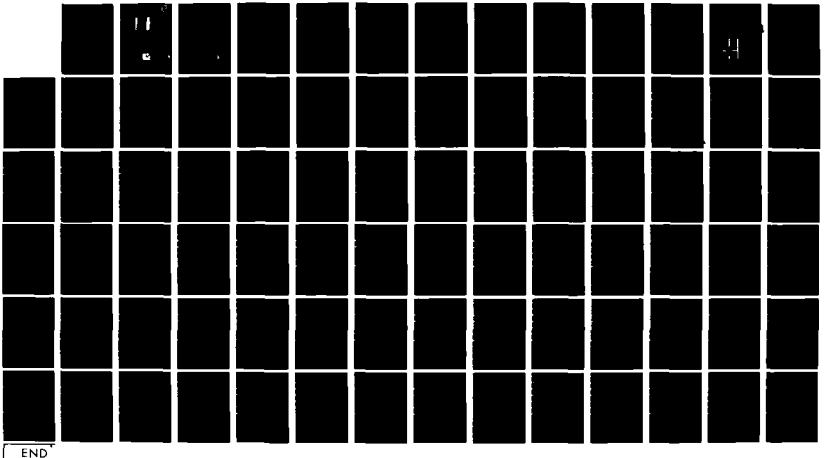
AD-A126 233 SIMULATION NETWORK FOR TEST AND EVALUATION OF DEFENSE SYSTEMS PHASE I SUR..(U) CLEMSON UNIV SC DEPT OF COMPUTER SCIENCE E W PAGE 15 MAY 83 MDA903-82-C-0211 1/1

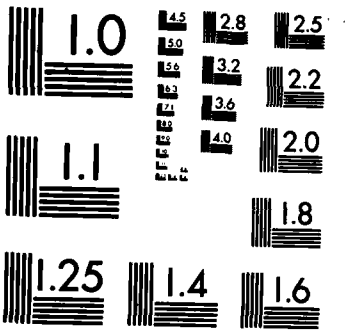
UNCLASSIFIED F/G 9/2 NL

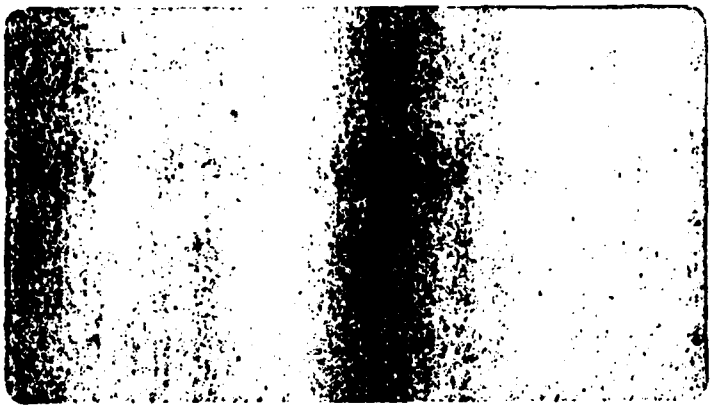


END

FILMED

DTIC





Contract No. MDA903-82-C-0211

Department of Computer Science

CLEMSON UNIVERSITY

SIMULATION NETWORK FOR TEST AND EVALUATION OF DEFENSE SYSTEMS

PHASE I

SURVEY OF DOD TESTBED REQUIREMENTS

Contractor No:	3-03-1769
Name of Contractor:	Clemson University
Contract Expiration Date:	15 May 1983
Contractor's Project Director:	E. W. Page
Short Title of Contract Work:	Geographically Distributed Simulations
Phone Number:	(803)656-2398

The views, opinions, and findings contained in this report are those of the author(s) and should not be construed as an official Department of Defense position, policy, or decision, unless so designated by other official documentation.

EXECUTIVE SUMMARY

1. Introduction

The objective of this study is to assess the applicability of modern computer and communication technology to the problems of testbed design and operation. The work is being conducted in two phases. Phase I, reported here, was undertaken to gather and assimilate information on several DOD testbeds. Phase II consists of a critical analysis of problem areas in networking simulations, the identification of simulation network requirements, and the development of a preliminary network design description.

2. Needs of the Testbed Community

As a result of the information gathering phase of the project, the following needs of the testbed community were identified:

1. improved capabilities for information gathering and sharing on testbed models, data, and resources,
2. adoption of hardware and software standards, where practical, to encourage the sharing and reuse of existing hardware and software resources, and
3. further investigation of the communication and control problems associated with geographically distributed testbeds

3. Recommendations

Several modern data processing and communications capabilities can be employed to address the above needs. The initial recommendations resulting from this study are that the potential of the initiatives listed below be explored.

1. The development and use of a community-wide electronic mail service.
2. The implementation of a database management system to catalog information on available models, data, hardware specifications, and testbed architectures.
3. The development and adoption of standards for the design and implementation of mathematical models and of both hardware and software testbed components.
4. The utilization of a computer-based communications network to improve the reliability and availability of communications for geographically distributed testbeds.

3.1 Electronic Mail

An electronic message transfer system will provide more timely communications and improved information management for members of the testbed community. The functions of an electronic message transfer system should be:

1. an inter/intra office mail system capable of creating,

transmitting, editing, filing, searching, and printing mail and documents,
2. a community/project bulletin board system,
3. a community/project broadcast message system, and
4. a calendar system for scheduling activities.

3.2 Database Management System

A database management system utilized by the test and evaluation community would enhance the sharing and standardization of information within the community. In a database management system, information can be stored in a computer system in such a manner that the information can be searched, correlated, and retrieved from a database. Additionally, if a database management system were part of a network of information sharing services such as an electronic mail system, the database could be accessed remotely by testbed personnel over a computer communications network. This would allow testbed personnel across the country to obtain information on a timely basis.

3.3 Adoption of Standards

The adoption of standards would promote the reuse of resources that have been developed by the testbed community. Standards also offer greater flexibility in testbed design in that testbeds can evolve and incorporate new components as they become available. Existing resources that do not exactly meet the fidelity or performance requirements of new testbeds could be used during the development and testing stage. This would allow the development process to proceed without waiting for key resources. The reuse of existing resources would reduce the overall cost of testbed development as well as the time needed before a testbed produces meaningful results. The applicability of standards in the following areas should be investigated:

1. mathematical models,
2. software, including programming methodology, programming languages, and simulation interfaces, and
3. communications.

3.4 Computer-Based Communications Network

A computer network can provide a basic framework for developing testbeds. The network should not be thought of as a dedicated set of communications links but as a delivery service that guarantees the timely delivery of information between sites served by the network. The physical communications system may incorporate dial-up lines, leased telephone company lines, microwave links, or satellite links, but the user is presented with the illusion of a dedicated circuit. A computer-based network provides the automatic services of error detection and correction, message routing, and load balancing. The services provided by a network relieve the testbed architects of the problems attendant with the use of traditional communications channels and thus allows them to concentrate on the testbed mission.

Table of Contents

List of Figures

FOREWORD

This report was prepared for the Director, Defense Test and Evaluation by the Department of Computer Science at Clemson University under contract MDA903-82-C-0211. H. Eugene Thompson of OSD was the technical monitor for this contract and made many significant contributions to the study.

Dr. E. W. Page served as the principal investigator for this study with assistance from Drs. A. Wayne Madison and Harold C. Grossman. Dr. Dan Warner of the Mathematical Sciences Department and Dr. John Spragins of the Electrical and Computer Engineering Department also contributed.

The objective of this study is to assess the applicability of modern computer and communications technology to the problems of testbed design and operation. Phase I of the study was undertaken to gather and assimilate information on several testbeds; it is not a critical review of DOD testbeds.

During the course of this phase of the project, the following testbed sites were visited:

TESTBED	LOCATION
IFFN	Kirtland AFB, NM
C^3CM	Kirtland AFB, NM
CSEDS	Morestown, NJ
JINTACCS	Ft. Monmouth, NJ
REDCOM JTC	MacDill AFB, FL
REDCOM ASG	MacDill AFB, FL

Additionally, visits were made to the Naval Ocean Systems Center (NOSC) and to Logicon, Inc. of San Diego, California to gain insight on

testbed design alternatives by talking with persons involved with both existing and new designs. Members of the project team also attended the Manned Simulator Testbed Workshop held at the MITRE Corporation on 22-24 June 1982.

This report provides an overview of modern testbeds, identifies many of the most pervasive problems faced by testbed designers and suggests approaches for dealing with the problems of testbed implementation.

1.0 Testbed Overview

The Department of Defense (DOD) has a well-established precedent of using testbeds to train personnel, test procedural and doctrinal concepts, and to test equipment. A testbed typically consists of computer simulations, operational hardware, communications subsystems, and human operators. A testbed will employ an appropriate mix of simulations, hardware, and humans to simulate a complex system with sufficient fidelity to satisfy the test objectives. Testbeds offer substantially more realism than computer simulations alone while maintaining a high degree of control over test variables. A testbed can come arbitrarily close to simulating an actual system but with more repeatability than is possible in testing a physical system. Because of the complexity of modern testbeds, several years are required for a testbed to progress from the conceptual stage to an operational stage.

1.1 Testbed Architecture

Although the testbeds surveyed have unique objectives and therefore unique design requirements, there is a common philosophy of organization as illustrated in Fig. 1. Fundamentally, a testbed consists of two parts:

1. an interconnection of physical components and computer simulations comprising the system being simulated, and

2. an auxiliary system that provides a test environment and monitors the system's behavior.

Most testbed architectures require a central simulation facility. The

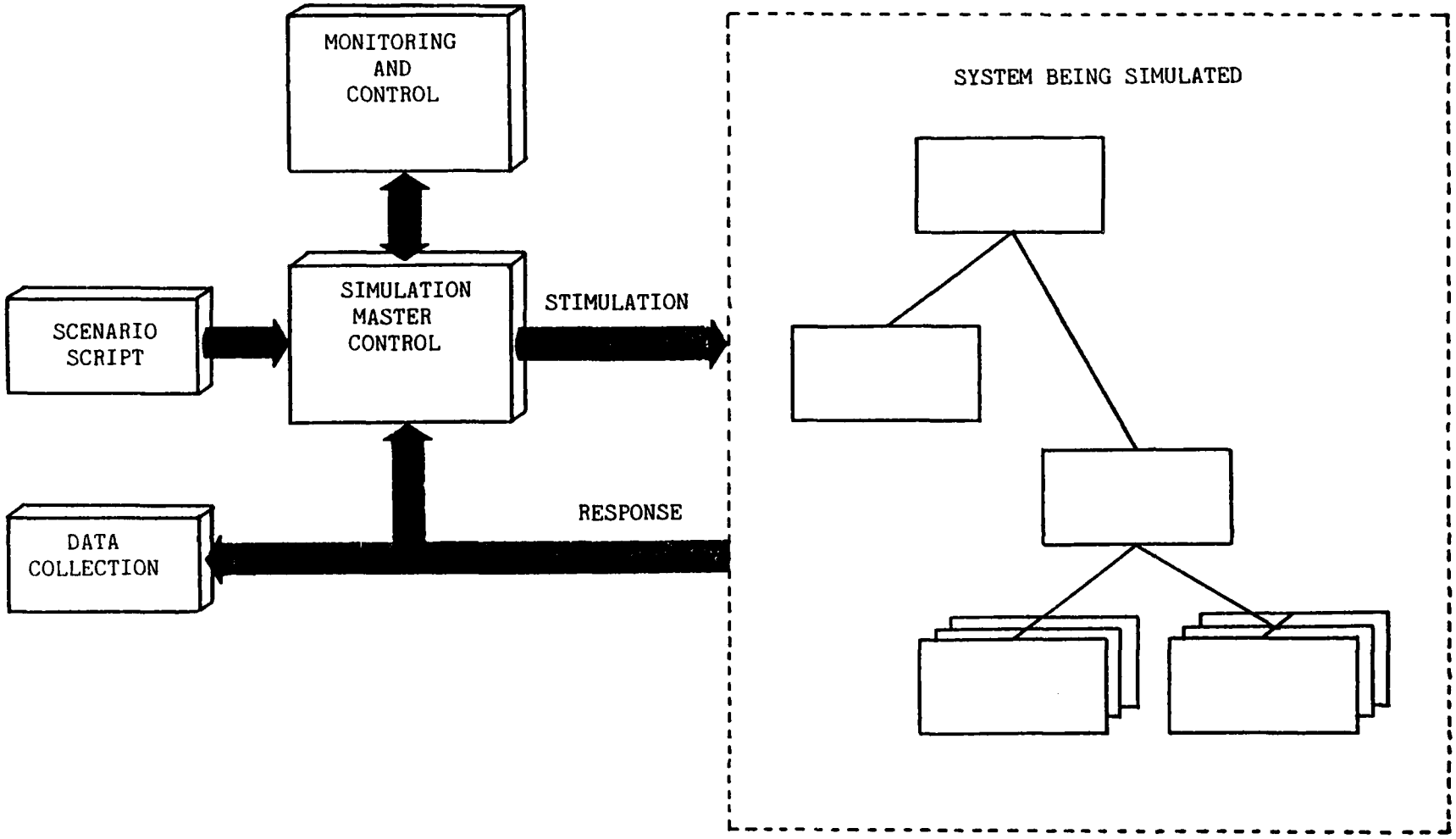


Figure 1: Testbed Organization

central simulation facility is typically supported by a powerful mainframe computer system that interprets a scenario script, generates stimulation data, and monitors responses from the simulated system. The central simulation facility may also support software simulations as if they were an integral part of the system being simulated.

1.2 Testbed Components

A system may be thought of as a collection of components and subsystems that operate within an environment. The testbed designer decomposes the system to be tested into its constituent parts and then determines which parts are best represented by actual hardware or human participants and which parts may be simulated. Figure 2 illustrates the decomposition of a system.

Testbed components may be operational systems, computer simulations or a combination of operational hardware and computer simulations. In the IFFN testbed, for example, the test system consists of both live and simulated components such as the HAWK and PATRIOT batteries, simulated radar tracks, manned command and control posts, and actual tactical data links. The simulated system at the CSEDS testbed consists of a live SPY-1 radar, manned radar and communication posts, the actual computer hardware and software used for shipboard operations, together with simulated threats and simulated interceptor missiles.

The testbed objectives together with time and cost constraints will dictate requirements for the degree of real-time operation necessary and the appropriate mix of live and simulated components. For testbeds

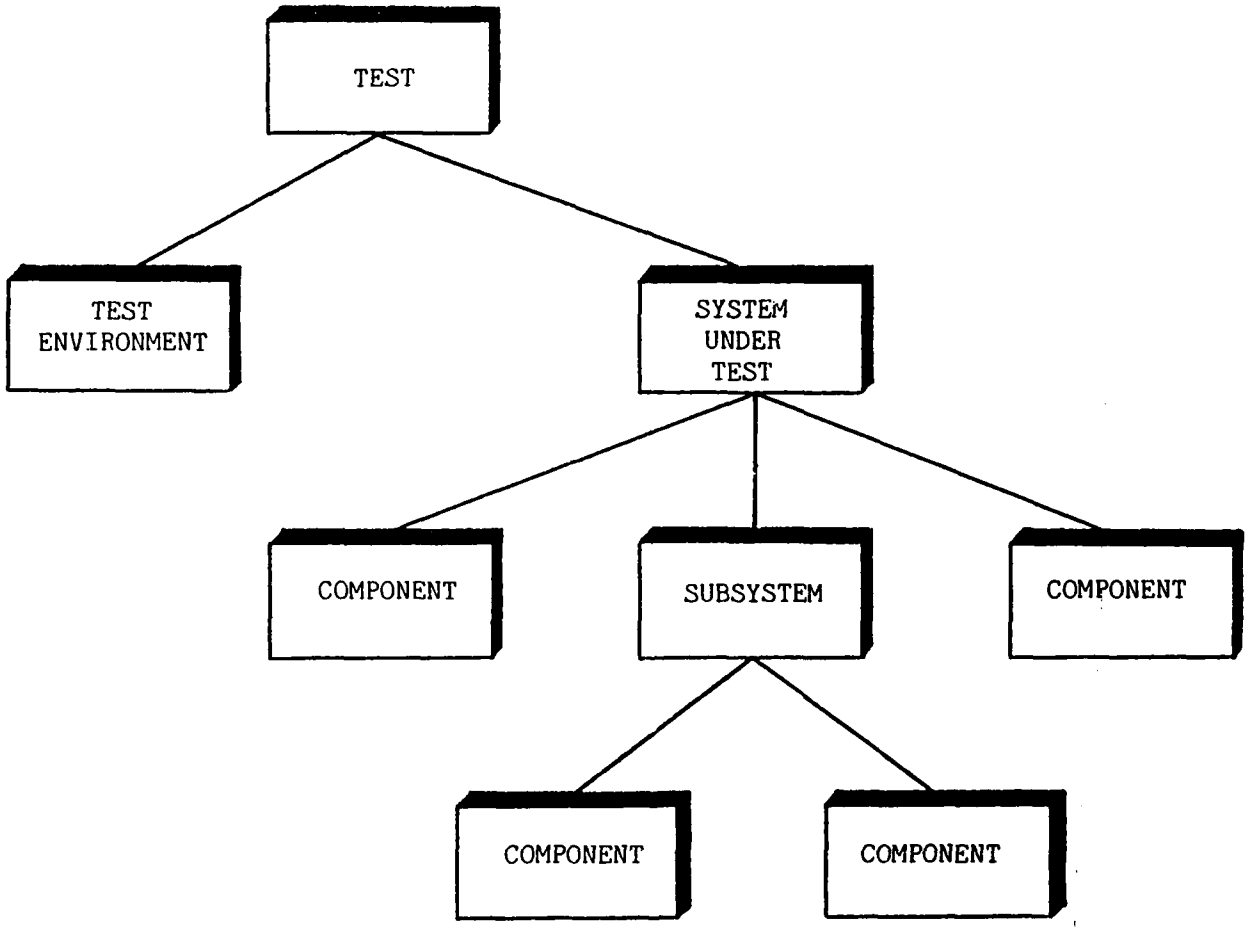


Figure 2: Decomposition of a System

such as IFFN that use human operators, events should be perceived as occurring in real time. A testbed consisting solely of computer simulations can use simulated time which may be faster or slower than real time.

1.3 Testbed Control and Monitoring

The efficacy of a testbed is strongly dependent upon the capabilities of the stimulation and data monitoring portions of the general model (Fig. 1). A test typically consists of three stages: pretest preparation, test execution, and post-test analysis. During the pretest preparation, a scenario script is developed that provides the test environment for the test. For some testbeds, a high-level language allows scenarios to be prepared off-line in an English-like language developed especially for scenario generation.

During the test execution stage, the master simulation control program interprets the scenario script and, in turn, generates the appropriate stimuli for the simulated system. The master simulation control also continuously updates displays available to the test director and his staff. Events occurring within the simulated system can alter the environment. An interceptor fired at a threat, for example, should remove that track from all simulated radar scopes if the master simulation control considers the intercept attempt to be successful. For some testbeds, the scenario script may be altered during testing by the test director.

During the test, data is collected for later analysis. This process

may be done in real time (i.e., the data is transmitted back to the central site as it is collected), or it may be done off-line (i.e., the data is recorded at the test component and sent back to the central site after the test is completed).

In the post-test analysis stage, the collected data is reduced and analyzed to determine the results of the test. The results can be used to measure the effectiveness of the system being tested for the given environment and to provide insight into changes that might be made to the system to improve its effectiveness. System deficiencies can also be high-lighted by using the collected data to replay the test. The results of taking an alternative course of action at a particular point during the test can be analyzed by replaying the test up to the point of interest and then continuing the test based upon the different actions.

1.4 Testbed Communications

The central simulation facility may communicate with the testbed components through both analog and digital leased lines, dedicated cables, or tactical HF links. A testbed such as JINTACCS, which is designed for interoperability testing, needs to employ actual tactical communications equipment. In the IFFN testbed, however, the test results are influenced only by the contents of the messages being transferred between sites, not by how the testbed designers chose to actually transmit them.

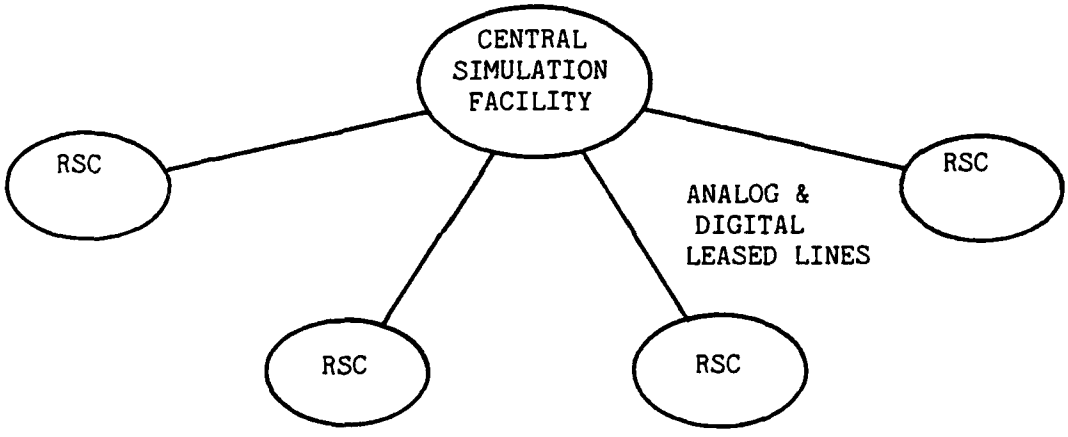
If the testbed component is an operational system such as an

airborne E-3A, actual tactical data links are used for communications with the central simulation facility. When the component is a computer simulation or a combination of operational hardware and computer simulations, the communication link between the central simulation facility and the remote component will likely be a leased line over which both stimulation and response messages will be exchanged in addition to tactical data.

The PATRIOT battery located at Ft. Bliss, TX is an example of a remote component that incorporates both operational and simulated components. The PATRIOT battery will interact with the IFFN central simulation facility at Kirtland AFB, NM using digital land lines. Messages generated by the central simulation facility are interpreted by a computer at the PATRIOT site which, in turn, presents a realistic air defense picture to human operators at the PATRIOT battery. Likewise, events at the PATRIOT site, such as launching an interceptor, generate messages to the central simulation facility which will alter the testbed environment.

1.5 Testbed Topology

From a topological point of view, a testbed consists of a central simulation facility and one or more simulation components. The components of the simulated system may be co-located as in the CSEDS testbed. Such testbeds represent a traditional approach to testbed design and present relatively few communications and control problems since the components can be attached to the simulation facility by cables. In other testbeds such as JINTACCS and IFFN, the components



REMOTE SIMULATION COMPONENTS (RSC)

Figure 3: Geographically Distributed Testbed

are geographically distributed as depicted in Fig. 3. There are three primary reasons motivating the distribution of testbeds:

1. Remote simulation components represent a combination of equipment and personnel and are not easily portable.
2. A given remote simulation component may be shared by more than one testbed.
3. A distributed testbed may more accurately represent the system being tested and may, therefore, increase the credibility of the test results.

The distribution of testbed components introduces the additional problems of delay and synchronization resulting from long-distance communications. This study will concentrate on the problems unique to geographically distributed testbeds.

2.0 Common Testbed Needs and Requirements

The survey of testbeds identified three areas of common concern among testbed designers:

1. the lack of well-established methods to collect and share information within the testbed community,
2. the problems of interfacing new testbeds with existing hardware and software components, and
3. the difficulties encountered in constructing and maintaining a communications systems between distributed components.

This section will review these problems. Section 3 of this report will offer initial suggestions for approaching these problems.

2.1 Information Gathering and Sharing

Each testbed surveyed had a significant need on the part of designers to gather and share information on models, performance data, interface requirements, and scheduling problems. This process is currently carried out through telephone conversations, literature searches and, more typically, site visits. The process is often lengthy and expensive and may fail to find pertinent information. At the Workshop on Manned Simulator Testbeds for the Defensive Air Battle held on 22-24 June 1982 at MITRE - Washington, numerous participants expressed the desire to have greater capability for sharing information within the testbed community. In some instances researchers at one installation were not aware of their counterparts at another installation, were not aware of the availability of critical data, and were not aware

of the activities that are currently underway at peer installations.

An initial step in designing a testbed is generally a models survey. This entails gathering information on models that have already been developed and that are potential candidates for inclusion in the new testbed. Models range from simulations of existing hardware to mathematical models of weather patterns. Each model is evaluated to determine if it meets the requirements of the new testbed. The results of the evaluation of the model may result in its inclusion in the testbed, the inclusion of a subset of the model, a modification of the model to meet the needs of the new testbed, or its rejection. The ability to utilize all or part of an existing model can result in a significant savings in time and cost associated with developing a testbed. The failure to locate or to properly identify a model that already exists can have a significant impact upon the cost and effort required to develop a testbed.

2.1.1 Models

A testbed is composed of a set of hardware, software, and human components that are being tested within a given environment. Depending upon the goals of the test, an individual component may be a physical component or it may be represented by a simulation. Simulated components must be based upon a mathematical or conceptual model that describes the dynamic behavior of the component with a level of fidelity that is adequate for the test. The models seldom need to mimic the full range of behaviors displayed by the physical component since usually only a small subset of the component's actions affect other components

in the overall system. The jettisoning of a fuel pod by an individual aircraft, for example, will probably have little effect upon the outcome of a full-scale air battle. A testbed for that individual aircraft, however, may require that the model accurately describe the behavior associated with the fuel pod component.

The major hindrance to sharing or reusing models is that each model displays a level of fidelity that is unique to the particular test for which the model was developed. Figure 4 illustrates how various models capture different levels of reality and that such models are often not subsets of each other. The various testbeds surveyed did find models of components that could be included in the new testbed. Upon closer scrutiny, however, a model was often rejected because it did not mimic the subset of behavior that was required by the new testbed.

The unique objectives of the different testbeds do not preclude the sharing or reuse of models. There are often many components of a testbed that are peripheral to the main test and can, therefore, be modeled rather loosely. For a testbed that is to include the ability to generate a large number of radar tracks, for example, several existing aircraft models may produce tracks with reasonable fidelity.

A methodology that was identified in the IFFN testbed is the use of table-driven models. In these instances, a generic model of a component is developed. The particular characteristics of a component for a given system are provided via a table. An aircraft, for example, could be modeled from a collection of generic models for subsystems such as the aircraft dynamics, radar systems, and weapons systems. To use the model to simulate different aircraft, one simply provides a

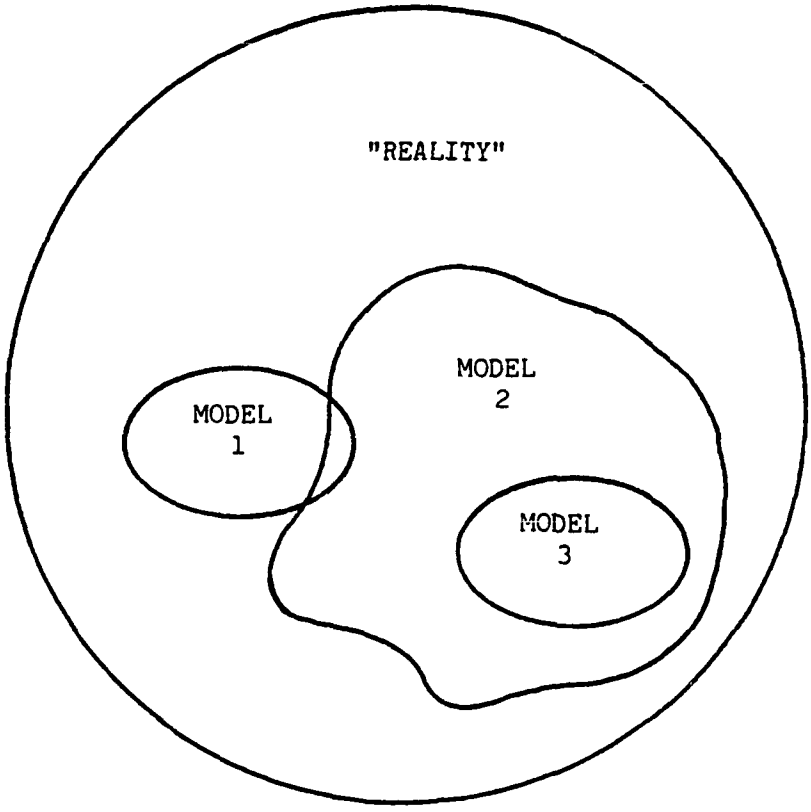


Figure 4: Illustration of Subsets of Behavior Mimicked by Models

table of characteristics for each of the subsystem components for each aircraft. Table-driven models appear to provide the potential for sharing and reuse.

Three categories of models that are of interest to testbed designers are:

1. mathematical or conceptual models,
2. computer simulations, and
3. testbed architectures.

2.1.1.1 Mathematical Models

A simulation of a testbed component must be based upon a mathematical description of the behavior of that component. Such models may range from complicated sets of differential equations to event-driven state transition procedures or a combination of both. The model usually attempts to describe only the behavior that relates to the characteristics of the overall system being tested.

The survey of existing mathematical models is an important step in the initial design of a testbed. Even in cases where an existing model does not provide the required fidelity or the computer simulations derived from the model are incompatible with the new testbed, the model provides a valuable kernel from which acceptable models can be developed. A thorough models search is crucial to the successful and timely development of a testbed.

2.1.1.2 Computer Simulations

A computer simulation is a software implementation of a model. The computer simulation mimics the dynamic behavior of the component over time for a given environment. The development of good computer simulations is time consuming and expensive. Such simulations, therefore, represent a valuable resource that should be shared. Unfortunately, even if the underlying mathematical model provides an acceptable level of fidelity, there are many hindrances to reusing computer simulations. Incompatibilities between testbeds may result from the languages and hardware configurations chosen for the implementation and from performance requirements that are imposed upon the simulation.

The computer simulation may be written in a variety of languages such as assembly language, Fortran, Jovial, and Simscript. A simulation written in one language will not easily interface with simulations written in other languages. Furthermore, the language used for a computer simulation may not be supported on another testbed's computer system.

The implementation hardware imposes constraints on factors such as memory requirements and arithmetic precision. A simulation developed for a system with substantial memory capacity may not run on smaller systems. Differences in the internal representation of numbers may result in unexpected differences in the behavior of the simulated system.

Performance requirements can also seriously constrain the portability

of a simulation. A slower computer may not be able to process the simulation in an acceptable amount of time, or the more stringent time requirements of a new testbed may not be met by the existing simulation.

Each of the testbeds surveyed encountered problems of this type in trying to use existing simulations. In general, mathematical models are reused or enhanced whenever possible; computer simulations, however, are seldom reused.

2.1.1.3. Testbed Architecture

A major design question in developing a new testbed is the overall architecture of the testbed itself. Should the testbed be centralized or distributed? What type of equipment should be used? How might models be used in conjunction with live units? A logical beginning to the development of a new testbed is to study other testbeds. A good example of the evolutionary development of a particular testbed architecture can be found by tracing the family of distributed testbeds that began with TACS/TADS. The JINTACCS, IFFN, and REDCOM JTC testbeds each have a need to utilize distributed components and have, therefore, adopted architectures similar to TACS/TADS.

2.1.2. Empirical Data

In developing and validating models, actual data on the system being simulated is needed. This data ranges from design specifications to actual performance data. The data is needed to aid in abstracting

the behavior of the component during the development of the model, to provide parameters to model, and to validate the model's fidelity.

Performance data collected from actual tests provides further bounds on parameters and also provides a means to validate the model. Empirical data is needed on both hardware and human performance. Since actual tests are usually quite expensive, such data represents a very valuable resource that should be made available to a wide audience. The testbed community has a significant need to share empirical data.

2.1.3 Interface Requirements

A testbed must interface with a variety of hardware and software components. The testbed designer must determine the interface requirements for each component. For hardware components this may involve physical requirements such as pin assignments along with electrical requirements such as wave forms and signal strengths. For software components, the interface requirements might involve simulation parameters and their formats, processing requirements, and output formats. The testbed designer typically must contact the supplier of the hardware or software to determine these interface requirements. Each testbed that uses a common component may have to duplicate these tasks. Documentation on interface requirements is another example of a vital set of information that should be readily available to the testbed community.

2.1.4 Project Management

The development and use of a testbed may involve different services and many different agencies, contractors, and consultants. The problems encountered in manually monitoring each facet of the testbed's development and implementation can be quite significant. Many of these problems can be attributed to problems in gathering and distributing information through traditional communications systems such as telephone conversations, mail, and site visits.

2.1.5 Resource Sharing

A testbed requires personnel and hardware resources that must be scheduled in advance of a test. Since many resources are involved in a single testbed and since many testbeds may be contending for a single resource, scheduling and allocation of resources can be a tedious and demanding task. The testbed administrator must resolve conflicts between many different calendars. The problem of resource scheduling is likely to become more complicated as testbeds become increasingly complex and are utilized more fully.

2.1.5.1 Hardware Scheduling

Hardware scheduling may require moving hardware to a test site for a single field test, or it may involve making that resource available as a remote component of a distributed testbed like IFFN. In either case, there are typically many other demands upon the resource. Some resources such as the E-3A simulators at Tinker AFB and the training

domes at McDonnell Douglas Corporation in St. Louis are in great demand and require very careful scheduling. For a testbed to get maximum usage of the resource and for the resource to be utilized as profitably as possible, careful scheduling is essential. The chart in Fig. 5 vividly illustrates the increasing demand that will be placed on hardware resources in the future as more testbeds vie for the same resources.

2.1.5.2 Personnel Scheduling

A test also requires that many different teams of people be brought together. These teams include the participants (players), test monitors, umpires, control technicians, and a general support staff. Calendar conflicts for personnel are likely to be even more severe than that for hardware resources. The personnel involved in tests must also be advised in a timely manner on changes in schedule and requirements.

2.2 Interfaces to Testbed Components

Each testbed must establish interfaces to the components that will comprise the testbed. In cases where the component is actual hardware or an operational unit, the interface must be tailored to meet the requirements of that component and to satisfy the needs of the testbed. Interfaces to software components are usually tailored to meet the unique requirements of the testbed for which the software is being developed. Although this approach appears to be very reasonable and

	SYSTEMS	JINTACCS	TACS/ TADS	IFFN	SMARTS	MAINSITE	OED	REDCOM	INTRA-SVC: SW DEV	INTRA-SVC: TNG	INTRA-SVC: OP TEST	INTRA-SVC: DEV TEST	INTRA-SVC: INTROP	INTRA-SVC: CM
Army	SOTAS			X		X			X	X	X	X	X	X
	TACFIRE	X				X			X	X	X	X	X	X
	ACAS	X		X		X	X	X	X	X	X	X	X	X
	IN					X	X		X	X	X	X	X	X
	SIGMA					X			X	X	X	X	X	X
	PATRIOT	X	X	X		X	X	X	X	X	X	X	X	X
	AN/TSQ-73	X	X	X		X	X	X	X	X	X	X	X	X
Navy	NTDS	X	X				X	X	X	X	X	X	X	X
	ATDS(E-2B/C)	X	X	X			X	X	X	X	X	X	X	X
Air Force	407-L	X	X	X	X		X	X	X	X	X	X	X	X
	E-3A	X	X	X	X		X	X	X	X	X	X	X	X
	TIPI/DCSR	X	?	X	X		X	X	X	X	X	X	X	X
Marine Corp.	MACCS	X	X	?			X	X	X	X	X	X	X	X
	MIFASS	X	?				?	?	X	X	X	X	X	X
	ITAOC	?	?	?			?	?	X	X	X	X	X	X
NSA	RIVET-JOINT	X	X	X	X		X	X	X	X	X	X	X	X
	COMPASS EARS	X	X	X	X		X	X	X	X	X	X	X	X

Figure 5: Potential Remote Components and Testbed Needs

cost-effective for a single testbed, major problems are usually encountered in trying to reuse such software in other testbeds. Four broad categories of interfaces have been identified:

1. hardware,
2. software,
3. communications, and
4. man/machine.

Designers for each testbed investigated in this study have addressed the problems of defining interfaces to their various components, and each group has developed unique solutions to the problems.

The interfaces developed for a testbed are determined by the test goals for that particular testbed. A testbed which certifies hardware communications capabilities must incorporate as much of the actual hardware communications components as possible for the test results to have any credence. A testbed directed towards the evaluation of command and control procedures may use emulations of the communications system when convenient without compromising the test results. Such variances in test requirements result in significant differences in interface designs and specifications. For a component that could be shared by several testbeds, interface differences add to the cost and development time for testbed implementation. The usual result of trying to incorporate an existing hardware or software component into a new testbed is the development of a totally new interface. There has been no coordinated attempt to develop standards for interfaces that might be utilized by more than one testbed.

2.2.1 Hardware Interfaces

Part of the information gathering stage in developing a testbed requires the identification of the interface requirements of the hardware components. These requirements, however, simply define the inputs to and the outputs from the device. Broader design questions include how should the testbed stimulate the device, what information should be gathered from its outputs, and where should probes be inserted to collect data? The answers to these questions determine the nature and complexity of the interface between the hardware component and the testbed's central simulation facility. For testbeds such as IFFN and JINTACCS, the sophistication of the interface and the distance between the central site and the remote unit strongly motivated the decision to employ a remote processor to stimulate the unit and to extract information on its actions. Other testbeds such as CSEDS can directly control and monitor the actions of its hardware components because of their proximity.

2.2.1.1 Signal Injection

A standard practice in incorporating hardware into testbeds is to provide the hardware with a simulated external environment by injecting data from the central simulation facility directly into the unit. This technique, called stimulation, was used by each of the testbeds investigated. The most common application is to disconnect a live radar from its displays and replace the radar with simulated video signals that are injected directly into the displays. Similar techniques are used to stimulate other hardware components.

Stimulation requirements are determined by the nature of the hardware interface and the environment to be simulated. The physical and electrical interface requirements for an individual component will be the same regardless of the testbed. What will be different is the amount of data that is to be injected into the system and the response time requirements for a particular test. A radar display, for example, may need to display just a few tracks for one test or several hundred tracks for another different test. Data volume and response time requirements will determine the complexity and sophistication of the unit that will provide the stimulation signals.

2.2.1.2 Component Monitoring

Each testbed must also monitor and record the actions of its hardware components. In some cases this can be limited to simply intercepting the normal outputs of the component; the data can be collected by monitoring the signals emitted by the device. In other situations, a greater level of detail may be required and therefore probes must be inserted into the device itself. In these cases, the type of data that is to be monitored is probably unique to the requirements of the individual testbed.

2.2.2 Software Interfaces

In each testbed investigated the designers considered the use of existing software components wherever appropriate. The desire to use existing software is hardly surprising since software development costs

rapidly become a major component of the overall cost of a testbed. The software development process is a principal contributor to delays in reaching initial testbed operational capability since software development times are difficult to estimate. Testbeds can seldom use existing software since the software is usually developed with the intention that it will be employed only within a single testbed. Testbed software, therefore, is heavily intertwined with the testbed's structure.

A technique used by IFFN shows promise for the reuse of software. In the IFFN approach, simulations interface with their external environment through the same types of data channels that the actual hardware would use. A software simulation designed in this fashion would then employ the same interface that is used to attach its corresponding hardware component into the system. Another major motivation for developing simulations of this nature is to simplify the process of interchanging actual and simulated components between tests.

2.2.2.1. Distributed Software Simulations

Several testbeds incorporate simulations that execute on systems that are remote from the central site. This technique is usually used to reduce the amount of data that must be generated at the central site and transmitted to the remote unit. An example is multiple radar tracks that must be updated in real time. The remote simulation can maintain the radar displays by locally generating tracks that do not change speed or direction. The central simulation facility then needs to transmit only updates to tracks that have new velocity vectors.

Distributed simulations present unique implementation problems. Simulations that are executed at one central site can share state information through a common memory. Synchronization between the various components at the central site is then maintained with a single real or simulated clock that insures the proper time-sequencing of events. When components of the simulation are distributed between computer systems that do not share a common memory, the exchange of state information and the synchronization of the simulations becomes significantly more complex. The underlying reason for this added complexity is the time delays introduced by the communication system.

Communications time delays present two major problems:

1. the time needed to transmit state information between simulations may be measured in milliseconds for distributed simulations rather than nanoseconds for centralized simulations, and
2. the amount of time delay will vary, depending upon the distance between the simulations and the nature of the communications medium.

Time delays in communications can clearly have a significant impact upon the performance of distributed simulations. Performance can be an overriding consideration in testbeds that must meet real-time requirements. This implies that distributed simulations must

1. minimize the amount of data that is exchanged between the distributed components, and
2. correspond to components that would normally be geographically distributed and would experience natural communications delays.

The designs of the JINTACCS and IFFN testbeds reflect these requirements.

For distributed simulations, each simulation must insure that events occur in the proper order, based upon information that is received via the communications system from other components. In IFFN, for example, the radar display at a remote site is generated by a simulation executing on a processor at that site. The simulation periodically receives update information from the central site. When a simulation generates a local event, the synchronization scheme must take into account that an earlier event may have been generated at another component and that notification of this event may have been delayed by the testbed's communications system. For example, a radar track that is being updated by a local simulation may actually represent a plane that has already been destroyed by another site. As a result, the test may be compromised by personnel at a remote site who are acting on old or inaccurate information.

The IFFN testbed addresses the problem of event sequencing by requiring that the simulations execute in real time. If all simulations present the same real-time behavior as the components they are simulating, the components will experience the same time delays as would be experienced in an actual operational environment. This approach can present problems. The distance between components in a testbed may be much greater than would be the case in an operational environment. The added delays that the longer distances introduce must be compensated for within the testbed's design.

Another major problem with real-time execution occurs when messages are lost or distorted by the testbed's communications system. Real-time execution may not allow time for the retransmission of erroneous messages. Such communications problems cannot be dismissed since they reflect testbed communications problems, not operational communications problems. Real-time execution also imposes major design constraints on the software. The design, implementation, and testing of real-time software is far more complicated and expensive than that for the equivalent software that does not need to meet real-time constraints.

Several methods that do not rely upon real-time execution for the synchronization of distributed simulations have recently been developed. A survey of these techniques is presented in Appendix A. These methods may provide a means for relaxing the real-time requirements of distributed simulations. Unfortunately, each method appears to significantly increase the amount of information that must be exchanged over the communications system.

2.2.2.2. Central Site Simulations

Central site simulations provide the environment for the overall test and provide simulations of those components that are not physically present during the actual test. These functions are not generally distributed to the remote sites since the degree of interaction between these components is very high. The processing requirements of such simulations often demand that powerful mainframes and unique hardware configurations be employed to satisfy the performance requirements of

the testbed.

2.2.3 Communications Interface

Each testbed must establish communications between its central facility and its components. In a centralized testbed, this problem may involve little more than connecting cables between the components. In distributed testbeds such as IFFN, JINTACCS, and REDCOM's JTC, the communications problems can be quite substantial. These problems can be divided into two components:

1. the interface between testbed components and the communication system, and
2. the physical communications medium.

This section will discuss the communications interface; section 2.3 will address the communications medium as a separate issue.

Most testbeds must provide at least five categories of communications:

1. tactical data,
2. stimulation data,
3. collected data,
4. test coordination, and
5. voice.

Other forms of communications such as video may also be required.

2.2.3.1. Tactical Data

The testbeds investigated used standard tactical data communications links to transmit data between components. Most provided TADIL-A and TADIL-B links and some incorporated additional links such as TADIL-C, ATDL, PATDL, and LINK-14. These communications standards as well as their interfaces are well-defined. Additionally, the formats of the messages and their content are standardized.

The main problem in establishing these communications links is in the transmission of the data itself. TADIL-A, for example, can be a major problem to testbed developers since the use of the actual TADIL-A modems imposes time constraints on the data link. Several approaches have been adopted to overcome this problem such as including dummy units in the TADIL-A network or excluding the modem from the communications system. TADIL-C presents similar problems for long-distance communications. In the future, testbed designers will have to address the even more severe time constraints and bandwidth requirements imposed by TADIL-J. TADIL-J is projected to become operational during the lifetimes of current and projected testbeds.

2.2.3.2. Stimulation Data

The central simulation facility must continually transmit the current state of the test to the various testbed components. The state data may include stimulation data or the stimulation data may be derived from the state data at the local site. The stimulation data must then be interpreted and converted to signals that are injected into the remote

testbed component. Each testbed has its own method for providing stimulation data and its own format for the data.

2.2.3.3 Data Collection

Most tests require that data be collected on each unit's actions and then transmitted back to the central site. The data may be used as feedback to drive the overall simulation, for post-test analysis, or for test replay. Data that is used for post-test analysis can be collected at the local site and transmitted back to the central site after the test. Data that affects the test must be transmitted in real time. The nature of the data collected reflects the needs of the individual testbed. Some testbeds collect only the data transmitted on the tactical data links; others collect more detailed information of the internal operation of components. No community-wide standards currently exist for the communications of the collected data.

2.2.3.4 Test Coordination

Tests usually require that communications be provided to setup and administer the test. Communications must be provided, for example, between the central site and test monitors at the remote site. These communications have nothing to do with the actual operation of the testbed component. Such communications are usually informal and reflect the individual needs of the testbeds.

2.2.3.5 Voice (secure and non-secure)

Voice communication is typically employed between operational units. A testbed must provide voice channels for the live participating players and for simulated human components that must communicate via voice with the live units. This typically involves the use of test personnel who read either from scripts or from computer generated messages. The testbed may also be required to provide secure voice channels. Voice requirements were found to be similar for each testbed.

2.2.4 Man/Machine Interface

The testbeds surveyed involved a "man in the loop"; that is, humans participate as actual components in the test. The need for such participation is obvious for training systems and equally obvious for testbeds that are intended to test doctrines and procedures tc be carried out by humans. The participation of humans in a testbed adds constraints to the design and implementation of the system.

2.2.4.1 Equipment

The best interface between a testbed and a human participant is the actual hardware that will be employed in the field. Testbeds usually incorporate actual hardware wherever possible. When this is not possible, designers try to provide a hardware interface that is as clos to the actual system as practical. The use of actual hardware may be constrained by availability and practicality. An F-15 simulator, for example, may be more practical and cost-effective than an actual F-15

for a given test.

2.2.4.2 Real-time Requirements

Humans must be given the illusion that the environment that they are being presented is realistic. A major component of this realism is the "real time" response of the system. Radar screens, for example, must display tracks that change in an expected fashion over time and must react to actions taken by the human participant. This imposes a major constraint upon the testbed since data must be processed and acted upon within specified time intervals in order to give the human participant the illusion that the system is real.

2.2.4.3 Fidelity Requirements

Time is just one of the fidelity requirements that are imposed upon a testbed by the inclusion of human participants. For radar tracks the human typically should not be able to distinguish between real and simulated tracks based upon their actions. Voice communications from simulated units must correspond to that from actual units. Communications problems and capabilities should match those found in the real world. The testbed designer must strive to present the human participant with an environment that is as close to the real world as is practical.

2.3 Communications

Testbed designers must deal with the constraints imposed by the communications medium. These include testbed topology, communications bandwidth, transmission delay, and reliability requirements.

For both practical and economical reasons, the physical medium used for the bulk of testbed communications is leased telephone lines. The leased lines are maintained by the telephone companies but are otherwise under the control of testbed personnel. If conventional voice channels are used for data transmission, a modem is required to convert digital signals into analog signals that can be transmitted over telephone lines. With present technology, voice-grade circuits are capable of providing data tranmission rates up to 9600 bits/second. The public telephone network is capable of supporting digital data transmission at rates up to 56 Kbits/second. Unlike voice-grade circuits that require a modem for digital data transmission, the 56 Kbits/second links employ digital signaling techniques implemented by a data service unit (DSU) that is supplied by the common carrier.

For point-to-point interconnections, testbed architects must determine the volume of data to be transferred between sites and the speed required. The required bandwidth may be obtained using multiple low-speed lines as illustrated in Fig. 6 or a single multiplexed high-speed line as illustrated in Fig. 7. The parallel approach of Fig. 6 becomes expensive and creates major management problems since multiple lines with possibly different hardware configurations must be maintained. Moreover, the problem of data encryption is more complicated in the parallel approach. The multiplexed approach of Fig.

7 eases the management problem since there are fewer links to be maintained. Additionally, the data encryption problem is simplified since bulk encryption techniques can be employed. Secure voice lines are a by-product of this approach if voice channels are digitized and passed through the same multiplexer and bulk encryption system.

2.3.1 Testbed Communications Topology

The physical testbed topology is dictated by the testbed's mission. Geographically distributed testbeds such as IFFN and JINTACCS employ a star configuration with the central simulation facility at the center. The star configuration routes all communications through the central simulation facility. The central simulation facility acts as a communication switch and can therefore implement an arbitrary interconnection of testbed components. The star configuration provides several advantages:

1. It facilitates the solution of the problems relating to synchronization of testbed components and the sharing of global data.
2. It provides for arbitrary logical interconnections of testbed components and allows impairment of selected links for test purposes.
3. It facilitates the distribution of software modifications to the remote sites and eases the problem of data collection.

Testbeds may employ some combination of leased lines, switched lines, independently owned lines, or tactical data links. Both analog and digital signaling techniques are employed. In a testbed such as

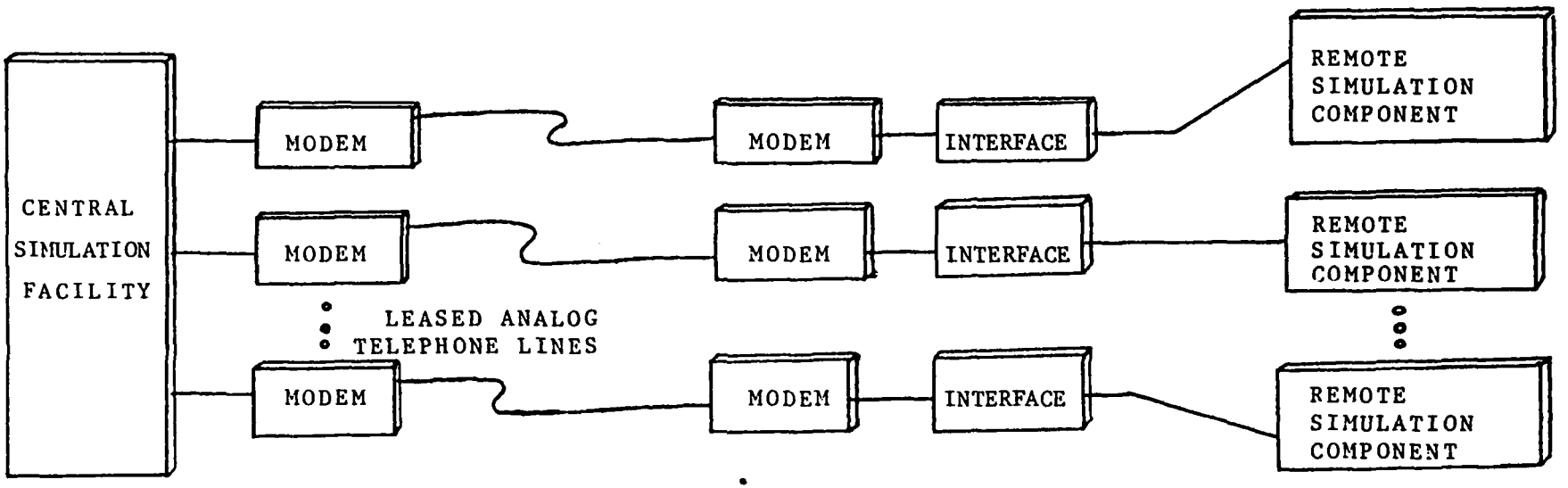


Figure 6: Multiple Low Speed Lines

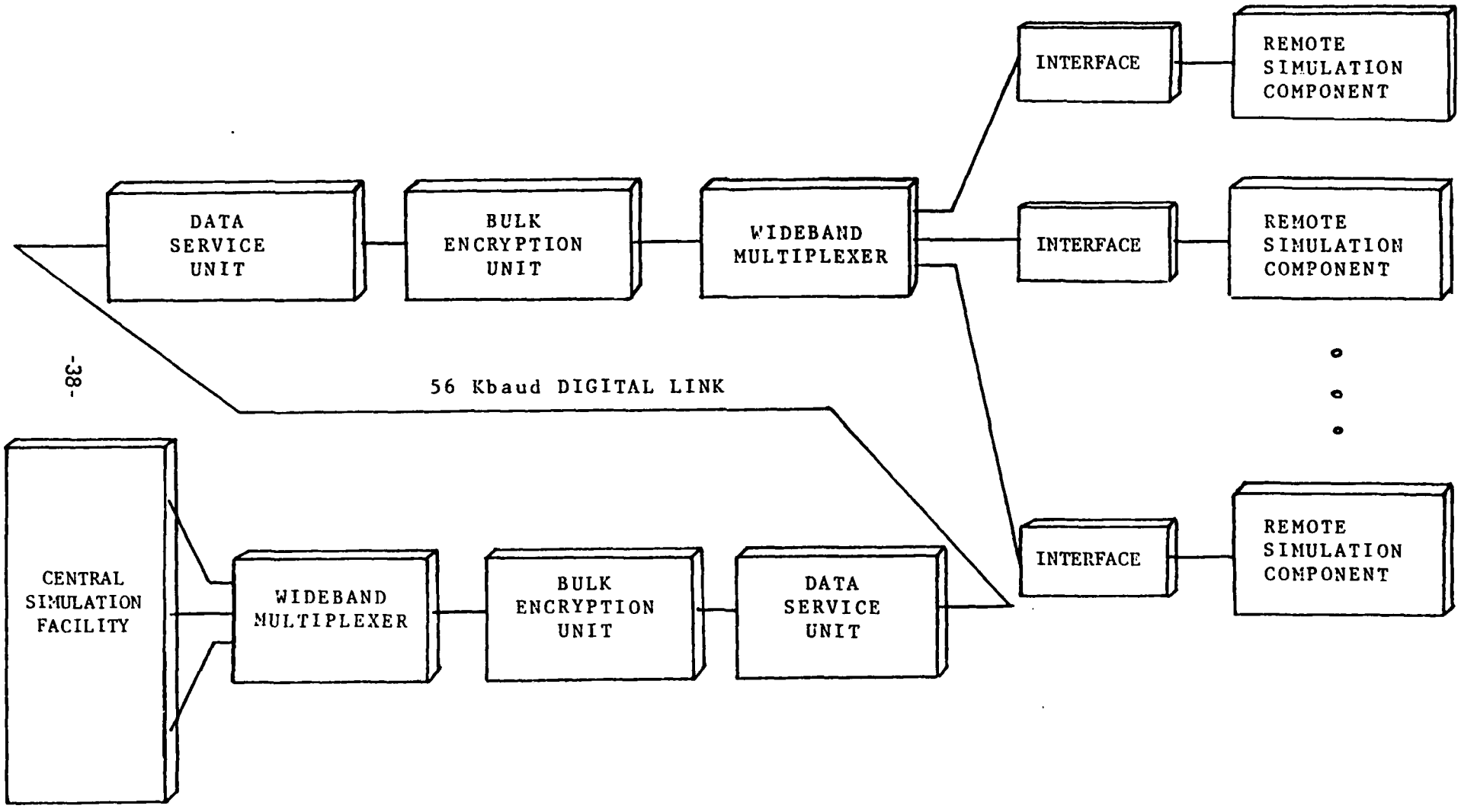


Figure 7: Single Multiplexed Line

TACS/TADS that makes use of actual ships, aircraft, and ground-based systems, the topology is determined by the physical arrangement of testbed components.

2.3.2 Transmission Bandwidth and Delay

Testbed designers are confronted with two distinct but related communication considerations: bandwidth and delay. Bandwidth refers to the rate at which information may be transferred while delay is the elapsed time between end-to-end transmissions across a communications link. A geostationary satellite link, for example, may exhibit a bandwidth equivalent to 20,000 voice channels; the end-to-end delay, however, takes roughly one quarter of a second. A coast-to-coast voice circuit incurs only a 20 millisecond delay but the bandwidth is limited to approximately 3 KHz.

The communications requirements for tactical data may be significantly different from those imposed by stimulation and monitoring needs. Current tactical data links require relatively modest bandwidths; however, the tactical standards impose time-critical signaling and protocol conventions. Stimulation and monitored data usually require higher bandwidths and minimal time delays.

Transmission time delays will result from the transmission channel itself (the physical link together with modems, multiplexers, encoders, and decoders) as well as from protocol conventions and the application of error control techniques. Studies have shown that human response time is on the order of 200 milliseconds which would seem to indicate

that a total delay of several hundred milliseconds would be acceptable for the majority of testbed communications.

Transmissions delays are not usually a problem for a testbed composed of simulation components in proximity to the central simulation facility. For geographically distributed testbeds, transmission delays are more significant, and testbed designers may have to make compromises to obtain transmission delays that are within acceptable limits.

2.3.3 Reliability and Availability

The reliability of a testbed's communications system and its components is a significant consideration since the failure or degradation of these subsystems can cause delays in testing ranging up to a few days. Given the number of hardware resources and personnel that may be involved in a test, such delays can have a significant effect upon the overall cost of a test program. Subsystems exhibiting intermittent failure can go undetected and can influence test results. Even though testbeds are designed using state-of-the-art commercial computer and communications equipment, reliability is a major concern to testbed designers.

A principal factor in the reliability of distributed testbeds is the leased telephone lines that the testbeds rely upon for their communications. The JINTACCS program has experienced substantial problems with the quality of their leased lines. They have also found that the lines must be continually used and tested to insure a

reasonable level of quality and availability. Leased lines provided by commercial carriers do not offer automatic error detection/correction capabilities or the ability to reroute data on alternate lines when the original lines fail before or during a test. Given the substantial costs of the communications network of leased lines employed by many testbeds, the performance obtained through this approach appears to be inadequate.

Testbed designers would like to further improve testbed reliability but can not always justify the attendant costs of additional hardware expenditures and added transmission delays that may be incurred because of the application of reliability improvement techniques.

3.0 Recommendations

This study has identified the following needs in the testbed community:

1. improved capabilities for information gathering and sharing on testbed models, data, and resources,
2. adoption of hardware and software standards where practical to encourage the sharing and reuse of existing hardware and software resources, and
3. further investigation of the communication and control problems associated with geographically distributed testbeds.

Several modern data processing and communications capabilities can be employed to address the above needs. The initial recommendations resulting from this study are that the potential of the initiatives listed below be explored.

1. The development and use of a community-wide electronic mail service.
2. The implementation of a database management system to catalog information on available models, data, hardware specifications, and testbed architectures.
3. The development and adoption of standards for the design and implementation of mathematical models and of both hardware and software testbed components.
4. The utilization of a computer based communications network to improve the reliability and availability of communications for geographically distributed testbeds.

3.1 Electronic Message Transfer

An electronic message transfer system is recommended to provide more timely communications and improved information management for members of the testbed community. Each participating installation should have a coordinator who is responsible for the electronic message transfer system. Part of this person's responsibilities would be to provide items or events of interest to the testbed community such as the project's newsletter, group visits, current activities or other pertinent information. The functions of an electronic message transfer system should be:

1. an inter/intra office mail system capable of creating, transmitting, editing, filing, searching, and printing mail and documents,
2. a public/group bulletin board system,
3. a public/group broadcast message system, and
4. a calendar system for scheduling activities.

The ability to electronically distribute mail and documents would introduce a proven technology to the test and evaluation community. Several studies have shown a productivity improvement after an organization initiates use of an electronic mail system, especially with respect to a more timely sharing of information. A bulletin board system would allow groups to post items of general interest to the entire community or a selected group. A broadcast message system would automatically inform users of urgent or global information. A calendar system would allow the scheduling of activities of test beds, meetings or resources in an orderly, productive manner.

Several electronic message transfer systems should be evaluated to determine their potential applicability. There are three independent components of an electronic message transfer system that should be evaluated independently of each other: communications medium, the computer terminal equipment, and the computer software. The communications medium evaluation should include a value-added network such as Telenet or Tymnet and a common carrier. The computer software evaluation should include several commercially available software mail packages. The computer terminal equipment that should be evaluated include a dumb terminal, a smart terminal capable of full screen editing, a small portable terminal suitable for carrying on an airplane, and several personal computers capable of off-line editing and down loading documents.

3.2 Database Management System

A database management system developed, implemented, and utilized by the test and evaluation community would enhance the sharing and standardization of information within the community. In a database management system, information can be stored in a computer system in such a manner that the information can be searched, correlated, and retrieved from a database. Retrieval of information can usually be accomplished in an English-like language, thus allowing non-computer personnel access to the database in a cost-effective manner. Additionally, if a database management system were part of a network of information sharing services such as an electronic mail system, the database could be accessed remotely by testbed personnel over a

computer communications network. This would allow testbed personnel across the country to obtain information on a timely basis.

A database management system would provide a central, standardized repository of pertinent test and evaluation information. Categories of potential information that would be shared via a database management system should be identified. Some possible categories might be the following:

1. simulations,
2. models,
3. test data,
4. documents,
5. scenarios,
6. software tools, and
7. characteristics of testbed components.

Many times the test and evaluation community is unaware of the existence or applicability of simulations and models that have been developed by the community. As a result, one of the first tasks in the development of a testbed is a survey of simulations and models that may be applicable to the testbed. For example, the IFFN testbed at Kirtland AFB contracted PE Systems Inc. to do a simulation and model survey. The USREDCOM JTC at MacDill AFB also commissioned a simulation model survey as part of their initial preparation. In both instances, the respective units were trying to gather information on simulations and models that might be pertinent to their activity. In addition to the inclusion of the simulations and models available, the

corresponding documentation and certification records should be included in the database.

The availability of simulation data along with a comprehensive description of the data would be of value to the testbed designer. The ability to share scenarios, especially if those scenarios were written in a high-level language, might make the testing between testbeds more uniform. Each testbed uses or develops software tools to help accomplish its job. It seems reasonable that these tools might be applicable to other testbeds. Rather than reinventing the same tools, a sharing of those tools, even if they needed to be slightly altered, would seem to be a step in the right direction.

The interface characteristics of interface components need to be documented and that documentation should be made available to the testbed community.

As an initial thrust on the database recommendation, a preliminary study should be initiated that would concentrate on the following questions:

1. What data would the installation have to contribute to a database?
2. What potential data would be of use to the installation?
3. What attributes of the data would be of interest to the installation?
4. What relationships between the data attributes would be of interest to the installation?

After the initial study, a compilation of the desired data categories along with their availability, attributes, and relationships would provide the basis for a comprehensive questionnaire. This questionnaire should

be sent to each installation to evaluate the potential of a database management system.

3.3 Adoption of Standards

The adoption of standards would promote the reuse of resources that have been developed by the testbed community. Standards also offer greater flexibility in testbed design in that testbeds can evolve and incorporate new components as they become available. Existing resources that do not exactly meet the fidelity or performance requirements of new testbeds could be used during the development and testing stage. This would allow the development process to proceed without waiting for key resources. The reuse of existing resources would reduce the overall cost of testbed development as well as the time needed before a testbed produces meaningful results.

3.3.1 Model Specification Standards

After identifying the testbed objectives and specifying the testbed architecture, the testbed designers can begin to develop the mathematical models for the simulated components. The standard system analytic approach to this process is to view the component as a "black box" with internal states, inputs, and outputs. The outputs at the current point in time are functions of the current internal state of the black box, while the internal state of the black box at the next point in time is a function of the current internal state and the current inputs.

A mathematical model of the component identifies the inputs, outputs, and internal states and specifies their relationships. In developing a model with a high level of fidelity, it is usually necessary to decompose the model into submodels. The testbed philosophy suggests that in principle it should always be possible to replace a simulated component by operational hardware. This attitude dictates that decompositions should always be "natural" (i.e., the inputs and outputs should always be physically measurable). This in turn suggests that the modeling process, in particular the documentation of the model, can be standardized. Such standards should certainly include the requirements that all inputs and outputs are listed, that all parameters are identified, and all relationships are defined. The advantages of standardizing the documentation include:

1. Retrieval of models from a database would be facilitated.
2. The entire model's survey process would be more efficient and effective.
3. Complete specifications should reduce software development time and costs.
4. The resulting software would likely be more standard thus increasing its potential for reuse.
5. Verification of the fidelity of the model would be eased.

The potential of standards for model specification needs further investigation.

3.3.2 Software Standards

DOD has initiated several studies in recent years that address the

problems of software development within DOD and its associated costs. The new programming language ADA* is an example of the results of one such initiative. The adoption of software standards could result in savings from the reuse of existing software and from the ability to more easily modify existing software to meet new needs.

3.3.2.1 Programming Methodology

Several new software design and implementation techniques that improve programmer productivity have developed in recent years. These techniques not only decrease the time needed to develop an operational program, but they also result in software that is easier to modify and upgrade. Several groups within DOD and more specifically within the testbed community have adopted various programming standards and are requiring that their programmers and contractors adhere to these standards.

The advantages of adopting a community-wide set of programming standards should be investigated. The use of a single set of standards would benefit individual testbed projects as well as increase the potential for reusing or modifying existing software.

3.3.2.2 Programming Languages

The testbed community should identify the smallest practical set of programming languages that can be used to develop testbed software.

---

* ADA is a registered trade mark of the Department of Defense.

These languages should represent languages that are generally available on both large and small machines. The capabilities and applicability of the new DOD-sponsored language ADA should be carefully considered. Limiting the set of languages used for the development of testbed software should greatly improve the ability to share software resources.

3.3.2.3 Simulation Interfaces

The ability to share and reuse simulations would be greatly enhanced if simulations communicated with their external environment through standard methods. In the IFFN testbed, simulations of testbed components communicate using the same formats and through the same communications channels as their equivalent hardware components. This is a good example of a technique that can be exploited to promote the reuse of software simulations. The additional costs of this technique must be balanced against the potential benefits of reusability and flexibility.

The use of generic models to construct simulations of specific testbed components also provides opportunities for the reuse of software simulations. A good computer simulation of a generic model could be utilized for many different applications by simply modifying the parameters of the model. The testbed community should consider the benefits of developing sound generic models.

3.3.2.4 Distributed Software

The reduced cost of hardware and the increasing sophistication of testbeds is encouraging the development of distributed software. In both the JINTACCS and the IFFN testbeds, remote processors are employed to drive data displays and to collect performance data in real time, thus reducing the load on both communications lines and on the central-site computer. Distributed processing introduces a new set of problems, the most severe of which is process synchronization.

Research into the problems of process synchronization is in its infancy and the implications of this research for the design of distributed testbeds need to be investigated. The successful development of practical techniques for distributing simulations will result in:

1. the capability to more effectively exploit the potential of low cost micro-processors,
2. the potential for reducing the amount of data that must be communicated between distributed components of a testbed, and
3. the ability to utilize existing simulation at a remote site and to take advantage of the expertise at that site.

The methodology for distributed simulations should be developed before a proliferation of different methods pervades the community.

3.3.3 Communications Standards

A communication standard would specify protocols, hardware

interfaces, and message formats. This study has identified the resources in Fig. 5 as a focal point for establishing standards. Currently, JINTACCS is using the RIU and the ESU to interface into remote hardware components and IFFN is developing the SSU to interface into its remote components. Some of these remote components such as the E-3A simulators and the PATRIOT are common between the two testbeds. Unless communication standards are agreed upon, there will be a proliferation of interface units, protocols, and message formats that must be supported by each remote resource.

Standards must address the issue that the resources may be used in the different testbeds in different ways. One testbed, for example, may incorporate a resource by using actual tactical equipment while another may use the same resource and simply pass the equivalent information over a leased line. The use of small computers within the remote component hardware interface provides the potential of having individual testbeds down load control and data monitoring programs and parameters that could tailor the interface to the needs of the testbed.

A communications standard for interconnecting shared resources to the central simulation facility would:

1. reduce time and cost necessary to implement a testbed,
2. make scheduling of scarce resources easier since each testbed could access the resource via the same interface, and
3. reduce the interaction required between the organization maintaining a resource and testbed designers in order to incorporate a remote simulation component into a testbed.

The standard selected for the hardware interface and the communication protocols should be a DOD or international standard; the message formats, however, would be determined by the needs of the testbed community.

3.4 Computer-Based Communications Network

A computer network can provide a basic framework for developing testbeds. A computer network would:

1. allow a remote simulation component in one testbed to be easily incorporated into another testbed,
2. simplify the job of providing interoperability between testbeds,
3. greatly enhance communications reliability and availability, and
4. potentially reduce the overall costs of DOD testbeds by reducing the number of dedicated communications links needed within the community and by increasing the productivity of the testing procedure.

A network should not be thought of as a dedicated set of communications links but as a delivery service that guarantees the timely delivery of information between sites served by the network. The physical communications system may incorporate dial-up lines, leased telephone company lines, microwave links, or satellite links, but the user is presented with the illusion of a dedicated circuit. The

computer-based network provides the automatic services of error detection and correction, message routing, and load balancing. The services provided by a network relieve the testbed architects of the problems attendant with the use of traditional communications channels and thus allows them to concentrate on the testbed mission.

A communications network may be viewed as a series of layers that are responsible for implementing a particular portion of the network according to a prescribed protocol. The most commonly used model is the ISO reference model as shown in Fig. 8. The model describes seven layers with each providing transparency in operation to the layer above it. Interaction between layers is possible only through well defined interfaces. The layering approach allows complex network design problems to be partitioned into independent, manageable modules that can be integrated to form the final system. Furthermore, the layering approach gives the designer the flexibility to alter layers in the structure, to incorporate technology advances or to enhance services, without affecting the other layers of the network.

With such a general purpose network, software designers can allow a process running on one computer to communicate with a process on another without having to understand the details of connection management, error control, flow control or routing. This makes it possible to develop application software that has no dependency upon the physical network. The software designer need only know the conventions for passing messages from one process to another. This general approach provides for the development of software that is independent of the physical network connections.

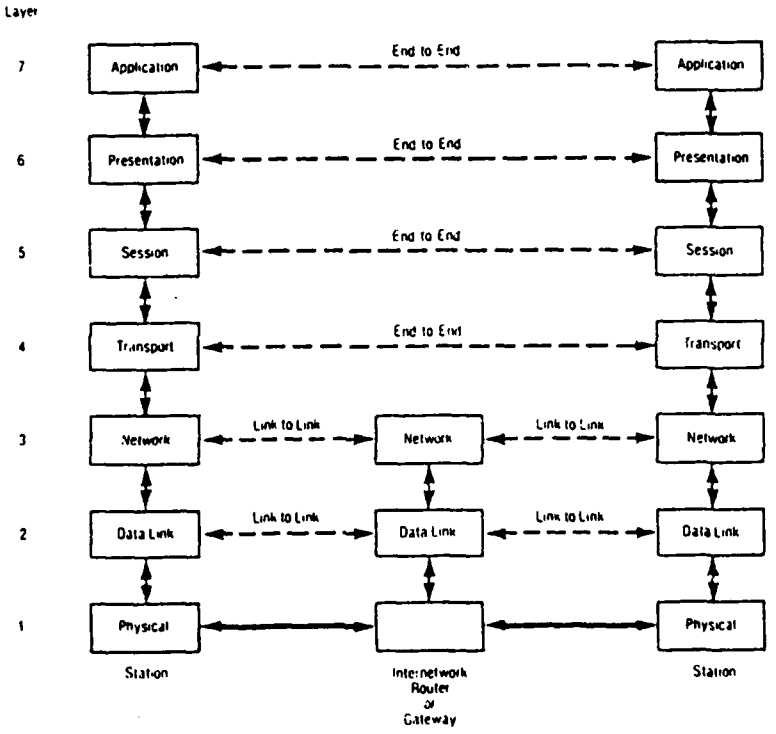


Figure 8: ISO Reference Model

A primary technical problem in designing a general network for the testbed community is the delay that the network incurs as a result of network overhead functions such as error detection and correction, routing, loading balancing, and packet assembly/disassembly.

The desirability of developing a computer-based communications network that could support many of the activities within the testbed community should be further investigated. This would include determining the particular characteristics of a network that will meet testbed communications needs and the development of algorithms that would provide efficient methods for controlling and synchronizing distributed simulations.

APPENDIX A

Distributed Program Synchronization

A.1 Motivation for Study of Geographically Distributed Programs

The field of distributed simulation is relatively new. Recent technological advances have created the ability to open an efficient and cost effective data path between geographically distributed programs on an interactive basis so they may cooperate in their processing. There are several important reasons why it might be desirable to consider this capability.

First, the components can be distributed based on the idea that some are more suited for execution within one type of computer system while others will execute more effectively in others (e.g., machines with floating point instructions). This allows the most cost effective performance system to be created. Advantages of such structures include flexibility to grow and to add or replace systems as requirements change.

Secondly, the behavior of many physical systems can be logically explained by discrete interactions between entities, but the mathematical methods are not available to represent these relations in a formulation from which general deductions can be made. Queueing systems are an example of this situation. Queueing systems are models of processes in which customers arrive, wait their turn for service, are serviced, and then depart. With simulation, it is possible to formulate a computer model for queueing systems. In many queueing systems, different

servers work simultaneously. This inherent parallelism can be exploited by developing the communication interconnections of these systems. In addition to increasing the execution speed, distributing the servers also provides a one to one correspondence between the real system and the simulator. It will be more cost effective if the memory requirements and the overhead due to interprocessor communication are minimized.

Figure A-1 illustrates some important results of recent technological advancements. Computing costs and communication costs are both decreasing, with computing leading the way. It is now reasonable to use local computing facilities for local processing and communicating the results as opposed to sending all the data over the communication lines to a central computing facility for processing. Software costs, however, are constantly increasing. It would be of great significance, from an economic view, if different users could log in over a network and interface with a geographically distributed program. This is especially attractive when the program is large and complex, written in an exotic language, machine dependent, or embedded in a web of libraries, special system calls, and other nonportable environmental features. Simulation programs often fall into this category. It is desirable to have the capability of interfacing with any number of simulation programs while allowing the developing, producing and maintaining of the simulation programs to remain locally distributed with the programming expertise.

Thirdly, in the situation where each simulation represents a distinct entity in the real system and the simulation requires real time interactions, it is ideal to be able to model a real system that is

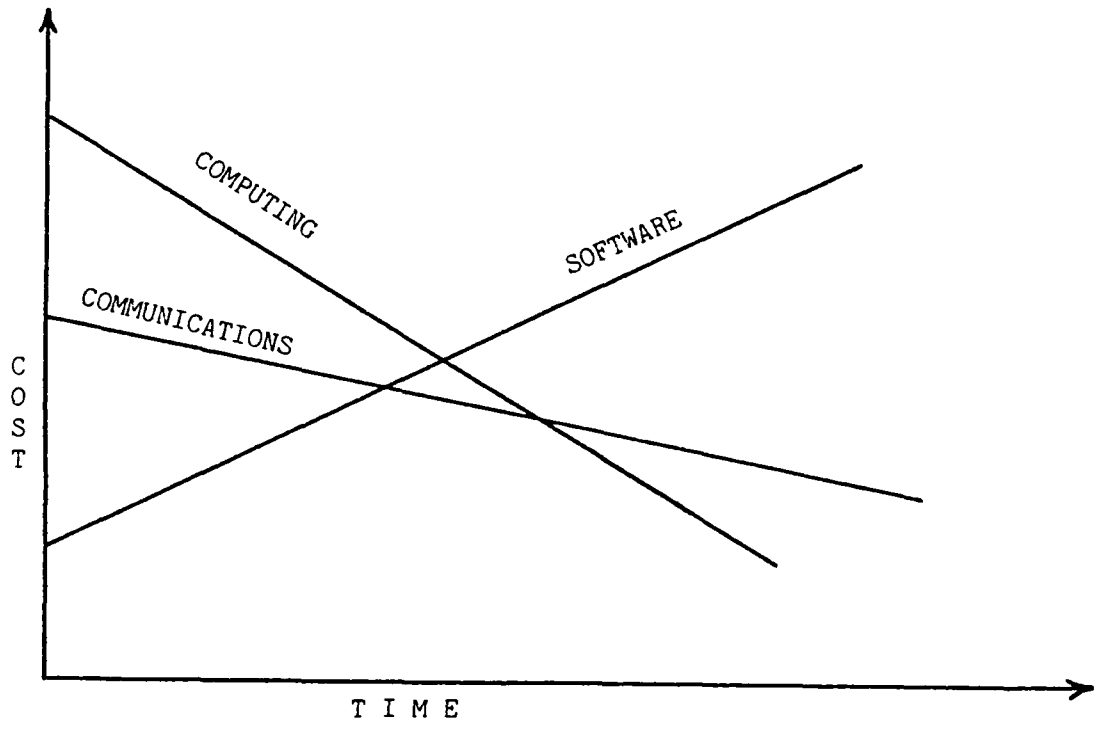


Figure A-1: Cost Trends in Computing and Communications

geographically distributed with geographically distributed simulations. This makes it possible to have each of the simulations physically located at the site of the corresponding entity in the real system. Therefore, those who would normally interact with a particular entity in the real system (i.e., those with expertise in that specific area) can interact, from the same location, with the simulation modeling that real system entity.

The concern with implementing discrete event simulations is threefold: communications, synchronization, and standards for simulation interfacing. Our current efforts are in synchronizing distributed simulations but the other two problems are mentioned briefly before beginning a discussion of synchronization.

A *distributed program* is a collection of processes which work on a common problem by communicating with each other only through messages and without sharing any global variables. Techniques must be developed defining both a standard format for these communication messages and the actual physical mechanism for passing communication messages from one process to another. Defining a standard methodology for simulation interfacing is of extreme importance. This will involve the specification, development, utilization and validation of computer simulations as well as the services of resource sharing.

A.2 Overview of Synchronization Algorithms

A.2.1 Background

The term *synchronization* refers to the mechanism used by cooperating entities to order events. The distribution and coordination

of state information is one of the main sources of problems in a distributed system. In a tightly-coupled distributed system, using shared memory, all cooperating entities have a common time. This is not the case in a loosely-coupled distributed system. Each entity in the cooperating set maintains a different view of its total states, due to arbitrary delays for message updating or reporting states. This essentially prevents any shared variables and requires message-passing for data communication.

A few distributed synchronization algorithms have appeared in the literature. Most are yet to be implemented in an actual distributed computer environment. Chandy and Misra [7,9] and Peacock [25], have defined and designed several algorithms for simulating queueing systems in a distributed manner. Some of these algorithms have been implemented/simulated on uniprocessors and some study on the performance of distributed algorithms has been done by Seethalakshmi [28].

A.2.2 Classification/Terminology

A scheme for classifying discrete simulations is proposed by Peacock [25]. This taxonomy divides simulations into categories based upon whether they are event-driven or time-driven. Furthermore, event-driven simulations are categorized as either tight or loose and time-driven simulations can be either scaled or unscaled in Fig. A-2.

Event-driven simulations enter a new state exactly when an *event* (a change in the state of the system) occurs; the time value for the state coincides with the time value associated with that event. In a time-

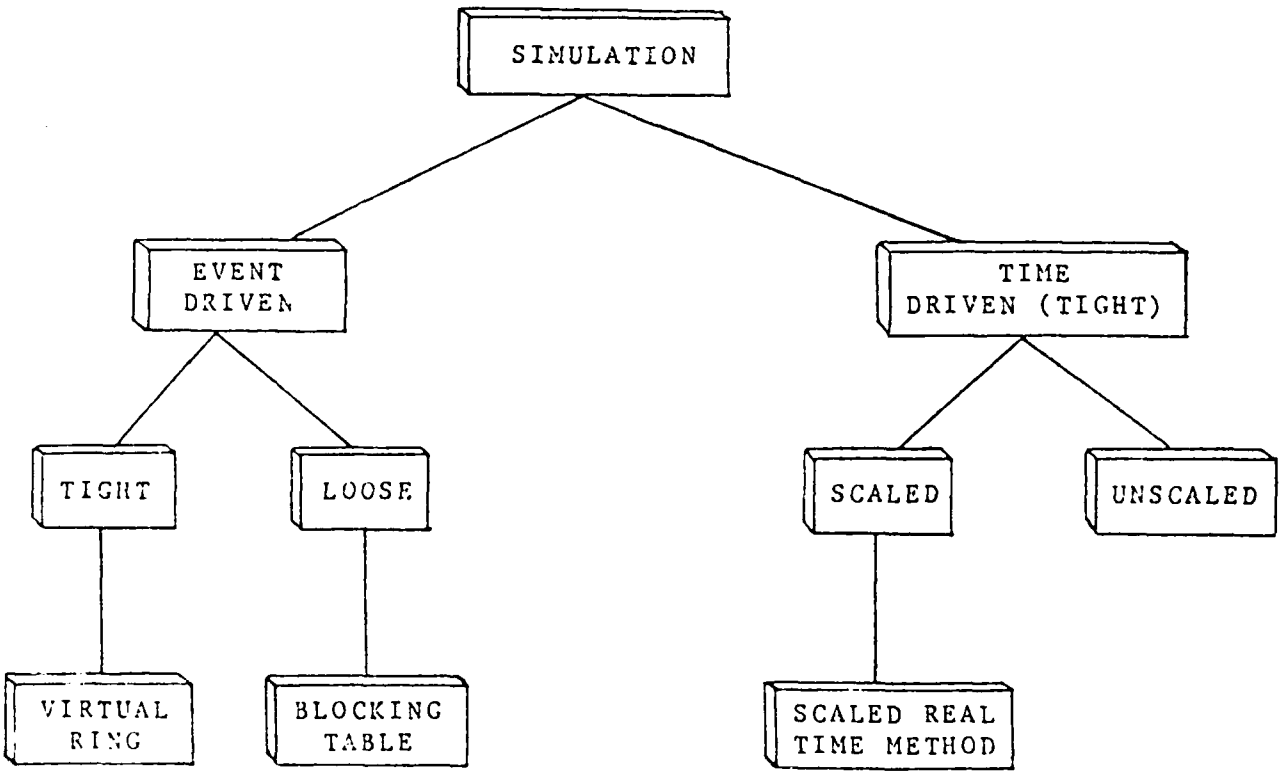


Figure A-2: Taxonomy of Simulation Classes

driven simulation, there is a constant positive change in the simulation time as the simulation progresses from one state to the next with zero or more events occurring in each state.

An event-driven simulation is *tight* if all processes in the simulation have the same simulation time, and *loose* if each has its own simulation time value. A time-driven simulation is *scaled* if there is a relationship between simulation time and real time whereby each increment of simulation time takes a constant amount of real time.

A.2.3 Scaled Real Time Algorithm

The scaled real time method is a technique for scaling time-driven simulations, allowing observation of the simulation dynamically. A hard wired timer which is able to generate interrupts must be set at the beginning of each event to kx/q where x is the simulation time for the event and q is the interval of real time units that are measured by each decrement of the timer. The algorithm causes a customer entering an empty queue to be processed immediately by setting the timer appropriately and suspending execution until the service time has expired. Other customers that arrive are queued until the server is free. This algorithm may result in nondeterminism when it is used in distributed simulations since the timing may vary from machine to machine.

In order to synchronize tight event-driven simulations without the use of a central controller, each process in the network must know, simply by the sending and receiving of messages, when it owns the next event to be processed--that is, the one with the lowest time value.

The virtual ring is one such synchronization algorithm that causes the orderly processing of the events in the simulation.

A.3 Virtual Ring Algorithm

The virtual ring synchronization mechanism takes advantage of the fact that producers are given unique and permanent names. This defines a total ordering on the set of producers. This ordering may be used to view producers as being organized as on a ring or as on a loop. Each producer has a unique predecessor and a unique successor. Such a logical structuring does not imply any particular physical topology.

In order to implement this synchronization, the network of processors must be virtually configured as a ring apart from the physical configuration provided by the network. There are other problems that can arise which are not addressed in the algorithm presented in Fig. A-3. In particular, the concept of preemptive events which allow the output from one process to move to the front of the queue for another process thereby preempting the other events. A brief description of the algorithm follows.

Each node along the ring has a rank associated with it which corresponds to the number of other processes which have lower minimum event times. When the rank associated with a process is zero, that processor has the lowest minimum event time and is then permitted to process the event which is to occur at that time.

```
procedure:
    declare 1  message,
               2  source  fixed binary,
               2  rank    fixed binary,
               2  min     fixed binary;
    declare (node, node_min, node_rank) fixed binary;
    node_min, node_rank = 0;
    do while (true);
         if node_rank = 0 then do;
              node_min = min_event time( ):
              node_rank = node_rank - 1;
              message.source = node;
              message.rank = 0;
              message.min = node.min;
              write (message); /* asynchronous write */
         end;
         read (message);        /* synchronous read */
         if message.source = node then
              node_rank = message.rank;
         else do;
              if node_min < message.min then
                   message.rank = message.rank + 1;
              if node_min <= message.min then
                   node_rank = node_rank - 1;
              write (message);   /* asynchronous write */
         end;
    end;
end;
```

Figure A-3: Virtual Ring Algorithm

After an event occurs, the process sends a message around the ring which has the value of its next event time. As this message is passed around the ring, its rank is incremented by each process that has a lower minimum event time than the message minimum event time, with the result that the message returns to the originating node with the correct rank for that event time. The process then reads the messages

from other nodes as they are passed around the ring, decrementing its own rank until it reaches zero and the event can occur.

A.4 Implementation

The virtual ring algorithm was implemented at Clemson on a single VAX-11/780 operating under VAX/VMS Version 3.0. The programming language was FORTRAN but included several operating system subroutines. Four processes were executed simultaneously; this simulated the processes as if they were executed on four different machines. Each process has a random number generator that creates the event times for the process. No events are passed from one process to another; consequently, no allowance is made for preemption.

Each time an event occurs, the process at which it occurs sends a message with the event time to a fifth process, or test-driver, which simply serves as an output device to print the event times to a file as they occur.

The processes are configured as a virtual ring and the communication of messages between them is accomplished by use of the MAILBOX feature provided by VMS. The mailboxes are virtual communication devices which are assigned virtual channels and have buffers which can store a variable number of variable-length messages. The buffers are implemented as queues where the size is a parameter which can be controlled by the programmer at the time that the mailbox is created. With a virtual ring, each process can write only to the process on its immediate right and to the test-driver. It can read only from the process on its immediate left.

The reads and writes in these processes are performed via low-level language calls to operating system subroutines and can be user specified as either synchronous or asynchronous. These system services have optional parameters which can be defined by the programmer. They include a message length, message source, message destination, and status information. To date, the implementation has not been totally successful due to an apparent error in the way in which the system performs synchronous input/output.

A.5 Summary and Future Work

The immediate goal is to realize a complete and successful implementation of the virtual ring algorithm on a single processor. This includes extending the algorithm to *n* processes and incorporating a mechanism to handle preemptive events. Efforts are also underway to implement other synchronization algorithms in order to make quantitative comparisons of various characteristics of the algorithms. A network model will be developed and implemented to allow a means for studying the performance of these algorithms in synchronizing distributed programs in a network environment.

Several criteria will be used to evaluate the performance of the synchronization algorithms. One important characteristic is the degree of concurrency that can be achieved among distributed simulations for a given algorithm. Other characteristics which can be associated with the algorithms include the amount of network traffic required for synchronization and the degree of processor utilization. Of particular interest are the ratio of synchronization messages to actual event messages, the processing requirements for synchronization and the

memory requirements imposed by the algorithms. A final performance consideration is the idiosyncratic characteristics of these algorithms and includes requirements associated with initialization, termination, and synchronization-message length. Empirical data gathered from the network modeling system will be presented on each of these characteristics. Finally, at some time during the investigation of synchronizing distributed programs, there will be a review of the potential use of synchronizing algorithms that require common memory and/or a centralized controller.

APPENDIX B

ANNOTATED BIBLIOGRAPHY

[1] J.E. Ball, E.J. Burke, I. Gertner, K.A. Lantz, and R.F. Rashid, "Perspectives on Message-based Distributed Computing," **Proceedings of the Computer Networking Symposium,** Gaithersburg, Md., December, 1979, pp. 46-51. This paper describes the aspects of a networking system (RIG), designed and implemented by the authors, which allows isolation of users from the details of the network and system configuration. The authors also describe the styles of message communication which have evolved in RIG.

[2] D.A. Bennett and C.A. Landauer, "Automatic Integration of Multiple Element Radars," **Proceedings of the First International Conference on Distributed Computing Systems,** Huntsville, Al., October, 1979, pp. 507-513. The AIMER (Automatic Integration of Multiple Element Radars) project is a software model of radar tracking. This paper describes the current implementation (status) of AIMER.

[3] R.E. Bryant, "Simulation on a Distributed System," **Proceedings of the First International Conference on Distributed Computing Systems,** Huntsville, Al., October, 1979, pp. 544-552. A simulation method is developed in which concurrent processes interact only by message-passing. The proper sequencing of event simulations is maintained by two decentralized, asynchronous control methods.

[4] B.P. Buckles and H.D. Fitzgibbon, "Distributed Emulation Control: An Algorithm With ESP," **Proceedings of the First International Conference on Distributed Computing Systems,** Huntsville, Al., October, 1979, pp. 536-543. This paper describes an algorithm for maintaining the time synchronization in a distributed computing environment. The algorithm requires very little communication overhead and is based on the notion of logical clocks.

[5] J.S. Campbell, "Navy Command, Control and Communications Systems Integration, Test and Evaluation Site," **Proceedings 1981 Southcon,** Atlanta, Georgia, January, 1981, session 5/1, pp. 1-9. This paper is a description of the Navy's approach to some of the fundamental problems of Command, Control, and Communications Support. The problem addressed is how to integrate the Navy's multiple units that may be required to simultaneously participate in missions. This paper is a description of the Navy's efforts to build and examine a full scale mock-up of the Navy's Command, Control, and Communications system.

[6] K.M. Chandy, V. Holmes and J. Misra, "Distributed Simulation of Networks," **Computer Networks** Volume 3, Number 2, April, 1979, pp. 105-113. A general discussion of distributed algorithms for discrete event driven simulations is presented. The authors develop outline proofs of their correctness.

[7] K.M. Chandy and Jayader Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," **IEEE Transactions on Software Engineering,** Volume SE-5, Number 5, September, 1979, pp. 440-452. This paper describes a distributed solution to system simulation where processes communicate only through messages; there are no global variables and there is no central process for message routing or process scheduling. The proposed solution has been empirically found to be efficient in preliminary studies. The paper presents formal, detailed proofs of correctness.

[8] K.M. Chandy and J. Misra, "A Nontrivial Example of Concurrent Processing: Distributed Simulation," **Proceedings of the Second IEEE Computer Society International Computer Software and Applications Conference,** Chicago, Il., November, 1978, pp. 822-826. This paper is concerned with the pros and cons of writing distributed applications software. Most applications software is highly sequential due to the sharing of variables. Here the authors focus attention on one such application: discrete-event simulation. They show how to develop distributed software for this application and outline proofs that their distributed software is correct. Their goal is to develop guidelines for writing distributed applications software.

[9] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," **Communications of the ACM,** Volume 24, Number 11, April, 1981, pp. 198-206. An approach to carrying out asynchronous distributed time-driven simulation on multiprocessor message-passing architecture is presented. Every process maintains its own logical clock and there is no global synchronization mechanism such as a global clock. With this approach, a deadlock situation is possible and a method for detection and recovery is examined.

[10] T.S. Chow, "Integration Testing of Distributed Software," **Proceedings of the Twenty-first IEEE Computer Society International Conference - Distributed Computing,** Washington, DC, September, 1980, pp. 706-710. This paper examines how to test the interactions between processes which can be modeled by finite state machines. A method has been proposed which can be used to automate the generation of scripts for integration testing of two processes. Test coverage and test application issues are also discussed.

[11] D.D. Clark, K.T. Pogran and D.P. Reed, "An Introduction to Local Area Networks," **Proceedings of the IEEE,** Volume 66, Number 11, November, 1978, pp. 1497-1517. This comprehensive paper on local area networks discusses what local area networks

are, their structures, the sorts of protocols that are used with them, and their applications. It also discusses the relationship of local area networks to long-haul networks and the impact of local area networks on computer communications.

[12] R.P. Cook, "The Starmod Distributed Programming System," **Proceedings of the Twenty-first IEEE Computer Society International Conference - Distributed Computing,** Washington, DC, September, 1980, pp. 729-735. Distributed programming is characterized by high communication costs and the absence of shared variables and procedures as synchronization tools. Starmod is a language, derived from Modula, which is intended for systems programming in the network environment. The Starmod system attempts to address the problem areas in distributed programming by creating an environment which is conducive to efficient and reliable network software construction.

[13] C.E. Ellingson and R.J. Kulpinski, "Dissemination of System Time," **IEEE Transactions of Communications,** Volume COM-21, Number 5, May, 1973, pp. 605-624. This paper is an introduction to problems of synchronizing physical clocks at geographically separated locations.

[14] Gilbert Falk and J.M. McQuillan, "Alternatives for Data Network Architectures," **IEEE Computer,** Volume 10, Number 11, November, 1977, pp. 22-29. This paper examines various existing and developing network architectures to support distributed data processing. Particular attention is given to the functional and performance requirements of the network and the currently available technologies.

[15] J.E. Freeman and S.H. Starr, "Use of Simulation in the Evaluation of the IFFN Process," **AGARD Conference Proceedings No 268 Modeling and Simulation of Avionics Systems and Command, Control and Communications Systems,"** Paris, France, October, 1979, pp. 15-19.

[16] M.L. Green, E.Y.S. Lee, S. Majumdar, and D.C. Shannon, "A Distributed Real Time Operating System," **Conference Record Distributed Data Acquisition, Computing, and Control Symposium,** Miami Beach, Fl., December, 1980, pp. 175-184. A distributed real time operating system is described which supported a distributed computer simulation. The operating system was tailored to a ballistic missile defense application to achieve maximum performance with minimum overhead. But it may be extended to other distributed applications.

[17] W.H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," **ACM Computing Surveys,** Volume 13, Number 2, June, 1981, pp. 149-183. This paper is a survey of techniques relevant to the solution of synchronization of access to shared objects (concurrent control) and recovery of objects in the event of error or failure. Concurrent control methods which are examined are locking, time

stamps, circulating permit, tickets, conflict analysis, and reservation. The proposed approach to solving the recovery problems is based on a software structuring abstraction called the atomic action. Considerations for implementing atomic actions are discussed.

[18] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," **Communications of the ACM**, Volume 21, Number 7, July, 1978, pp. 558-565. The relationship of physical time and event ordering in a distributed system is examined. A partial ordering of the events in a system is defined on the relation happened before. An algorithm is presented for extending this partial ordering to a total ordering which can be used to synchronize events. This concept is applied to a simple synchronization problem.

[19] A.B. Maccabe and R.J. Leblanc, "A Language Model for Fully Distributed Systems," **Proceedings of the Twenty-first IEEE Computer Society International Conference - Distributed Computing**, Washington, DC, September, 1980, pp. 723-728. A model of program organization is developed. The goal of this development is to provide a foundation on which languages designed for fully distributed systems may be based. Such languages are intended to support the construction of modular programs which effectively utilize distributed systems. Although based on the concept of loose coupling of processors, the abstraction provided by this model will be useful for programming in other environments.

[20] W.C. McDonald, H.O. Welch, and J.D. Reynolds, "Distributed Simulation Host: A Distributed Testbed," **Proceedings of the IEEE Computer Society's Fourth International Computer Software and Applications Conference**, Chicago, Il., October, 1980, pp. 562-568. The authors present the results of an experiment in hosting a distributed simulation of a large, complex, computer-based weapon system with a distributed testbed composed of eight VAX 11/780 computers. The host provides extensions to the VAX/VMS operating system to support process-to-process communication, virtual terminal capabilities, and network operational commands and services. Major lessons learned during the experiment are summarized, and the performance of the distributed testbed is analyzed.

[21] R.M. Metcalfe and D.R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," **Communications of the ACM**, Volume 19, Number 7, July, 1976, pp. 395-404. This is a paper describing the Ethernet multiaccess channel, its design principles and implementation.

[22] G.J. Nutt, "An Experimental Distributed Modeling System," Xerox Palo Alto Research Center Report, Palo Alto, California 94305, January, 1980. Current trends in system design are leading toward the use of multiple computers (distributed components) to implement various kinds of information processing sys' s in

order to increase performance while providing autonomous computing facilities. This paper describes a modeling system used to investigate one form of distributed systems, namely distributed office information systems. However, many of the ideas presented transcend this application. The simulator portion of the modeling system is unique in that it is itself a distributed program.

[23] J.K. Peacock, Eric Manning, and J.W. Wong, "Synchronization of Distributed Simulation Using Broadcast Algorithms," **Computer Networks**, Volume 4, Number 1, February, 1980, pp. 3-10. A class of algorithms, called broadcast algorithms, are developed for synchronizing time-driven distributed simulation. Since the performance of a broadcast algorithm depends on the properties of the inter-process communication facility, three different types of communication facilities are investigated.

[24] J.K. Peacock, J.W. Wong, and E.G. Manning, "Distributed Simulation Using a Network of Processors," **Computer Networks**, Volume 3, Number 1, February, 1979, pp. 44-56. This paper presents a hierarchical classification scheme for various simulation methods and examines some of the problems of synchronization, deadlock prevention and inter-process communication associated with the methods. Several algorithms are presented and discussed for solving these problems.

[25] "Realtime System with Replay Monitors Air Combat Exercises," **Computer Design**, Volume 20, January, 1981, pp. 74-79. This paper describes a computer controlled system that allows training personnel to monitor the details of the combat training missions in real time and allows users to review events subsequently. The system allow users to obtain exact data on the relative motions of participating aircraft, the trajectories of simulated weapons, and the occurrence of simulated hits or misses.

[26] F.B. Schneider, "Synchronization in Distributed Programs," **ACM Transactions on Programming Languages and Systems**," Volume 4, Number 2, April, 1982, pp. 125-148. A method is described and illustrated for implementing synchronization in distributed programs where there is no capability or desirability for shared memory. Viewing process execution as a sequence of phases, a phase transition occurs when execution of one phase ceases and execution of another is attempted. The synchronization technique discussed in this paper is based upon these phase transitions.

[27] M. Seethalakshmi, "A Study and Analysis of Performance of Distributed Simulation," M.S. Report, University of Texas at Austin, May, 1979. This report is a description and detailed analysis on the performance of a distributed algorithm proposed by Chandy and Misra for discrete event simulation.

[28] D.A. Spencer, "Limitations on the Use of Distributed Environment Simulation as a Means of System Testing," **Proceedings of the First International Conference on Distributed Computing Systems**,"

Huntsville, Al., October, 1979, pp. 593-600. A common method of testing large computer systems which must deal with a complex external environment is to simulate that environment using another computer system which replaces the normal input/output interfaces of the system under test. This simulator receives output data from the system under test and makes appropriate changes to the state of the simulated environment. Reasons are given why it may be desirable to test a distributed system with a distributed environment simulation.

[29] A.S. Tanenbaum, "Network Protocols," **ACM Computing Surveys,** Volume 13, Number 4, December, 1981, pp. 453-489. This paper is a tutorial about computer networking and in particular about the notion of structuring a network as a hierarchy of layers. The International Organization for Standardizations Reference Model of Open Systems Interconnection (ISO OSI) is the model used as a guide for this introduction to network protocols

[30] T. Utsumi, "GLOSAS Project (GLObal Systems Analysis and Simulation)," **Proceedings of the 1980 Winter Simulation Conference,** Orlando, Fl., December, 1980, pp. 165-217. GLOSAS Project proposes the development of an International peace gaming simulation of energy, resources and environmental systems to provide decision makers with comprehensive information. The methodologies and geographically dispersed dissimilar computers will be interfaced and executed interactively and cooperatively, as parts of the total simulation required.

[31] Stuart Wecker, "Computer Network Architectures," **IEEE Computer,** Volume 12, Number 9, September, 1979, pp. 58-72. This tutorial analyzes some of the current conceptual and implementation developments in computer architectures from a top-down design viewpoint. The paper begins with a discussion of the user requirements and then developes a communication mechanism to realize the requirements.

[32] Stuart Wecker, "The Design of Decnet- A General Purpose Network Base," **Conference Record IEEE/ERA - ELECTRO 76 International Electronics Convention,** Boston, Ma., May, 1976, section 4-3, pp. 1-8. Decnet is a Digital Equipment Corporation software implementation of a general purpose networking base upon which many varied and diverse networks may be built. This paper discusses the Digital Network Architecture (DNA) and presents examples of how some common functions are implemented upon the network base.

[33] D.M. Wilcox, "Super-Resolution Algorithms in Distributed Real-time Signal Processing Systems," **Proceedings of the Society of Photo-optical Instrumentation Engineers,** Volume 241, 1980, 220-3. Many current infrared optical telescope systems require the resolution of closely spaced targets which are optically unresolved. The real-time implementation of a super-resolution application of this algorithm is defined and its performance characterized. The application of this algorithm in a distributed

signal processing environment is discussed and its simulation on a distributed computer testbed is outlined.

END

FILMED

4-83

DTIC