark program development and testing : Prototype evaluation will be analyzed by benchmark programs for various softwares and hardware.
- (5) Sensitivity analysis : The performance variation due to the changes in resources should be studied.

Our BRSL will have complete system integration. High level hardware-software simulator, analysis tool and tester will be integrated into the BRSL system. We will extend and implement our prototype design of static analyzer and high level benchmark system (simulator) to be integrated altogether. Final version of the integrated BRSL system tool will be tested and demonstrated.

References

1. Moe Shahdad, "An Overview of VHDL Language and Technology," *23rd Design Automation Conference*, pp. 320-326, 1986.
2. M.R. Barbacci, *The ISPS Computer Description Language*, Dept. of ECE, Carnegie-Mellon University, August 1977.

3. J. Bara and R. Born, "A CDL Compiler for Designing and Simulating Digital Systems at the Register Transfer Level," *Proc. Int. Conf. CHD's Applications*, pp. 96-102, 1975.
4. F.J. Hill and G.R. Peterson, *Digital Systems: Hardware Organization and Design*, New York, Wiley, 1978.
5. S.G. Shiva, "On the Selection and Development of an HDL," *Proceedings of the Seventeenth Annual Hawaii International Conference on System Sciences*, pp. 120-128, 1984.
6. Paul R. Ponville, Lois M. L. Delcambre, and Steve P. Landry, "Describing Distributed Environments," *IEEE International Conference on Distributed Computing Systems*, pp. 296-302, 1986.
7. James M. Butler and A. Yavuz Oruc, "EUCLID: An Architectural Multiprocessor Simulator," *IEEE International Conference on Distributed Computing Systems*, pp. 280-287, 1986.
8. Tadashi Ae, Reiji Aibara, Minoru Etoh, and Hiroshi Matsumoto, "A Massive Multiprocessor Simulator for Performance Evaluation," *International Conference on Supercomputing*, pp. 73-82, 1985.
9. F. Andre and A. Joubert, "SiGLe: An Evaluation Tool for Distributed System," *IEEE International Conference of Software Engineering*, pp. 466-472, 1987.
10. Kathleen M. Nichols and John T. Edmark, "Modeling Multicomputer Systems with PARET," *IEEE Computer*, pp. 39-48, May 1988.
11. John T. Ellis, James W. Hooper, and Tony A. Johnson, and W. Daniel Hillis, "An Architecture Description Language (ADL) for Describing and Prototyping Distributed Systems," *Proc. of the Seventeenth Annual Hawaii International Conference on System sciences*, pp. 105-119, 1984.
12. K.M. Chandy and J. Misra, "A Nontrivial Example of Concurrent Processing: Distributed Simulation," *COMPSAC*, 1978.
13. R.E. Bryant, "Simulation on a Distributed System," *COMPSAC*, 1979.
14. D.R. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Tiern Warp Mechanism," *Proceedings of the Conference on Distributed Simulation*, vol. 15, no. 2, pp. 63-69, Jan. 1985.

15. J. Misra, "Distributed Discrete-Event Simulation," *Computing Surveys*, vol. 18, no. 1, pp. 39-65, March 1986.
16. K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *CACM*, vol. 24, no. 11, pp. 198-206, April 1981.
17. A. Prakash, "Modeling, Simulation, and Analysis of Systems with a Large Number of Interacting Entities," *Ph.D. Thesis*, Computer Science Division, Dept. of EECS, University of California, Berkeley, CA 94720, Dec. 1988.
18. Y.-C. Shim, "Knowledge-Based Integrated Mechanism for the Management of Distributed Systems," *Qualifying Proposal*, Computer Science Division, University of California, Berkeley, CA 94720, 1990.