1-1-1991

OSI-based Communications Architecture For The Distributed Interactive Simulation Application Utilizing The ISODE: An Evaluation Of A Prototype

Margaret L. Loper

INSTITUTE FOR SIMULATION AND TRAINING

May 1991

An Evaluation of a Prototype OSI-Based Communications Architecture for the Distributed Interactive Simulation Application Utilizing the ISODE

M. Loper • D. Shen • J. Thompson • Dr. Henry Williams

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
[illegible] of Sponsored Research

IST-TR-91-16

INSTITUTE FOR SIMULATION AND TRAINING

An Evaluation of a Prototype

OSI-Based Communications Architecture for the Distributed Interactive Simulation Application Utilizing the ISODE

Margaret Loper • David Shen • Jack Thompson • Dr. Henry Williams

May 1991

IST-TR-91-16

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

AN EVALUATION OF A PROTOTYPE OSI-BASED COMMUNICATIONS ARCHITECTURE FOR THE DISTRIBUTED INTERACTIVE SIMULATION APPLICATION UTILIZING THE ISODE

Technical Report

Margaret Loper
David Shen
Jack Thompson
Dr. Henry Williams

May 1991

Institute for Simulation and Training
University of Central Florida

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER I INTRODUCTION 1
Statement of Problem 1
Distributed Interactive Simulation (DIS) 2
Open Systems Interconnection (OSI) 4
Introduction 4
OSI Layer Descriptions 7
Layer 1: Physical Layer 7
Layer 2: Data Link Layer 8
Layer 3: Network Layer 9
Layer 4: Transport Layer 10
Layer 5: Session Layer 10
Layer 6: Presentation Layer 11
Layer 7: Application Layer 11
Application Layer Infrastructure 14
ISO Development Environment (ISODE) 18
Computer Network Performance 20

CHAPTER II APPROACH 23
Methodology 23
Hardware Environment 23
Software Environment 24
Test Plan 25
Research of Problem 27
Summary of Relevant Research 27

CHAPTER III IMPLEMENTATION 28
Encoding Protocol Data Units in ASN.1 28
Encoding DIS PDUs 33
Experimental Constraints 39
Building the Distributed Interactive Simulation Prototype Application in ISODE 41
Naming and Addressing Information 43
Remote Operations Module 46

Abstract Syntax Module 47
Initiator and Responder Programs 48
Prototype DIS Communication Architecture Execution 50
SEND Program: Detail Description 51
Experimental Performance Analysis 54
Performance Statistics 55
Results and Analysis 58

CHAPTER IV CONCLUSIONS 65

APPENDICES 69

A. Distributed Interactive Simulation Protocol Data Unit Structures 70

B. SEND Programs and Databases 87

C. Abstract Syntax Notation One (ASN.1) Modules 147

BIBLIOGRAPHY 157

LIST OF TABLES

1. Commonly Used OSI and Non-OSI Protocols 13

2. DIS Entity State PDU 34

3. Length of DIS PDUS 40

4. Performance Data for Experiment 1 56

5. Performance Data for Experiment 2 57

LIST OF FIGURES

1. Open Systems Interconnection Reference Model 5
2. Application Layer Protocols 15
3. ISODE Model of OSI Protocols 19
4. Experimental Network Hardware Configuration 24
5. PDU Round Trip Path 26
6. ASN.1 Mappings 29
7. ASN.1 Module Components 33
8. Dynamic and Static Facilities of Distributed Applications 41
9. ISODE Static Facilities 48
10. Flow Diagram of the SEND Program 52
11. Experimental Network 58
12. Experiment 1: Message Size vs. Average Round Trip Time 61
13. Experiment 1: Message Size vs. Average Round Trip Time (Sample Size = 500) 62
14. Experiment 2: Message Size vs. Average Round Trip Time 63
15. Experiment 2: Message Size vs. Average Round Trip Time (Sample Size = 500) 64

ABSTRACT

The advent of computer networking has spawned a wide variety of applications and technologies. Among the most exiciting and promising of these is Distributed Interactive Simulation (DIS). This technology involves the interconnecting of simulated (virtual) environments which allow participants in one simulation to interact, in real time, with those in another simulation, and observe their interactions by means of out-the-window views of the individual simulators.

The key to the evolution and worldwide acceptance of this DIS technology will be the adoption and integration of standard computer networking methodologies (i.e., Open Systems Interconnection (OSI)) into the overall DIS system communications architecture design.

This paper presents results obtained from research into the building of a prototype communications architecture for the DIS application utilizing the ISO Development Environment (ISODE). The ISODE is a software implementation of the upper three layers of the OSI protocol suite, which runs on UNIX-based workstations. ISODE uses the Transmission Control Protocol/Internet Protocol (TCP/IP) lower layer communications protocols supplied by most UNIX systems to perform the actual inter-computer communications over Ethernet. The purpose of ISODE is to provide a working environment in which to experiment with the upper OSI stack layers. ISODE is a public

domain software package and source code is provided to allow for modification of the software.

This research thesis presents the following: 1) a description of the DIS application and its requirements; 2) a discussion of OSI concepts and their applicability to the DIS application; 3) a brief overview of the ISO Development Environment and its services and facilities; 4) a description of the prototype DIS architecture developed using ISODE; 5) a discussion of the experiment and test plan for evaluating the prototype DIS architecture; 6) data obtained and analysis thereof; and, 7) conclusions drawn from this research.

CHAPTER I

INTRODUCTION

Statement of Problem

The advent of direct computer-to-computer communications (computer networking) has opened the possibility of interconnecting many different types of computer-based systems. Until recently, most devices used in Simulation and Training (S&T), which usually contained an embedded computational resource, operated in a stand alone mode. Today there is a major emphasis being placed on the development of distributed S&T systems which are networkable. In this context, the term "networkable" includes the requirement that the S&T systems are capable of communicating (transmitting and receiving) information which can be interpreted by other devices attached to the network, thereby allowing for real-time open interaction between such devices.

To achieve simulator interoperability, a standard communications protocol is being developed. The Distributed Interactive Simulation (DIS) draft standard establishes the requirements and provides the rationale for the Protocol Data Units (PDUs) exchanged between DIS entities. Since this emerging standard is primarily concerned with Department of Defense (DoD) simulator interoperability, the concept of Open Systems and specifically the Government Open Systems Interconnection Profile (GOSIP) has become an important issue. As of August 1990, compliance with GOSIP is mandatory and binding for U.S. government procurement of new network products (computers) and services. GOSIP defines a common set of data communications protocols which enable systems developed by different vendors to interoperate and enable the users of different applications on these systems to exchange information. GOSIP is compliant with the International Organization for Standardization (ISO) standards.

The major goal of the DIS standardization efforts is to provide an open communications (network protocols) environment in which DIS vendors can develop standard DIS compliant products. The goal of the Open Systems Interconnection (OSI) approach is to provide international open standards for inter-computer communication. Therefore, it is important to study the relationship between DIS and standardized communication suites, such as, the OSI protocol suite. This research effort develops and evaluates a prototype OSI-based communications architecture for real-time Distributed Interactive Simulation applications. In particular, the universal data structures present in OSI communications are thoroughly investigated in terms of DIS applications. Tools and services available in the ISO Development Environment (ISODE) are used to design and and conduct the experiments performed in this research.

Distributed Interactive Simulation (DIS)

The Institute for Simulation and Training (IST), based at the University of Central Florida (UCF) in Orlando, has the DoD mission to standardize the information passed between simulators participating in a networked training exercise. This is being accomplished in a government-industry-academia workshop forum by synthesizing material from successful networked simulations and by detailed analyses of competing technologies. In August 1990, IST submitted a draft standard of DIS for government and industry approval. In January 1991, IST delivered the draft military DIS PDU standard [MCDO91].

The issuance of the draft standard is only a small step in the development of a usable public domain network protocol standard which will provide truly open interoperability between simulations. In fact, the DIS PDU Standard defines only the PDU data structure used by the Application Layer (layer 7) of the seven layer OSI network protocol stack. There is a great deal of work which needs to be done to specify, precisely, the network services required by the DIS application, and then to translate these requirements into an OSI solution.

Distributed Interactive Simulation also refers to the physical placement of the interactive simulators or simulations participating in a military training exercise. Networked simulator exercises can be executed in both Local Area Networks (LANs) and Wide Area Networks (WANs). A number of simulators may participate in an exercise at one time and these simulators must share information about the simulated world in which they are interacting. This information includes: Entity State, which contains information about the entity being simulated; Weapons Fire, which describes the type of munition fired, location of the weapon, and the velocity of the munition; Weapons Detonation, which is issued when the trajectory of the fired munition is terminated; Collisions, which is issued by a simulator when it determines that a collision has occurred between the issuing entity and another entity; Radar, which designates a radar is being used by the entity; and Repair and Resupply, which requests and acknowledges these services. Each of the twelve DIS PDUs has a header which specifies the identification number associated with the DIS exercise, the protocol version, and the type of PDU that follows. For the purpose of this paper, a generic PDU will be used in transmissions across the prototype OSI communications architecture stack. The DIS PDU data structures are included in Appendix A.

Open Systems Interconnection (OSI)

Introduction

The Open Systems Interconnection (OSI) Reference Model was developed in 1977 by the International Organization for Standardization (ISO) in response to the need to interconnect heterogeneous (developed by different vendors) computers. OSI defines a framework for the interaction of users and applications in a distributed data processing environment [ISO84]. This environment may include a variety of computer and terminal equipment, as well as many different kinds of communications technologies. The standards for connecting "open" systems for distributed applications are based on a structuring technique called **layering**, in which the communication functions of the network are divided into a hierarchical set of layers. Each layer performs an integral subset of special functions required to communicate with another layer of similar type. Two layers which correspond in this manner are called peer layers. The peer layers communicate by means of a set of rules or conventions known as protocols. Each layer in the OSI reference model relies on the operation and services of the adjacent lower layer to perform more primitive communication functions. The interface between these layers is known as the Service Access Points (SAPs). The seven layer OSI reference model is shown in Figure 1.

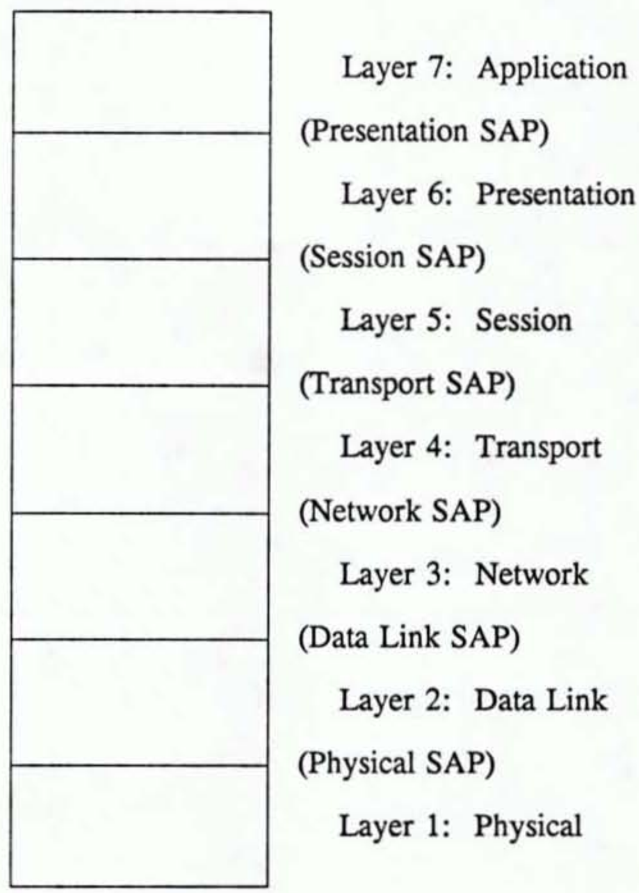

Figure 1. Open Systems Interconnection Reference Model

A brief definition of each layer of the OSI model, according to [STAL87], follows:

Layer 1: The **Physical Layer** is concerned with transmission of unstructured bit stream over the physical link. This includes the mechanical, electrical, and procedural characteristics to establish, maintain, and deactivate the physical link.

Layer 2: The **Data Link Layer** provides for the reliable transfer of data across the physical link.

Layer 3: The **Network Layer** is responsible for establishing, maintaining, and terminating connections.

Layer 4: The **Transport Layer** provides reliable, transparent transfer of data between end points.

Layer 5: The **Session Layer** provides the control structure for communication between applications. It establishes, manages, and terminates connections between cooperating applications.

Layer 6: The **Presentation Layer** performs transformations on data to provide a standardized application interface and to provide common communications services.

Layer 7: The **Application Layer** provides services to the users of the OSI environment.

In each of the seven layers, a layer service is defined to identify the set of functions provided by the layer. A service user of a layer is an entity in the adjacent higher layer. Layer services in OSI are of two general types: connection-oriented (CO), which allow the service users to establish and use logical connections; and connectionless (CL), which allow the service users to exchange information without having to establish a connection. A CO service is provided in three distinct phases:

Phase 1: Connection Establishment - The service user and service provider negotiate the way the service will be used. This is also referred to as a "binding".

Phase 2: Data Transfer - The service users exchange data.

Phase 3: Connection Release - The binding between users is discarded.

A CL service has only one phase, namely, **Data Transfer.** In a CL service, there is no ongoing relationship established between service users. Thus, there is no connection establishment or connection release phase in a CL service. The OSI application layer services are CO in nature. However, the lower layer services offer CO service with optional interfaces to CL protocols.

OSI Layer Descriptions

The following sections supply more information on each layer of the OSI reference model, such as the services provided within each layer and the most commonly used protocols. These descriptions are based on [TANE88].

Layer 1: Physical Layer. The Physical layer is concerned with transmitting raw data, i.e., unstructured bit streams, over a communication channel. The design of this layer involves such issues as signal voltage swing and bit duration. Mechanical, electrical, and procedural interfaces, as well as the physical transmission medium are also considered in the design of this layer.

One of the most common physical layer standards in use today is RS-232-C, which specifies a 25-pin connector between two devices. Also included in this layer are the IEEE 802 protocols, adopted by ISO (IEEE 802.3, 802.4, 802.5). These standards define the Carrier Sense Multiple Access With Collision Detection (CSMA/CD) (ISO 8802/3), Token Passing Bus (ISO 8802/4), and Token Ring (ISO 8802/5) protocols, respectively. The IEEE 802 protocols are used mainly in local area networks. The newest addition to the physical layer protocols is the Fiber Distributed Data Interface (FDDI), which specifies the use of optical fiber as the transmission medium. FDDI has been adopted by ISO and is specified by ISO 9314. The last commonly used Physical layer protocol is the CCITT Integrated Service Digital Network (ISDN). ISDN standards allow the integration of data, voice and video over the same digital links. ISDN spans the Physical, Data Link, and Network layers (Layers 1, 2, and 3) of the OSI model. In the Physical layer, ISDN is defined by the International Telegraph and Telephone Consultative Committee (CCITT) 1.430 and 1.431.

Layer 2: Data Link Layer. The Data Link Layer has the task of reliably transfering data across the physical link. This is accomplished by having the sender break the input data into data frames with the necessary synchronization, error control, and flow control information. Since the physical layer only accepts and transmits a stream of bits without regard to structure, it is the task of the data link layer to create and recognize these boundaries. This is accomplished by attaching special bit patterns to the beginning and end of a frame.

Some of the protocols which fall into this layer include the High-Level Data Link Control (HDLC), which is a synchronous bit-oriented protocol (ISO 7776) and the IEEE Logical Link Control (LLC) (ISO 8802/2). The LLC protocol is an IEEE protocol which has been adopted by ISO. In the IEEE standards, the LLC is used in conjunction with the Medium Access Control (MAC) protocol. The MAC manages the communication over the link, while the LLC manages the frame transmitted over the link. The MAC operates over both the OSI Physical and Data Link Layers. MAC is not an ISO adopted protocol. ISDN is also a common protocol for the Data Link layer and is defined by CCITT Q.921.

Layer 3: Network Layer. The Network Layer provides the upper layers with independence from the data transmission and switching technologies used to connect systems. A key design issue for network layer protocols is determining how packets are routed from source to destination.

The Network layer is a point of divergence between the ISO and non-ISO communities. Within the CCITT protocol suite, the X.25 (Layer 3) protocol is the standard for public packet switched networking for Wide Area Networks (WANs). X.25 has been adopted by ISO as a Layer 3 network standard (ISO 8208). The Department of Defense also has a protocol which falls into Layer 3 of the OSI model. This protocol is called the Internet Protocol (IP) and is responsible for internetwork routing and delivery. IP (defined in Request For Comment (RFC) 791) is not an ISO standard. The CCITT ISDN is defined in this layer by Q.931. There are two additional OSI Network layer protocols, Connectionless Network Protocol (CLNP) and Connection Oriented Network Service (CONS). CLNP (defined by ISO 8473) accomplishes the routing of messages by adding addressing information to each message, thus operating in a connectionless manner. CONS (defined by ISO 8348) allows the Transport service to bypass CLNP when operating over a single X.25 subnetwork.

Layer 4: Transport Layer. The goal of the Transport Layer is to accept data from the Session layer, divide the data into smaller units if necessary, pass this data to the network layer, and ensure that all pieces arrive correctly at the destination. These functions will be performed in reverse order at the destination. This transparent delivery of data between end points provides end-to-end error recovery and flow control. The transport layer also determines what type of service to provide to the session layer; connection oriented (CO) service with messages delivered in order or connectionless (CL) service with no guarantee about the order of delivery. The transport layer is the end-to-end or source-to-destination layer.

For this layer, ISO specifies the Transport Protocols (TP) which consists of 5 classes (TP0-TP4). These classes are defined in ISO 8073. The TP0-TP3 protocol classes work with a CO-mode network service, while TP4 works with both CO- and CL-mode network services. The OSI model also defines the Connectionless Transport (CLT) defined by ISO 8602. CLT provides a connectionless datagram service. However, the most widely accepted and used transport protocol is the DoD Transmission Control Protocol (TCP), defined in RFC 793. TCP, like IP, is not an ISO standard.

Layer 5: Session Layer. The Session Layer allows users on different computers to establish and use a connection, called a session. The Session layer provides the structure for controlling the dialogue or communication between applications. This dialogue may be two-way simultaneous, two-way alternate, or one-way. The Session layer can also provide a checkpointing mechanism so that if a failure occurs, the session entity can retransmit all data since the last checkpoint. The Session service is defined by ISO 8327. Examples of a Session service might be to invoke a remote log-in or to transfer files between two computers.

Layer 6: Presentation Layer. The Presentation Layer is concerned with the syntax and semantics of the data being transmitted between machines. Typically, computers have different methods for representing data (ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code), one's complement, two's complement, etc.) The job of the Presentation layer is to manage the abstract data structures which define the different representations and convert these representations between internal and external devices. Examples of Presentation protocol functions include text compression and encryption. The Presentation layer is defined by ISO 8823.

An integral part of the Presentation and Application layers is the concept of an Abstract Syntax Notation (ASN). An ASN defines data structures in a machine-independent fashion. Currently, there is only one ASN in OSI, Abstract Syntax Notation One (ASN.1). ASN.1 provides a formal notation for specifying the data that cross the interface between the application and presentation layer and is defined by ISO 8824.

Layer 7: Application Layer. The Application Layer supports the communication requirements of applications (i.e., information processing tasks) requiring co-ordinated processing activities in two or more open systems. The Application Layer is supported by the Presentation Layer, which contains facilities for representing information exchanged between application-entities (AEs) and the Session Layer, which contains the mechanisms that may be used for controlling interactions between AEs [ISO9545]. An AE is an aspect of an Application Process (AP). An AP is an element within an OSI-compliant system which performs the information processing for a particular application [ISO7498]. As the highest layer in the OSI reference model, the Application layer provides a means for the APs to access the OSI environment. Hence the Application layer does not interface with a higher layer. The purpose of the Application layer is to serve as the window between corresponding APs which are using the OSI to exchange meaningful information. APs exchange information in the Application Layer by means of AEs, application protocols, and presentation services [ISO 7498].

The current generation of OSI Application Layer protocols are based on a connection-oriented transport service with either a connection-oriented or a connectionless network service. The DIS protocol, which is currently being investigated, would logically fit into the Application layer. Some ISO Application services which are currently available include the following:

Directory Services (DS) is responsible for the management of names and associated attributes, such as addresses. A name is an explicit description of an entity within the application. Each application uses the DS to determine the presentation address of its peer. DS is both an ISO standard (9594) and a CCITT standard (X.500).

File Transfer, Access and Management (FTAM) allows network users to transfer files between heterogeneous systems and to access remote files and records on other systems. FTAM is defined by ISO 8571.

Message Handling System (MHS) is the standard for electronic mail and messaging between heterogeneous systems. MHS provides the capability of handling, transferring, and forwarding messages. MHS is defined in CCITT by X.400.

Virtual Terminal (VT) allows terminals on a heterogeneous network to interact with hosts regardless of terminal type. A user at one terminal could gain access to any host on the network. VT is defined by ISO 9041.

The most commonly used OSI and non-OSI networking protocols are summarized in Table 1.

TABLE 1

COMMONLY USED OSI AND NON-OSI PROTOCOLS

	OSI	CCITT	IEEE	DoD
Application	FTAM (8571) VT (9041) DS (9594)	MHS (X.400) DS (X.500)		
Presentation	(8823)			
Session	(8327)			
Transport	TP0-TP4 (8073) CLT (8602)			TCP (RFC793)
Network	CLNP (8473) CONS (8348) X.25 (8208)	(X.25) ISDN (Q.931)		IP (RFC791)
Data Link	LLC (8802/2) HDLC (7776)	ISDN (Q.921)	LLC (802.2) MAC	
Physical	802.3 (8802/3) 802.4 (8802/4) 802.5 (8802/5) FDDI (9314) RS-232	ISDN (1.430,1.431)	MAC (802.3) (802.4 (802.5)	

Application Layer Infrastructure

Within the Application layer, there exist Application Processes (APs) which perform information processing for a particular application. The communication aspects of theses processes are rpresented by Application Entities (AEs). The AEs are composed of one or more Application Service Elements (ASEs), which is the part of the AE which provides an OSI environment capability, using appropriate underlying services. [ISO7498] The way in which the ASEs interact with each other and with the underlying services defines the application protocol used by the application entity [ROSE90a]. An Application protocol is a service, such as file transfer. When invoked, an application protocol forms an application context with its peer system and is assigned an Application Context Name (ACN). Subsequently, the ACN is assigned an Object IDentifier (OID). An example of an application context name in the ISODE is ISO FTAM or ISO VT. In this experiment, the DIS will have an ACN of ISODE DIS, since it is an ISODE application, not an ISO application. The relationship between APs and ASEs is shown in Figure 2.

When a specific instance of an AP wishes to communicate with an instance of an AP in some other open system (i.e., AP1 in System 1 wishes to communicate with AP1 in System 2), it must invoke an instance of an AE in the Application layer of its own open system. It then becomes the responsibility of this instance of the AE to establish an association with an instance of an appropriate AE in the destination open system. This process occurs by invoking instances of entities in the lower layers. When the association between the two AEs has been established, the AP can communicate

[ISO7498]. It is important to note that each peer AP is composed of the same ASEs. Also, each ASE communicates only with its peer ASE in a remote system. This is accomplished by assigning unique presentation context information (PCI) to each ASE so that the application protocol data units (APDUs) can be delivered to the correct ASE.

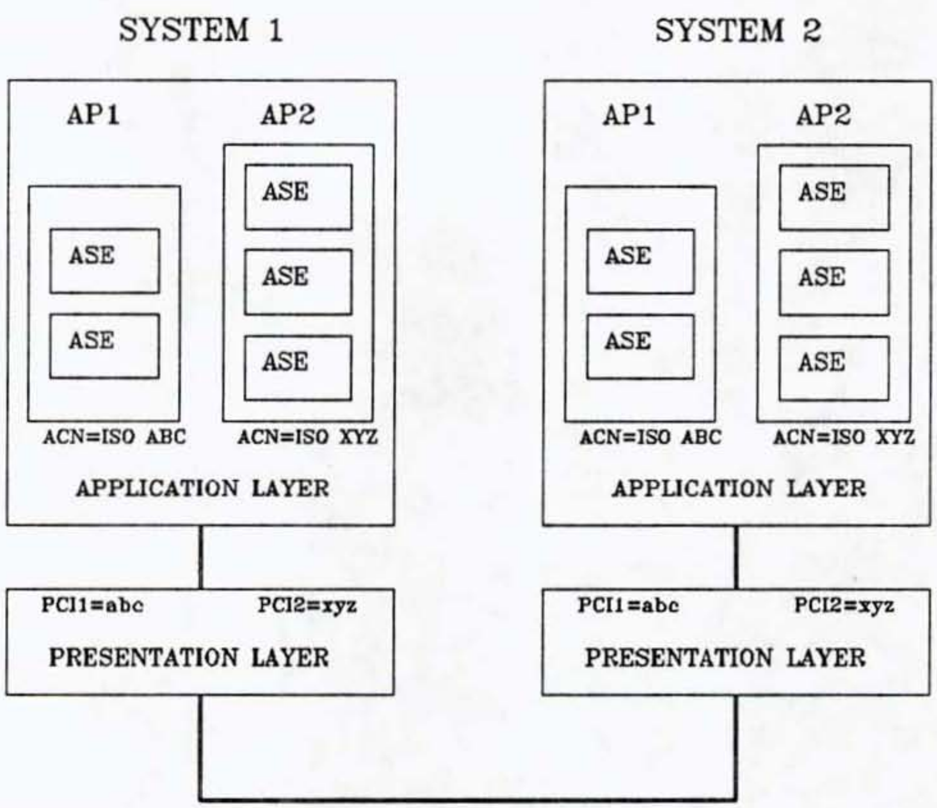

Figure 2. Application Layer Infrastructure

There are two categories of ASEs: common-ASEs, which provide capabilities that are generally useful to a variety of applications and specific-ASEs, which provide capabilities required to satisfy the particular needs of specific applications [ISO84]. Three common-ASE's currently exist in the OSI reference model for building application processes. They are described as follows [ROSE90a]:

Association Control Service Element (ACSE) establishes and releases the association to the remote or peer system. An association is a binding between two entities that is supported by an underlying presentation connection. The ACSE manages the Application associations. As a consequence, all OSI applications contain an ACSE. The ACSE has two phases: association establishment and association release. The ACSE is defined by ISO 8650.

Reliable Transfer Service Element (RTSE) is responsible for bulk mode data transfers between systems. The term "bulk mode" refers to the size of the data being transmitted. RTSE provides the service of reliably transfering arbitrarily large amounts of data from one application entity to another. The RTSE service has three phases: association establishment, data transfer, and association release. Data transfer may take more than one transfer, depending on size. When the transfer is completed and confirmed, the requesting entity is given an acknowledgement that the transmitted object has been secured by the RTSE on the accepting side. If the transfer fails, the requesting application entity is notified and appropriate corrective actions are taken by the protocol. The RTSE is defined by ISO 9066.

Remote Operations Service Element (ROSE) is a superset of many conventional Remote Procedure Call (RPC) facilities. ROSE is used to manage the request/reply interactions for an application entity. When an application entity requests an operation, it is said to be "invoking" or "initiating" the operation. Similarly, an application entity that receives the request is called the "performer" or "responder". The ROSE has two phases: binding an association and invoking operations. The ROSE is defined by ISO 9072.

Application services, such as the Message Handling Service (MHS), utilize the ACSE to open and close the association with the remote peer entity; utilize the ROSE to manage the remote request/reply to transfer the file; and utilize the RTSE to provide the reliable transfer of information. The prototype architecture for DIS developed in this thesis will utilize only the ACSE and the ROSE. This is due to the nature of simulator PDU traffic. That is, the transmission of DIS PDUs does not necessarily require reliability. In this experiment, a connection will be established with the remote system using the ACSE and the PDU data transmitted using the ROSE. However, the actual DIS implementation might utilize the RTSE for bringing new members on-line to an exercise by reliably transferring the battle history data required to update a simulation to the current state of the exercise.

ISO Development Environment (ISODE)

The feasibility of using OSI network protocols to provide network communication services for the DIS application is currently under investigation as part of the development of the DIS standard. One problem which surrounds this effort is the lack of a full seven layer OSI protocol stack implementation with which to experiment. A partial implementation of an OSI stack has been developed and is currently being used internationally to study the upper four layers of the OSI stack. This "quasi-OSI" software application is called the ISO Development Environment (ISODE).

ISODE is a non-proprietary software implementation of the upper layers (Application, Presentation, and Session) defined by ISO [ROSE90b]. This software runs on UNIX-based workstations and utilizes the DoD TCP/IP protocol suite (layers 4 and 3 protocols, respectively) to provide inter-workstation communication over Ethernet. The TCP/IP protocol suite is mature and well tested. It is used by a large number of U.S. computer manufacturers. Consequently, application developers are using the TCP/IP protocol suite to study OSI-based protocols in the upper layers while avoiding the development of the less defined OSI lower layer infrastructure. The tools and services of the ISODE are implemented through a set of software routines, libraries, and databases written in the C programming language. The services and protocols represented in the ISODE are shown in Figure 3.

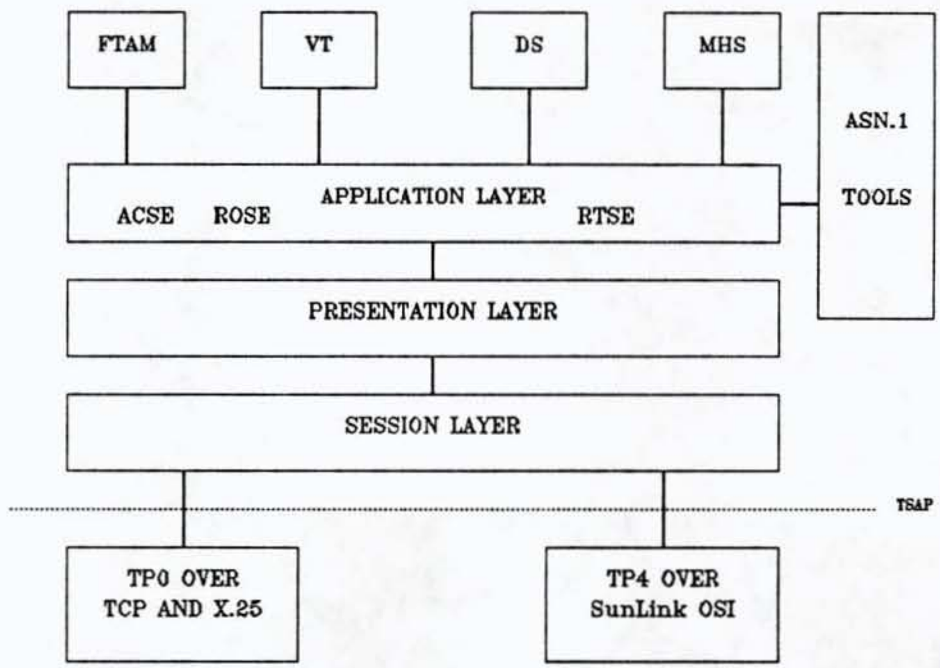

Figure 3. ISODE Model of OSI Protocols

As mentioned earlier, the ISODE uses the TCP/IP protocol suite for Network and Transport services. ISODE supports a Transport service class 0 (TP0) interface for the TCP and X.25, and a TP4 interface for SunLink OSI. SunLink OSI is a proprietary product developed by the SUN Computer Corporation.

ISODE implements the elements of the OSI upper layer infrastructure in the following way [ROSE90b]. First, the raw facilities available to applications are modeled. These include the ACSE, ROSE, RTSE, and the abstract syntax and transfer mechanisms from ASN.1. The services upon which the application facilities are built are also described. These include the Presentation service (including the PSAP), the Session service (including the SSAP), and the Transport Service Access Point (TSAP). Also modeled in the ISODE Application Layer are the Application Layer protocols defined by ISO. ISODE currently includes FTAM, FTAM/File Transfer Protocol (FTP) gateway, DS, and VT (basic class, TELNET profile). Modules planned for future ISODE release include: OSI MHS and MHS/Simple Mail Transfer Protocol (SMTP) gateway. ISODE is also aligned with the U.S. GOSIP in its mapping strategy for service definitions. Mapping is a process handled by the Directory Services (DS). This process associates the distinguished name of the application entity with its presentation address. When the presentation address is given to the service element in the application layer, a connection can be established [ROSE90a].

Computer Network Performance

The performance of a computer communications network is a crucial factor in determining the feasibility of executing specific applications using its communication services. Different applications require different levels of network performance. Applications such as file transfer and electronic mail require reliability and interoperability. While other applications, such as DIS, require not only reliability and interoperability, but also high-speed, real-time communication services in order to simulate a real world environment. Therefore, the performance metrics used to evaluate an application should be tailored to satisfy the applications requirements.

Network performance consists of two elements: 1) the network hardware performance and 2) the communication protocol architecture (software) performance.

Network hardware performance is a function of each computing device connected to the network. Once the network hardware is configured, it is a fixed factor which influences the final performance results. Although the hardware performance is fixed, it can be modified to increase efficiency by replacing lower performance devices with higher ones (i.e., a 10Mbps medium can be replaced by a 50Mbps medium). Communication protocol architecture is the second major component which determines overall network performance. The communication protocol architecture establishes the dialogue procedures between two or more machines. It provides the functionalities such as error recovery, flow control, packet routing, synchronization, and serialization.

It is difficult to measure the communication protocol performance apart from the network hardware influence since the performance of the communication protocols are dependent on the speed, implementation, and reliability of the hardware platform. Therefore, it is necessary to identify the performance associated with the hardware platforms and establish a common environment for all measurements. If this environment can be established, measuring network performance can be viewed as measuring the performance of the communication protocol architecture.

Network performance can be expressed in the following terms: **reliability**, which includes error checking, data loss, security, and recovery; and **speed**, which includes throughput, latency, and idle time. An analysis of the reliability of the ISODE communication architecture is beyond the scope of this experiment. However, this research focuses on evaluating the ISODE architecture performance in terms of speed and effective transmission capability. In this context, performance will measure how efficiently the ISODE protocols handle the PDU traffic and the possible errors that may occur during the communication process.

CHAPTER II

APPROACH

Methodology

The objective of this thesis is to gain insight into the details of implementing an OSI-based network for the DIS application. ISODE tools and facilities were used in to develop the prototype DIS architecture. Experiments were conducted and performance data were gathered and analyzed using both ISODE and UNIX facilities. This experiment consisted of four major steps:

1) Encoding PDUs in the ASN.1 language;
2) Building the Distributed (DIS) Prototype Application using ISODE;
3) Sending the DIS PDUs between workstations via the ISODE stack; and,
4) Collecting and analyzing time domain data relating to the application-to-application transfer characteristics of the prototype DIS implementation.

Hardware Environment

The hardware setup for the experiments consisted of two Sun-4 SPARC (Scalable Processor ARChitecture) workstations and two Motorola VME 1147 workstations, all interconnected via ETHERNET. The Sun workstation is a high-performance, bit-mapped workstation, utilizing a Reduced Instruction Set Computer (RISC) architecture

CPU. The Motorola is a single board UNIX system designed specifically for real-time, multi-processing configurations. This network is represented in Figure 4.

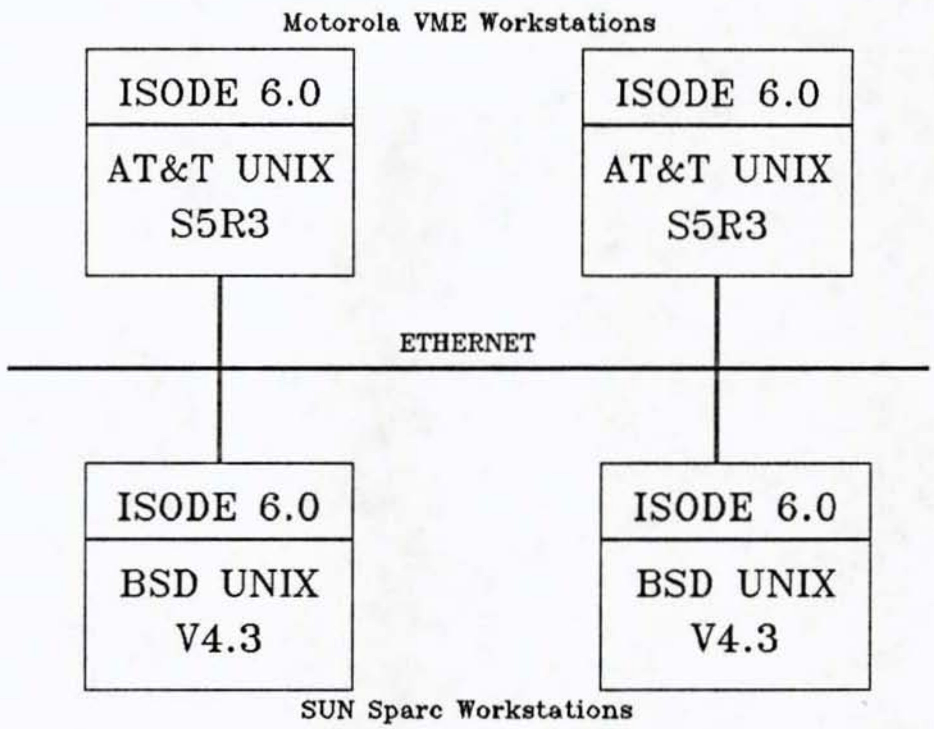

Figure 4. Experimental Network Hardware Configuration

Software Environment

All workstations in the experiment used the UNIX operating system. The Sun SPARC stations used the SunOS operating system which is an enhanced version of the 4.2 BSD and 4.3 BSD UNIX system derived from the University of California at Berkeley. The Motorola workstations used AT&T, System V.3 UNIX. UNIX commands and scripts were used in the experiments for program execution and data logging. Scripts are a collection of UNIX commands which perform a specific function.

All software for this project was written in the C programming language. The programs involving transfer of PDUs from Application Layer to Application Layer across the network utilized the ISODE libraries and databases. These libraries are collections of C source programs which can be modified by the user for specific applications. Software listings for the programs created in this experiment are included in Appendix B.

Test Plan

A set of sending and receiving (initiator/responder) programs called **SEND** was designed using the ISODE remote operations utilities. These programs provided the capability to establish an initial connection with a remote application; transmitted generic PDUs across the network from initiating process to responder process; received the same PDU reflected back from the responder process; and released the connection. The SEND programs also had a built-in clock that counted the time starting the moment after connection establishment and ending immediately before the connection was released. Therefore, the clock registered the time elapsed to send and receive the echo of the generic PDU, giving an indication of the speed of the protocol stack running on each workstation. With this measurement scheme, the connection establishment and release times were not included. What remained was the Round Trip Time (RTT) of the PDU.

The RTT was the elapsed time for a single PDU to perform a round trip between two computers. The round trip path is shown in Figure 5. It included sending the PDU down the ISODE stack, across the TCP/IP, onto the Ethernet where it was transmitted to the target host (or responder). Once the PDU arrived at the responder, it passed through the TCP/IP, through the ISODE, finally reaching the target remote operations application. To complete the round trip, the PDU passed through the ISODE, through the TCP/IP, across the ETHERNET, up the initiator's TCP/IP and ISODE, where it arrived at the host remote operations application.

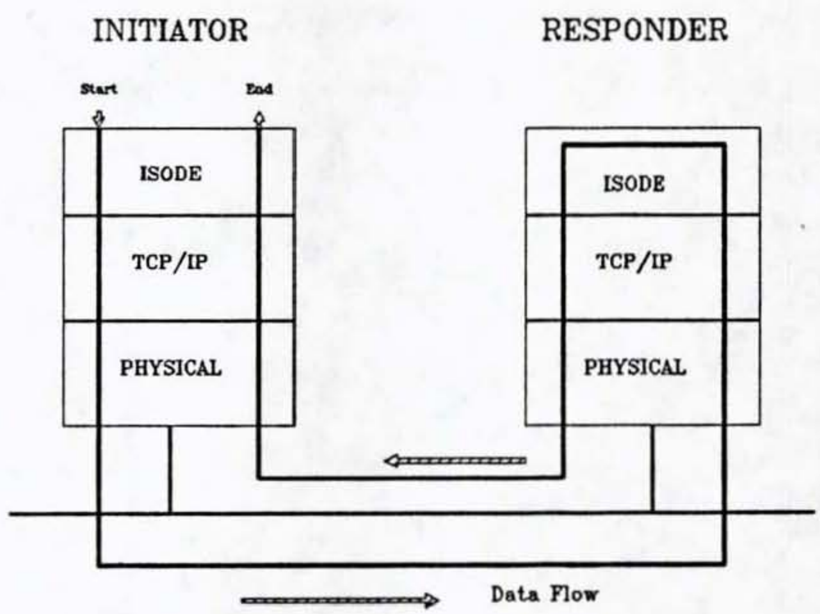

Figure 5. PDU Round Trip Path

Research of Problem

Summary of Relevant Research

An extensive literature search was conducted on the following topics: real-time simulation and simulation architectures; OSI protocols; and communication network protocol performance. As a result of this survey, one project was identified as being relevant to this research thesis. The paper, **SAFENET - A Navy Approach to Computer Programming** [PAIG90], describes the development of a military real-time computer architecture using OSI standards. A description of the SAFENET project follows.

The **Survivable Adaptable Fiber Optic Embedded Network (SAFENET)** program is an effort by the U.S. Navy to develop standard computer network profiles which meet the requirements of Navy shipboard mission critical computer systems. There are two SAFENET standards: SAFENET I and SAFENET II. SAFENET I is based on the IEEE 802.5 LAN standard (Token Ring), while SAFENET II is based on FDDI (Fiber Distributed Data Interface, ISO 9314). Each standard describes a network profile which covers the full seven layer ISO model. Each SAFENET standard can be implemented as any of three protocol suites: OSI, lightweight, or the combination of the two. The OSI suite is intended to provide fully ISO compliant networking, while the lightweight suite is intended to support systems with real-time communication requirements. The SAFENET physical topology is based on a dual counter-rotating ring architecture which provides many survivablity features [PAIG90].

CHAPTER III

IMPLEMENTATION

Encoding Protocol Data Units in ASN.1

An abstract data type is a concept for describing a data structure in a machine-independent manner. Although a data structure may have a concrete representation on a given system (i.e., a "struct" in the C language), its corresponding abstract data type is defined in a implementation-independent manner called the **abstract syntax.** There is also a well defined set of rules associated with a data structure. These rules are termed the **abstract transfer notation.** The abstract transfer notation serializes (i.e., converts to a bit stream) the abstract syntax and generates a data stream corresponding to the abstract data type for transmission on the network [ROSE88]. This process is termed **encoding** of data structures. When the data are received at the destination, the process is executed in reverse order. This process is termed **decoding.** This mapping process is shown in Figure 6.

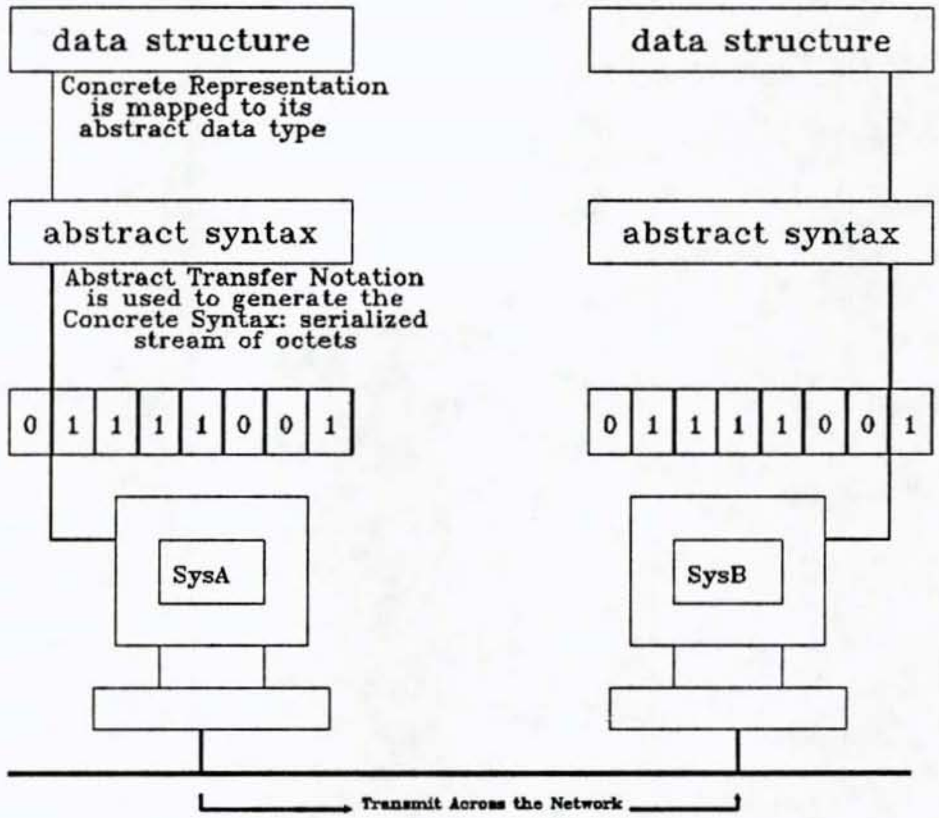

Figure 6. ASN.1 Mappings

ASN.1 descriptions consist of several tokens or expressions. These expressions can take the form of one of the following: **words,** which consist of upper- and lower-case letters, digits, and hyphens; **numbers,** which consist of digits; strings, which are either character, hexadecimal, or binary; and **punctuation**. A collection of ASN.1 descriptions is termed a module [ISO882]. The high-level syntax of a module is as follows:

```
<module>  DEFINITIONS ::=
BEGIN

<linkage>

<declarations>

END
```

The <**module**> term names the module. For our purposes, the module name will be the name of the DIS PDU. For example the DIS Entity State PDU will have a name of EntityState; similarly, the DIS Update Threshhold Request PDU would have a name such as UpdateThreshRequest.

The <**linkage**> term links this module with other modules. Within this section of the module, any other modules which should be imported or exported will be identified. For the DIS application, the DIS PDU header or descriptor file, which is common to all DIS PDUs, can be defined once, and then imported into all DIS PDU modules.

The <**declarations**> term contains the actual ASN.1 definitions. Three kinds of objects are defined using ASN.1: types, values, and macros. Each object is named using an ASN.1 word, for which alphabetic case convention is important. For a type, the word starts with an uppercase letter; a value word starts with a lowercase letter; and a macro consists of entirely uppercase letters.

An ASN.1 type is defined in the following manner:

```
NameOfType ::=

        TYPE
```

NameOfType would represent a data field within a DIS PDU and **TYPE** would describe its declaration, such as INTEGER. The ASN.1 notation defines a collection of types to be used in the module declaration. The following types are available: simple, object, constructor, tagged, and meta. DIS PDUs will primarily use the simple and constructor types. A brief description of each type follows.

Simple types are viewed as the primitive data elements. ASN.1 defines the following simple types:

> BOOLEAN - data type taking one of two distinguished values True or False
>
> INTEGER - data type taking a cardinal number as its value
>
> ENUMERATED - represents the complete set of values that a data type is allowed to assume
>
> REAL - data type taking a real number as its value
>
> BIT STRING - data type taking zero or more bits as its value
>
> OCTET STRING - data type taking one or more octets as its value
>
> NULL - data type that is a place-holder

Two of the simple types turned out to have complicated semantics. Consequently, a separate type called **object types** was created to handle them. These **object types** are as follows:

> OBJECT IDENTIFIER - data type denoting an authoritatively named object; provide a means of describing an object regardless of the semantics associated with it

OBJECT DESCRIPTOR - denotes a textual string that also references an object

Simple types can be combined to build complex data types within the <declaration> of a module. These types are called **constructor types.**

SEQUENCE - data type denoting an ordered list of zero or more elements

SEQUENCE OF - data type denoting an ordered list of zero or more elements of the same ASN.1 type

SET - data type denoting an unordered list of zero or more members

SET OF - data type denoting an unordered list of zero or more members, each member having the same ASN.1 type

Tagged types provide a method for distinguishing unique occurrences of the same ASN.1 data types. There are four different classes of tags:

UNIVERSAL - provides a global identification of the well-known data types discussed thus far

APPLICATION-WIDE - provides identification within a given ASN.1 module

CONTEXT-SPECIFIC - provides identification unique to a constructor type

PRIVATE-USE - provides a unique identification within a given project-specific agreement

The last type, **meta**, transcends both simple and constructor types.

CHOICE - data type that is defined as the union of one or more data types

ANY - data type that is the union of all possible data types

EXTERNAL - data type that is defined by some document outside the current module

SUBTYPES - a refinement of some "parent" ASN.1 type

ASN.1 also specifies syntax for describing **value** objects. Value notation produces human-readable descriptions of the ASN.1 transfer syntax. The last object defined by ASN.1 are **macros.** The macro facilities are used to capture additional semantic information. This is accomplished by ASN.1 macro notation, which literally rewrites the grammar rules of ASN.1.

The components of an ASN.1 module are diagramed in Figure 7.

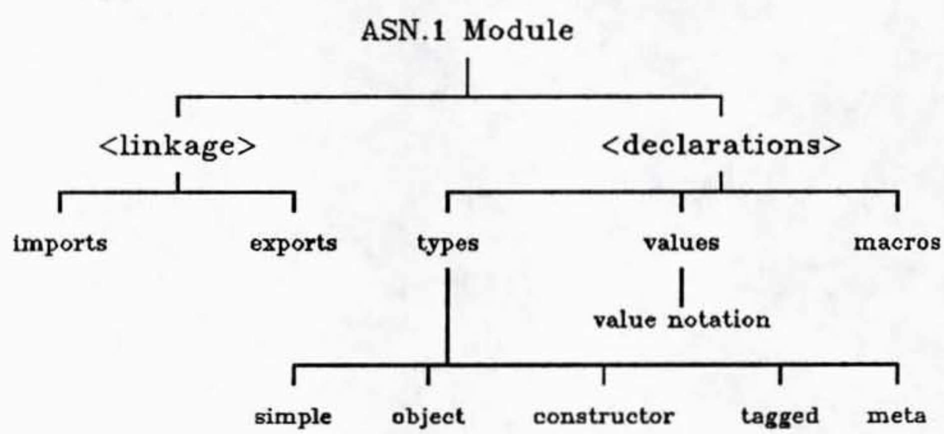

Figure 7. ASN.1 Module Components

Encoding DIS PDUs

To explore the application of ASN.1 to DIS PDUs, the DIS Entity State PDU will be examined. The bit layout for this PDU [MCDO91] is shown in Table 2. In the DIS

TABLE 2

DIS ENTITY STATE PDU

FIELD SIZE (octets)	ENTITY STATE	PDU FIELDS	FIELD TYPE
6	Entity ID #	Site ID Host ID Entity ID	16 bit Integer 16 bit Integer 16 bit Integer
2	PADDING	Unused	16 bit
8	Entity Type	Entity Kind Domain Country Category Subcategory Specific Extra	8 bit enumeration 8 bit enumeration 16 bit enumeration 8 bit enumeration 8 bit enumeration 8 bit enumeration 8 bit enumeration
4	Timestamp		32 bit Integer
24	Entity Location	X - component Y - component Z - component	64 bit Floating Point 64 bit Floating Point 64 bit Floating Point
12	Linear Velocity	X - component Y - component Z - component	32 bit Floating Point 32 bit Floating Point 32 bit Floating Point
12	Linear Acceleration	X - component Y - component Z - component	32 bit Floating Point 32 bit Floating Point 32 bit Floating Point
12	Entity Orientation	Psi Euler Theta Angles Phi	32 bit Angle 32 bit Angle 32 bit Angle
12	Angular Velocity	X - component Y - component Z - component	32 bit Integer 32 bit Integer 32 bit Integer
8	Dead Reckoning Parameters	TBD	64 bits
4	Entity Appearance		21 bit Integer
12	Entity Marking		11 bit Character Set
4	Capabilities		32 bit Boolean
3	PADDING	Unused	
1	# of Articulated Parts		8 bit Integer
Varies	Articulated Parts		

Entity State PDU ASN.1 module, all fields represented in the PDU must be defined. The following is an example ASN.1 encoding definition:

```
EntityStatePDU ::=
        SEQUENCE{
                EntityId,
                Padding1,
                EntityType,
                TimeStamp,
                EntityLocation,
                EntityLinearVelocity,
                EntityLinearAcceleration,
                EntityOrientation,
                EntityAngularVelocity,
                DeadReckoningParameters,
                EntityAppearance,
                EntityMarking,
                Capabilities,
                Padding2,
                NoArticulatedParts,
                ArticulatedParts
    }
```

The first field in the Entity State PDU is the Entity ID #. This field is composed of three subfields, Site ID, Host ID, and Entity ID. Each subfield is represented by a 16 bit integer. In an ASN.1 module, the Entity ID # field would be coded as follows:

```
EntityId ::=
        SEQUENCE{
                SiteId
                        INTEGER,
                HostId
                        INTEGER,
                EntityId
                        INTEGER
        }
```

Where the subfields are represented in a **SEQUENCE** (ASN.1 constructor type) and identified as the ASN.1 simple type **INTEGER**.

The next field in the Entity State PDU is the PADDING field. Since this field is unused, it can be represented by the simple type **NULL**.

```
Padding ::=
        NULL
```

The Entity Type field is comprised of seven subfields: Entity Kind, Domain, Country, Category, Subcategory, Specific, and Extra. Each subfield is represented by an 8 bit or 16 bit enumeration. This field would be coded in an ASN.1 module as follows:

```
EntityType ::=
        SEQUENCE{
                EntityKind
                        ENUMERATED,
                Domain
                        ENUMERATED,
                Country
                        ENUMERATED,
                Category
                        ENUMERATED,
                Subcategory
                        ENUMERATED,
                Specific
                        ENUMERATED,
                Extra
                        ENUMERATED
        }
```

```
EntityKind ::=
        ENUMERATED{
                Other (0),
                Platform (1),
                Munition (2),
                LifeForm (3),
                Environmental (4),
                CulturalFeature (5)
        }

Domain ::=                      -- for Platform, Life Form, Environmental,
        ENUMERATED{             -- and Cultural Features
                Other (0),
                Land (1),
                Air (2),
                Surface (3),
                Subsurface (4),
                Space (5)
        }

Domain ::=                      -- for Munition
        ENUMERATED{
                Other (0),
                AntiAir (1),
                AntiArmor (2),
                AntiGuidedMunition (3),
                AntiRadar (4),
                AntiSatellite (5),
                AntiShip (6),
                AntiSubmarine (7),
                BattlefieldSupport (8),
                Strategic (9),
                Miscellaneous (10)
        }
```

```
Country ::=
    ENUMERATED{
        Other (0),
        Afghanistan (1),
        Albania (2),
        ....
        Zambia (179),
        Zimbabwe (180),
        PalestineLiberationOrganization (181),
        Neutral (200)
    }
```

The Country subfield has not been fully described here due to the size of the enumeration (i.e., approximately 200 countries). Similarly, the Category, Subcategory, Specific, and Extra subfields vary depending on the Entity Type specified in the Entity Kind subfield. Therefore, these fields are also not elaborated but would follow the same format which has been identified.

As seen in the above example for the Entity Type field, once a subfield is initially identified (as with the SEQUENCE type), the enumeration can be further detailed by defining the subfield as a separate entity (i.e., EntityKind ::= ENUMERATED{ }).

Other fields in the Entity State PDU, such as Entity Location, Linear Velocity, and Linear Acceleration would be defined in an ASN.1 module as a SEQUENCE of REAL types. These **REAL** subfields would then be further defined by identifing the mantissa, base, and exponent for each. Additionally, the Entity Marking field would be identified as a simple **OCTET STRING** and the capabilities field would be defined as a simple **BOOLEAN** type.

Experimental Constraints

Since the DIS PDU standard has only recently been released, there are very few, if any, simulators which generate and use the DIS PDUs. At IST, one project is underway to generate and test these PDUs [CENG91]. However, the programs being generated have not fully implemented all of the DIS PDUs. Therefore, actual DIS PDU information is not available for testing.

As stated earlier, the testing performed in this thesis measures the round trip time from connection establishment to connection release. Conceptually, the bytes transmitted across the ISODE stack can be generated by any means, e.g., a string input from the keyboard represents the same information, with respect to measuring round trip time, as an actual DIS PDU. Therefore, a significant factor in the experiment is the length of the information being transmitted across the network through the ISODE stack. The lengths of the DIS PDUs are stated in Table 3.

A typical entity represented by the Entity State PDU would be an M1 A1 main battle tank. A tank might typically portray two articulated parts (i.e., the main gun and the turret) each of which might have one articulated parameter (i.e., pitch angle for the gun and yaw angle for the turret). Depending on the vehicle modeled, the Entity State PDU could conceivably be very large (i.e., a ship may have upwards of ten articulated parts), possibly in the 200 to 250 byte range. Therefore, the lower boundary established for DIS PDU size is 16 bytes (Resupply Cancel PDU plus Header file). Similarly, the upper bound for a DIS PDU is 250 bytes (Entity State PDU plus Header file).

TABLE 3

LENGTH OF DIS PDUS

DIS PDU	Length in Bytes
Header File	4
Entity State	$124 + n(12m_i + 4)$
Weapons Fire	84
Weapons Detonation	108
Update Threshold Request	40
Update Threshold Response	16
Service Request	16 + 12n
Resupply Offer	16 + 12n
Resupply Received	16 + 12n
Resupply Cancel	12
Repair Response	16
Collision	16
Radar	$20 + r(12s_i + 44)$

n = number of articulated parts
m_i = number of articulated parameters
r = number of radar systems
s_i = number of entities illuminated

The generic DIS Entity State PDUs used in this experiment were input from a user-generated file **EXEC**. This file sends varying length strings (1 byte to 440 bytes) which are described in the ASN.1 module by a SEQUENCE of **IMPLICIT GraphicString**, which is a **Universal Tagged** type of 25 (See [ISO8824].). By using IMPLICIT, only the tag associated with **GraphicString** will be transmitted on the network. The ASN.1 module developed for the generic DIS PDU prototype architecture (**PDU-ASN.PY**) is located in Appendix C.

Building the Distributed Interactive Simulation Prototype Application in ISODE

In OSI, remote operations are viewed as an integral part of the methodology for building distributed applications. A Remote Operation (RO) is a request/reply operation between two Application Processes (AP), located either in a local or distributed environment. In a RO, an operation is invoked by an AP. In response, its peer returns an outcome of the operation. The most basic application consists of an **initiator**, which requests the service or desired operation, and a **responder**, which provides the service or operation. The concept of the initiator/responder distributed application is shown in Figure 8.

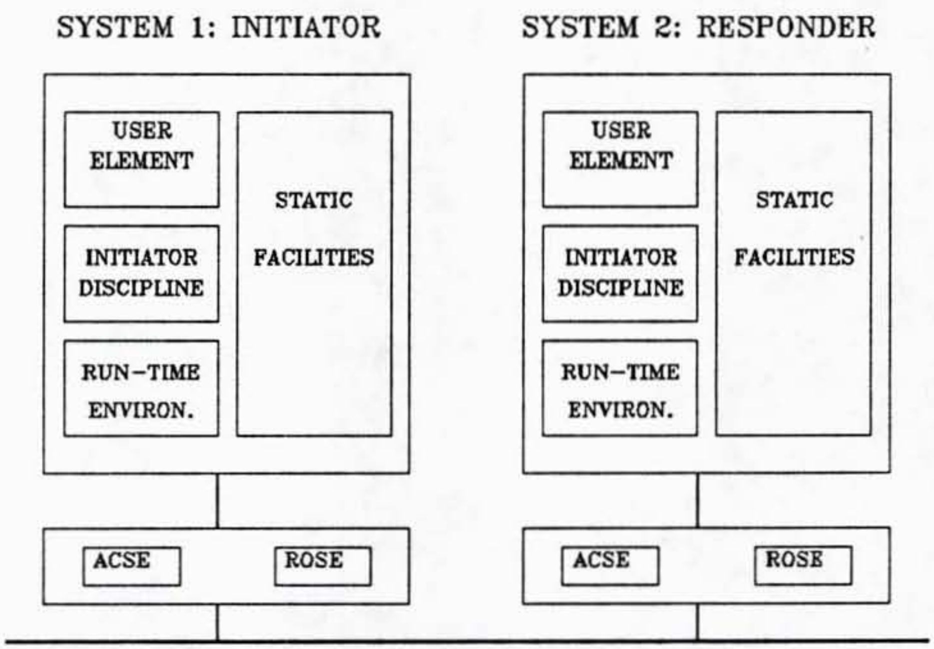

Figure 8. Dynamic and Static Facilities of Distributed Applications

The initiator and responder both have static and dynamic facilities [ROSE90a]. The RO-specification, which is the formal definition of the remote operation, is the static facility. This includes the ASN.1 data structures which enumerate the complete set of operations, errors, and abstract data types which are used by the system. The dynamic facilities include the following: the initiator and responder disciplines, which initiate or respond to the service or desired operation; the ACSE, which establishes and terminates the associations used by the entity; the ROSE, which manages the request/reply interactions; the DSE, which maps the service required by the system onto the entities available on the network; and finally, the run-time environment, which maps the C structures, using ASN.1 compilers, into the corresponding abstract syntax which is then presented this to the RO service for delivery.

The Remote Operation (RO) facilities within OSI are designed to provide the mechanisms for building diverse applications such as message handling, directory services, and remote database access. Using RO, the generic PDUs can be transmitted between remote simulators, or as is the case for this experiment, between remote workstations.

There are four distinct steps to building a distributed application in ISODE: defining the naming and addressing information in the ISODE databases; building and compiling the remote operations module; compiling the abstract syntax module; and building the initiator and responder programs. The following sections describe how these steps were accomplished for the DIS prototype.

Naming and Addressing Information

In the OSI reference model, naming and addressing information is the function of the Directory Services Element (DSE) of the DS protocol. At a minimum, the DSE determines the presentation address of each application participating in a binding. A binding provides the mechanisms for establishing an association between two application entities or processes. This binding process is accomplished in the following way. The identity of an application entity is its distinguished name in the OSI Directory, which is an authoritative description of the AE. The application contacts the OSI Directory, presents the distinguished name of the AE with which it is interested in communicating, and asks for the presentation address attribute associated with that name. A presentation address consists of a presentation selector, a session selector, and a transport address. A transport address consists of a transport selector and one or more network addresses. The presentation address is given to the service element in the Application Layer to establish a connection. This address is passed to the presentation service, which uses the presentation selector. The remainder is given to the session service, which uses the session selector. The ultimate remainder is given to the transport service. The transport service looks at each network address and decides which mode of network service (connection-oriented or connectionless) will be used for the address. Based on the derived network service, the communications quality of service desired by the application, the transport service selects a transport protocol. The network addresses are then ordered by preference, and for each network address, the transport service starts the appropriate transport protocol and the underlying network service is invoked. [ROSE90a]

In ISODE, the naming and addressing information is managed through the use of databases. There are three databases used for this function [ROSE90b]:

isobjects: which maintains the mappings between object descriptors (OD) and object identifiers (OID) (ODs and OIDs were described in the ASN.1 object types section);

isoentities: which manipulates the mappings between application-entity information and presentation addresses; and,

isoservices: which maintains the mappings between textual descriptions of services, service selectors, and local programs.

The application services which need to be defined in the isobjects, isoentities, and isoservices databases are the following:

abstract syntax: This describes the data structures being exchanged by the service.

application context name: This describes the protocol being used by the service.

application-entity information: This uniquely names an entity in the network.

presentation address: This locates an entity in the network.

local program: this identifies the program on the local system which implements the service.

First, the **abstract syntax** presentation context information (PCI) of the service, along with the **application context** need to be identified. This is performed in the **isobjects** database through the use of object identifiers (OID). The object identifier tree 1.17.1 was used for defining local services in ISODE. Therefore, the new service will be assigned the number 1.17.1.n, where n is the lowest unassigned number in the tree. The **isobjects** database will contain the abstract syntax PCI as the first notation in the

1.17.1.n subtree, and the application context as the second. The entry appears as follows:

"isode send string demo pci"	1.17.1.13.1
"isode send string demo"	1.17.1.13.2

Next, the template for the **application-entity information** and **presentation address** should be defined. These are outgoing connections and are defined in the **isoentities** database. The application-entity information is currently an object identifier, from the 1.17.4.1 subtree in ISODE. The presentation address is composed of a presentation selector, a session selector, a transport selector, and a set of network addresses. Within ISODE, this is implemented by using an empty (i.e., no information) presentation and session selector, a unique transport selector, and a simple default template for the network address (This is filled in during connection establishment.). The "empty presentation and session selector" is an implementation decision made by the ISODE authors. The **isoentities** database will be edited to appear as follows:

default sendstring	1.17.4.1.8 #1041/

Finally, the program on the local system which implements the desired service should be defined. This definition is used for incoming connections and is performed in the **isoservices** database. The **local program** in this experiment that transmits DIS Entity State PDUs across the network is the **SEND** program. Therefore, SEND will need to be defined in **isoservices**. The strategy used for allocating the presentation addresses above necessitates a mapping only between the transport selector and the **SEND** program. Therefore, the entry in the **isoservices** database appears as:

"tsap/isode sendstring"   #1041 ros.send

The **isobjects, isoentities**, and **isoservices** databases developed for this experiment are included in Appendix B.

Remote Operations Module

The RO module defines the operations, errors, and abstract syntax of the data structures to be exchanged by the service. The operation performed in this experiment consisted of sending the generic PDU to the remote host (or workstation) and reflecting back the PDU to the initiating process (or workstation). The error defined in the RO module alerted the user if congestion occured during the transmission, preventing the sending of the PDUs. The last definition included in the RO module was the abstract syntax of the generic PDU. This was defined in a previous section and can be found in Appendix C. The RO module, **SEND.RY**, can also be found in Appendix C.

The ISODE program ROSY (Remote Operations Stub-generator (YACC-based)) reads the ASN.1 module that uses the RO-notation and generates the following: a set of remote operation definitions with associated data types (**PDU-OPS.C**); a set of error definitions with associated data types (**PDU-OPS.C**); and, a set of C stubs and definitions that are either invoked or called to request an operation by the performer (**PDU-STUB.C**)[ROSE90]. (YACC is a UNIX operating system facility which generates code that interprets the syntax rules of the language, e.g., a compiler-compiler). The software routines produced by the ROSY compiler are included in Appendix C.

Abstract Syntax Module

An abstract syntax module defines the data structures being exchanged by the service, as defined earlier in this chapter. ISODE provides two compilers for encoding this module. The POSY (PEPY Optional Structure-generator (YACC-based)) program reads an ASN.1 module and produces the following: the corresponding C structures definitions and an augmented ASN.1 module. This augmented ASN.1 module relates the C structures (**PDU-TYPE.C**) to their ASN.1 counterparts (**PDU-TYPE.PH**). The augmented module is also read by the PEPY (Presentation Element Parser (YACC-based)) compiler. The PEPY compiler generates and interprets ASN.1 encodings (**SEND.C**). Using the augmented ASN.1 module, PEPY can produce C code fragments that map between the C structures and the augmented ASN.1 [ROSE90a]. The C structures produced by the POSY program are mapped by the ISODE run-time environment to the abstract syntax and then used for mapping the abstract syntax to the machine specific concrete syntax. The encoding process described above enables the invoker and performer to deal only with the native machine C structures. This creates an open systems interface which is entirely automatic [ROSE90a]. The software structures produced by the POSY and PEPY compilers are included in Appendix C. This entire compilation process is depicted in Figure 9.

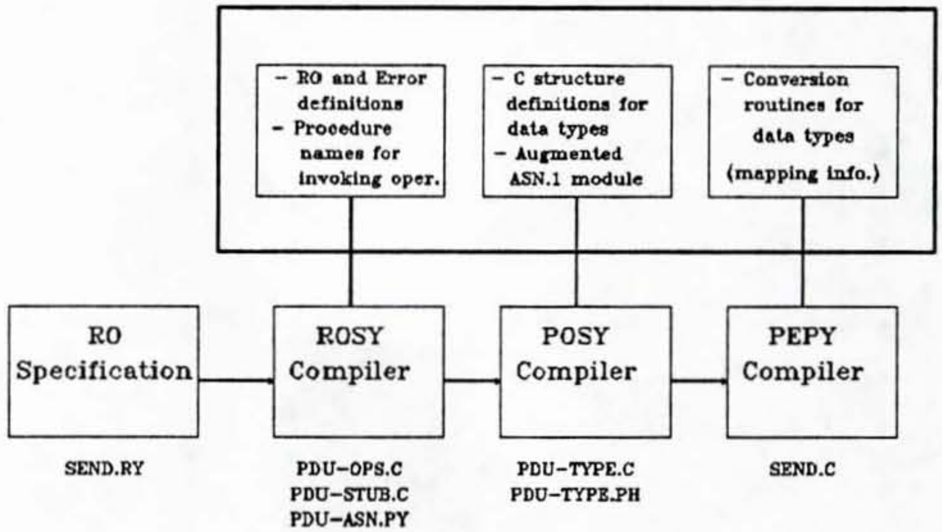

Figure 9. ISODE Static Facilities

Initiator and Responder Programs

The initiator and responder disciplines for the SEND program were developed for this experiment from the existing ISODE libraries and programs. By adding C code to the ISODE utilities, the user is able to design the operation of the desired application. A sample of the available utilities include: event handlers, routines to set underlying services, and routines to poll network activity. For more detail on ISODE routines and libraries, consult [ROSE90b] or [LOPE91].

An initiator is responsible for four operations: association establishment, operation invocation, association release, and error handling. There are two forms of initiators [ROSE90b]:

interactive: The user runs a program and interactively directs the invocation of operations; and,

embedded: As part of its running, the program automatically forms an association and invokes operations as required.

For the purposes of this experiment, the **interactive** initiator was implemented. This allowed the user to direct the transmission of PDUs and therefore to control the operation of data transmission.

The responder is responsible for three functions: association management, operation response, and error handling. A responder may also take on one of two forms [ROSE90b]:

single association: Each time the service is requested, a new instantiation of the program implementing the service is executed (a dynamic approach); and,

multiple association: Each time the service is requested, the request is given to a single, already excuting, instantiation of the program which implements the service (a static approach).

For the purpose of this experiment, the **multiple association** responder was implemented. This allowed the user to examine the performance in a static environment.

The SEND initiator and responder programs are included in Appendix B.

Prototype DIS Communication Architecture Execution

This experiment was conducted on three of the four computers connected to the ETHERNET network. The fourth workstation was not used due to software problems. The workstations used in the experiment included the two SUN Sparc stations (Falcon and Ibis) and one Motorola VME workstation (Heron). The experiment took the following factors into account: network load, processing capability of each workstation, and the size of the message to be transmitted.

The experiment was conducted in two parts. The first part of the experiment allowed the SUN workstation **Ibis** to communicate with the Motorola workstation **Heron** using the ISODE communication architecture and the **SEND** programs. The second experiment consisted of both SUN workstation, **Falcon** and **Ibis**, transmitting generic PDUs. In both experiments, the computer's CPUs were dedicated to performing this communication task, which excluded time for tasks outside this experiment. During the experiments, activity on the network was limited to the generic PDUs being transmitted. The network was used essentially as a point-to-point link [SHEN91].

The **SEND** program used in this experiment could be invoked in one of two ways. First, by entering the **SEND** command from the keyboard, an interactive loop would be entered which would allow the sending of more than one generic DIS PDUs. The second method, file driven, was chosen for this experiment. The **SEND** program was invoked from the program **LOOP**, which subsequently invoked the **SEND** program several times, each time transmitting a different length PDU (i.e., 1, 10, 100, 200, and 440 bytes). By transmitting varying length PDUs, the varying lengths of the DIS PDUs could be modeled. For each length PDU, the **SEND** process was repeated sending a varying number of the PDUs (i.e., 1, 10, 100, 500, and 1000). This was to ensure that the data collected would be unbiased and robust for statistical analysis. For each interation of the **SEND** process, statistical data were gathered on the round trip time (RTT).

SEND Program: Detail Description

The **SEND** program made use of the remote operations services implemented in ISODE. This program sent a generic PDU from the initiating process across the ETHERNET to the responding process. Once at the responder, the PDU was reflected back to the initiating host. If the connection was not established, an error message was returned and displayed. A diagram depicting the SEND process is shown in Figure 10.

The SEND process was accomplished through an interactive initiator, which managed the association and invoked the operation, and a static responder, which performed the operation. The initiator performed four operations: association establishment, operation invocation, association release, and error handling. During association establishment, the application-entity information and presentation address for the desired service were computed, along with the application context (ACN) and default presentation context information (PCI) for the service. Also, a session reference identifier was chosen. This was done in the **ryinitiator** routine using the ISODE **AcAssocRequest** routine. At this time, the **tsap** deamon was contacted to invoke the responder. The **tsap** deamon is a process that runs in the background of the UNIX operating system which handles all incoming connections for ISODE. The tsap process

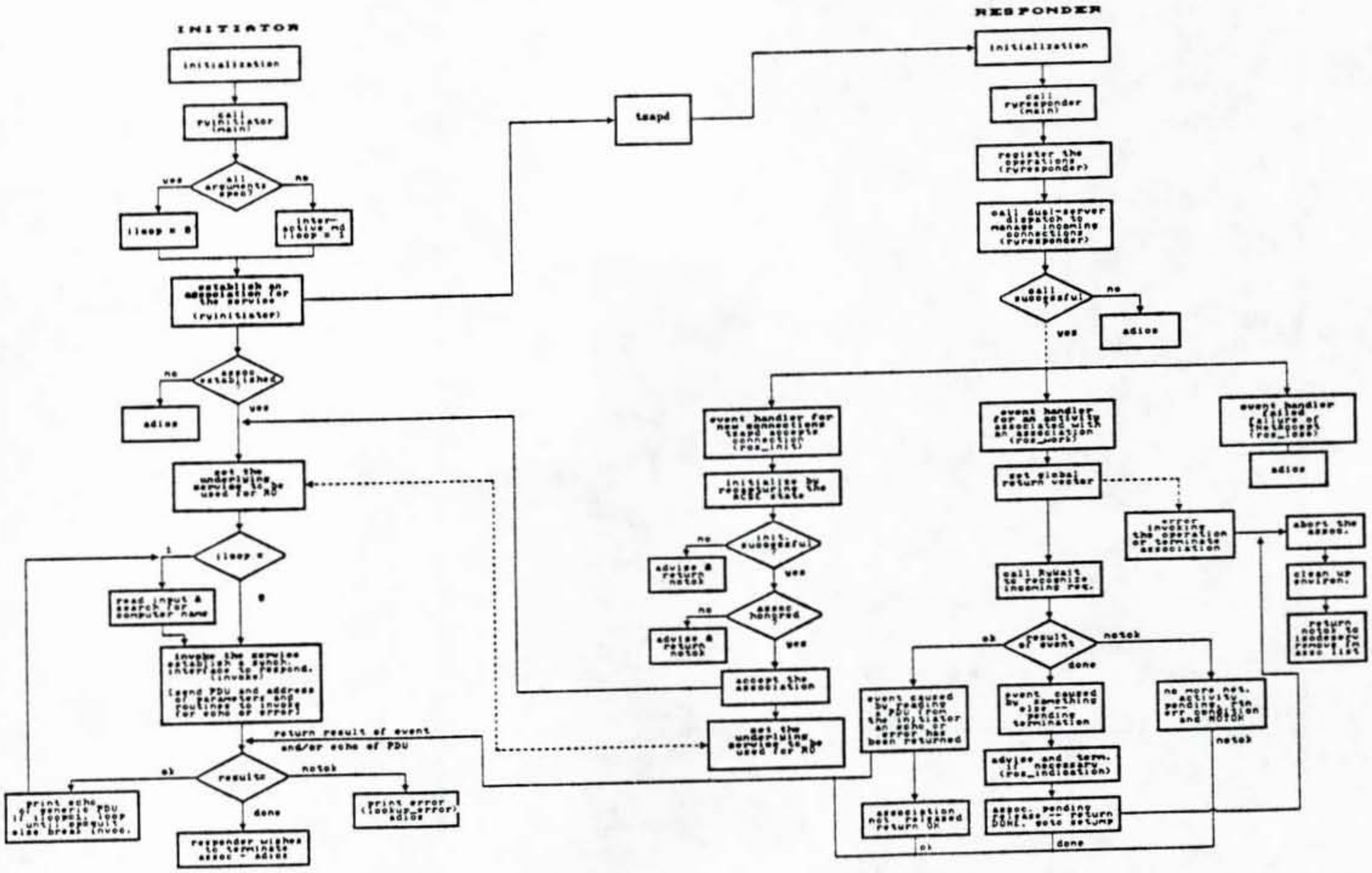

Figure 10. Flow Diagram of the SEND Program

provides the communication for the remote systems using the ISODE connection services. If an association was established with the responder, the underlying service to be used for the remote operation (the presentation service) was set using routine **RoSetService**.

At this time, the interactive loop was entered. A line was read from the input and a search was performed to determine which computers the SEND process would execute on. The invocation was performed through a synchronous interface implemented in routine **RyStub**. The invoked operation returned one of three results: error, done, or the echo of the SEND operation. The result of the operation, the echoed DIS PDU, was displayed on the screen, and the association was released. Since the user was in an interactive loop, another PDU was then transmitted. The association was released when the **quit** operation was invoked.

Any time an error was encountered, an **adios** or **advise** routine reported the error and terminated the association if appropriate.

The responder was responsible for three functions: association management, operation response, and error handling. Association management was implemented in routine **ryresponder**. After initializing the invoked program, the SEND operation was registered with the ISODE **RyDispatch** routine. The routine **isodeserver** was then called to set the addresses of event-handlers and to manage any associations. If the call to **isodeserver** was successful, then the program terminated immediately.

When an event associated with a new connection occured, the event-handler **ros_init** was invoked. This routine first called **AcInit** to re-capture the Association Control Service Element (ACSE)-state. If the initialization was successful, the routine

AcAssocResponse was called to deal with the incoming association from the initiator. If the association was accepted, the underlying service for remote operations was set using the **RoSetService** routine.

If any activity associated with an association occurs, the event-handler **ros_work** was invoked. This routine set a global return vector using **setjmp(3)** and then called ISODE routine **RyWait** to poll for the next operation-related event. This usually resulted in the registered operation being performed. Next, the operation, sending the PDU, was attempted. If it was successful, the result (an echo of the DIS PDU) was returned to the initiator by the **RyDsResult** routine. Otherwise, the error was returned by the **RyDsError** routine. The **RyWait** then indicated that no more network activity was pending. If extraordinary conditions existed for the association, routine **ros_indication** was called. This routine processed any errors that occurred, caused control to return to the **setjmp** call, and terminated the association.

The **isodeserver** routine used the **TNetAccept** routine to wait for the next event on existing associations and new connections. If failure occured during this operation (i.e., network listening failed), the **ros_lose** routine was advised. This routine logs the error condition and terminates the operation with one of the **adios** or **advise** routines.

Experimental Performance Analysis

As stated earlier, the network performance in this experiment is evaluated in terms of speed. In this context, speed is measured in terms of the Round Trip Time

(RTT) for a generic DIS PDU to transmit from one workstation to another, and then be transmitted back to the originating host. The latency associated with the transmission will give an indication of the network performance. Latency is the time delay between the transmission of data and the reception by the peer entity. It is related to transit delay across the network, but also includes the associated processing delays (i.e., processing encountered at each level within the ISODE architecture stack).

Performance Statistics

In order to obtain consistent and meaningful performance assessment data, the SEND program was executed multiple times while keeping all external factors constant, and storing all round trip time (RTT) measurements in a data file. UNIX shell programs were designed [SHEN91] to compute the statistics [HOGG87] as follows:

Minimum - the minimum round trip time

Maximum - the maximum round trip time

Median - the middle observation when the observations were arranged in increasing order of magnitude

Mean - the average value of the observations

Variance - a measurement of dispersion of the observations

Standard Deviation - the square root of the variance

Tables 4 and 5 present the data that was gathered [SHEN91] during the experiments.

TABLE 4

PERFORMANCE DATA FOR EXPERIMENT 1

Sample Size	PDU Bytes	Minimum Time (ms)	Maximum Time (ms)	Mean Time (ms)	Median Time (ms)	Variance (ms)	Standard Deviation (ms)
1	1	40	40	40	40	0	0
	10	30	30	30	30	0	0
	100	40	40	40	40	0	0
	200	50	50	50	50	0	0
	440	40	40	40	40	0	0
10	1	30	40	38.9	40	8.9	2.983
	10	40	40	40	40	0	0
	100	40	40	40	40	0	0
	200	40	70	47.9	50	75.7	8.701
	440	40	50	47	50	21	4.583
100	1	30	90	39.58	40	49.85	7.06
	10	30	50	39.19	40	15.36	3.919
	100	30	80	41.18	40	40.43	6.358
	200	40	90	45.54	49	46.37	6.81
	440	40	60	47.09	50	24.55	4.955
500	1	30	130	39.688	40	44.436	6.666
	10	30	90	39.424	40	32.046	5.661
	100	30	120	40.34	40	29.93	5.471
	200	40	110	45.648	50	41.274	6.424
	440	39	120	48.26	50	52.634	7.255
1000	1	30	120	39.6	40	38.406	6.197
	10	30	110	39.457	40	29.501	5.431
	100	30	130	40.728	40	34.003	5.831
	200	36	120	46.159	50	54.16	7.359
	440	40	120	48.127	50	4336.982	65.856

TABLE 5

PERFORMANCE DATA FOR EXPERIMENT 2

Sample Size	PDU Bytes	Minimum Time	Maximum Time	Mean Time	Median Time	Variance	Standard Deviation
1	1	36	36	36	36	0	0
	10	39	39	39	39	0	0
	100	37	37	37	37	0	0
	200	39	39	39	39	0	0
	440	40	40	40	40	0	0
10	1	29	37	30	29	5.6	2.366
	10	29	36	30.1	29	4.3	2.074
	100	29	38	31.1	30	6.3	2.51
	200	32	39	33.1	32	5.3	2.302
	440	33	40	34.2	34	4.2	2.049
100	1	28	37	29.55	29	1.15	1.072
	10	29	36	29.65	29	1.29	1.136
	100	29	40	30.47	30	1.85	1.36
	200	31	40	32.46	32	1.99	1.411
	440	33	40	33.56	33	1.03	1.015
500	1	22	38	29.606	29	1.52	1.233
	10	29	99	29.924	29	10.912	3.303
	100	29	114	30.784	30	22.134	4.705
	200	29	107	32.446	32	12.3	3.507
	440	33	85	33.654	33	6.332	2.516
1000	1	24	36	29.574	29	1.081	1.04
	10	26	37	29.824	30	1.332	1.154
	100	23	96	30.515	30	5.586	2.363
	200	25	39	32.46	32	1.373	1.172
	440	30	108	33.793	33	12.649	3.557

Results and Analysis

As mentioned earlier, this experiment was conducted on three of the four computers connected to the ETHERNET network. The workstations used in the experiment included the two SUN Sparc stations (Falcon and Ibis) and one Motorola VME workstation (Heron), as shown in Figure 11.

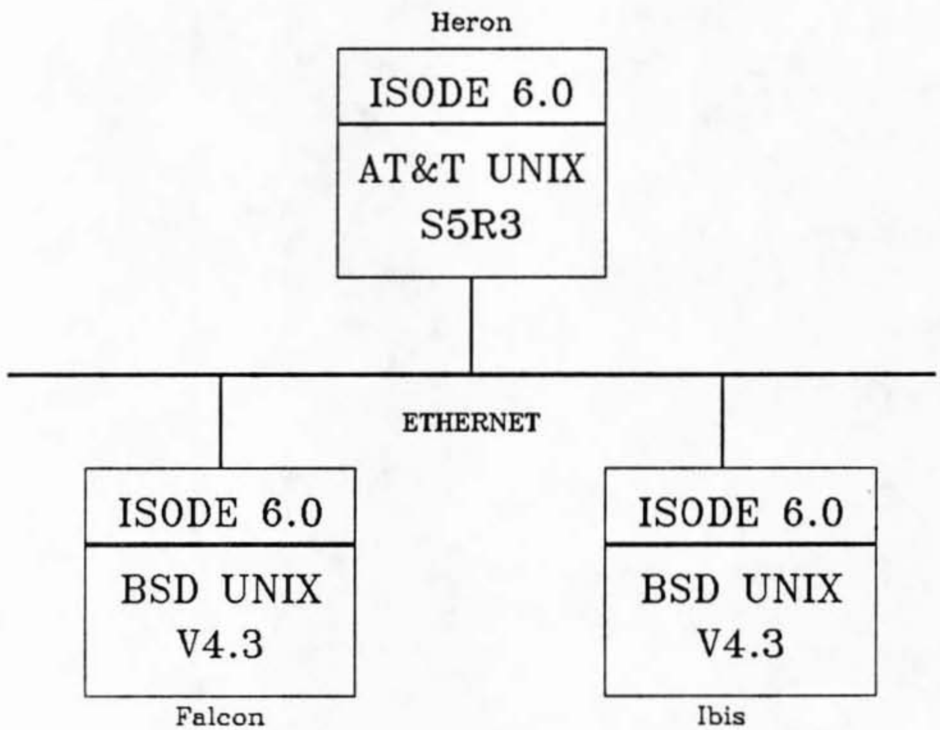

Figure 11. Experimental Network

The experiment was conducted in two parts. The first part of the experiment allowed the SUN workstation **Ibis** to communicate with the Motorola workstation **Heron** using the ISODE communication architecture and the **SEND** programs. The second experiment consisted of both SUN workstation, **Falcon** and **Ibis**, transmitting generic PDUs. In both experiments, the computer's CPUs were dedicated to performing this communication task. During the experiments, activity on the network was limited to the generic PDUs being transmitted. The network was used essentially as a point-to-point link [SHEN91].

The results of the first part of the experiment are shown in Figure 12. The graph indicates that as the size of the observations (samples) increase, the mean value of the round trip time tends to converge. The graph also shows correlation between the time elapsed for the message to perform a round trip between two computers and the size of the message transmitted. This graph indicates that as the DIS PDU increases in size, more time will be required to transmit these PDUs across the network. Therefore, the DIS PDU which will potentially exhibit the largest latency problems will be the Entity State PDU. This latency will vary depending on the degree of complexity modeled in an entity. For example, a tank is less detailed than a ship.

Figure 13 plots the message size versus the round trip time for 500 iterations of the test. This graph indicates that the variance remains approximately unchanged through out the experiment. This confirms that the experiments were conducted in an adequately controlled environment.

The results of the second part of the experiment are shown in Figure 14. The profiles of this graph are similar to those in Figure 12. However, the RTT is significantly lower. This demonstrates that the faster RISC microprocessors on the SUN workstations accounted for a significant portion of the communication performance in this experiment. Again, as the size of the PDU transmitted increases, the RTT also increases.

Figure 15 presents the variance behavior in the second experiment for a sample size of 500. This graph indicates that the environment was adequately controlled for this experiment. Also, when comparing Figure 13 with Figure 15, the difference in workstation processors becomes more obvious. The median RTT in Figure 13 is approximately 50ms, while the median RTT in Figure 15 is approximately 32ms.

(From Ibis to Heron)

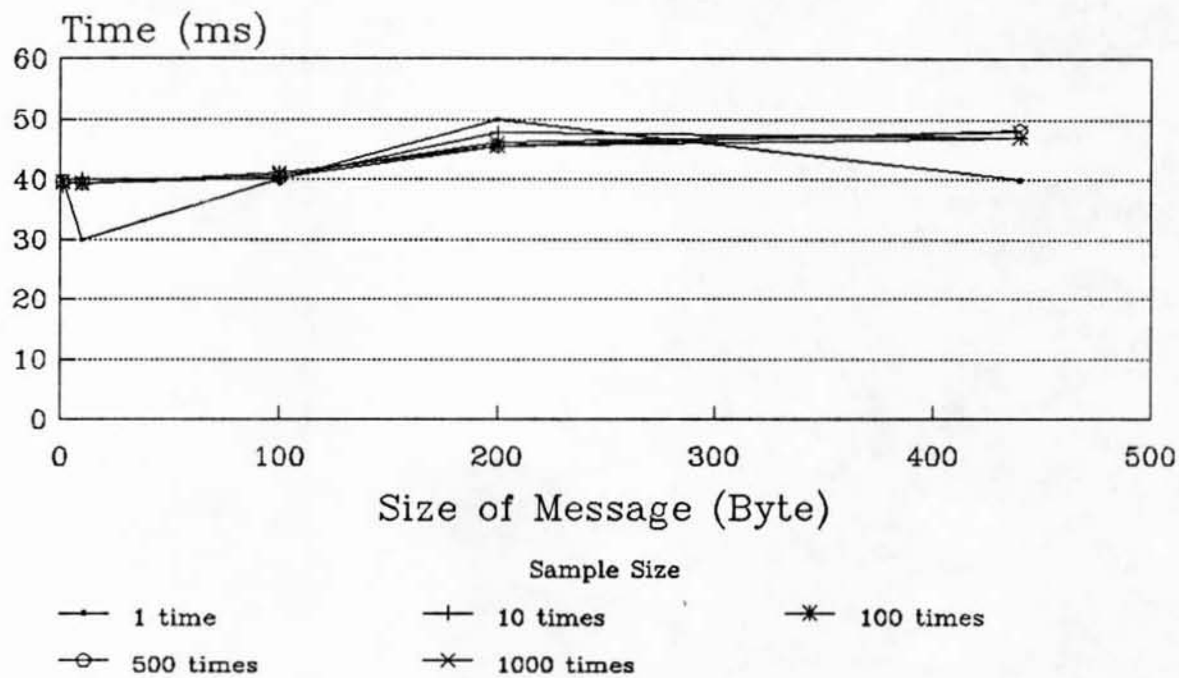

Figure 12. Experiment 1: Message Size vs. Average Round Trip Time

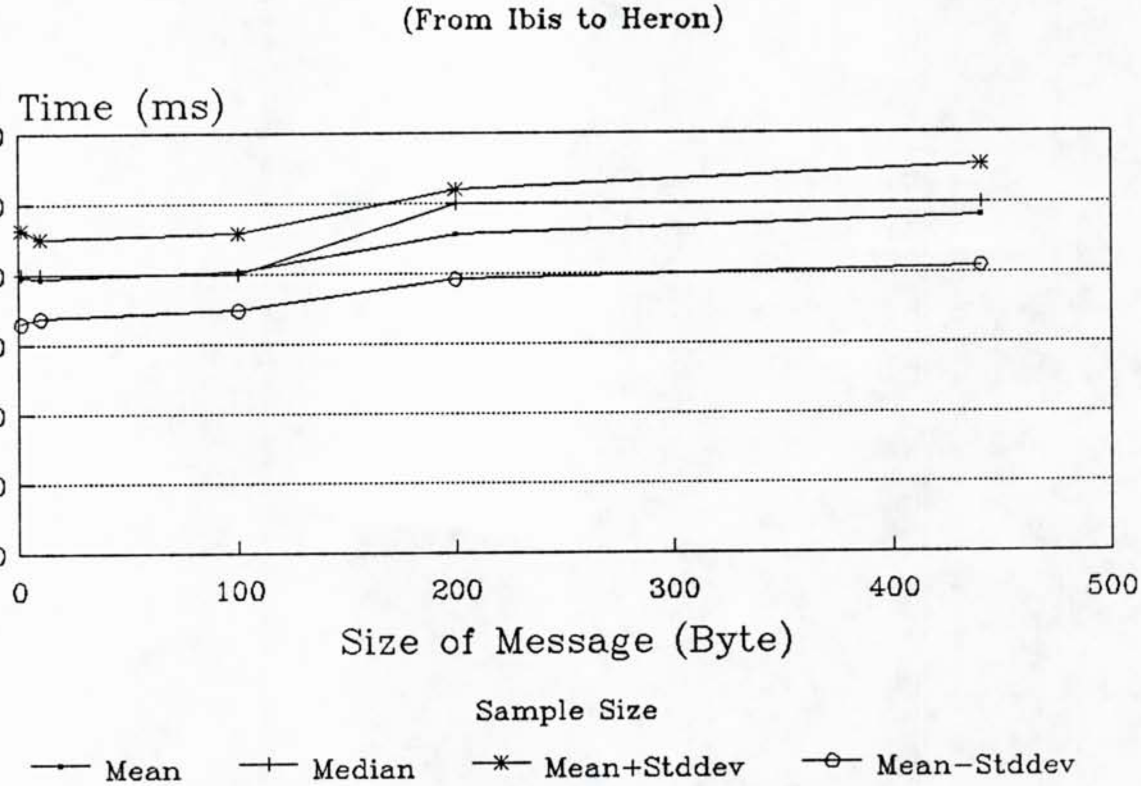

Figure 13. Experiment 1: Message Size vs. Average Round Trip Time

(Sample Size = 500)

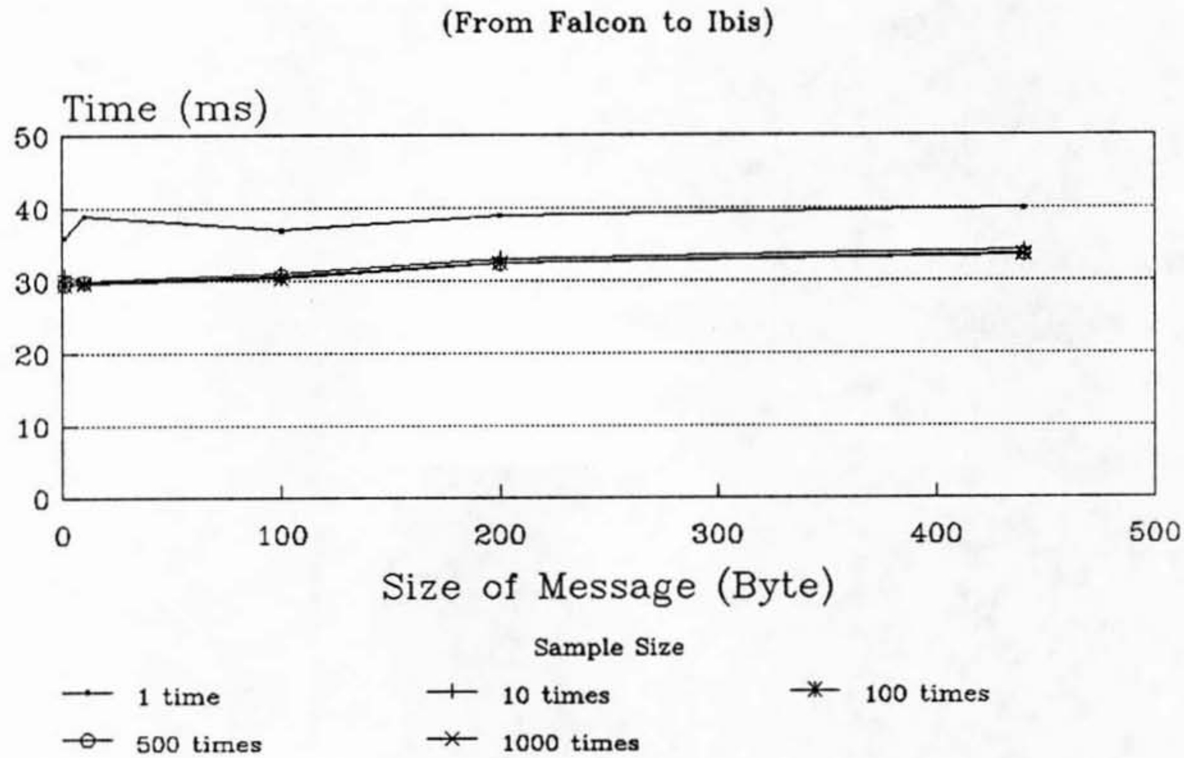

Figure 14. Experiment 2: Message Size vs. Average Round Trip Time

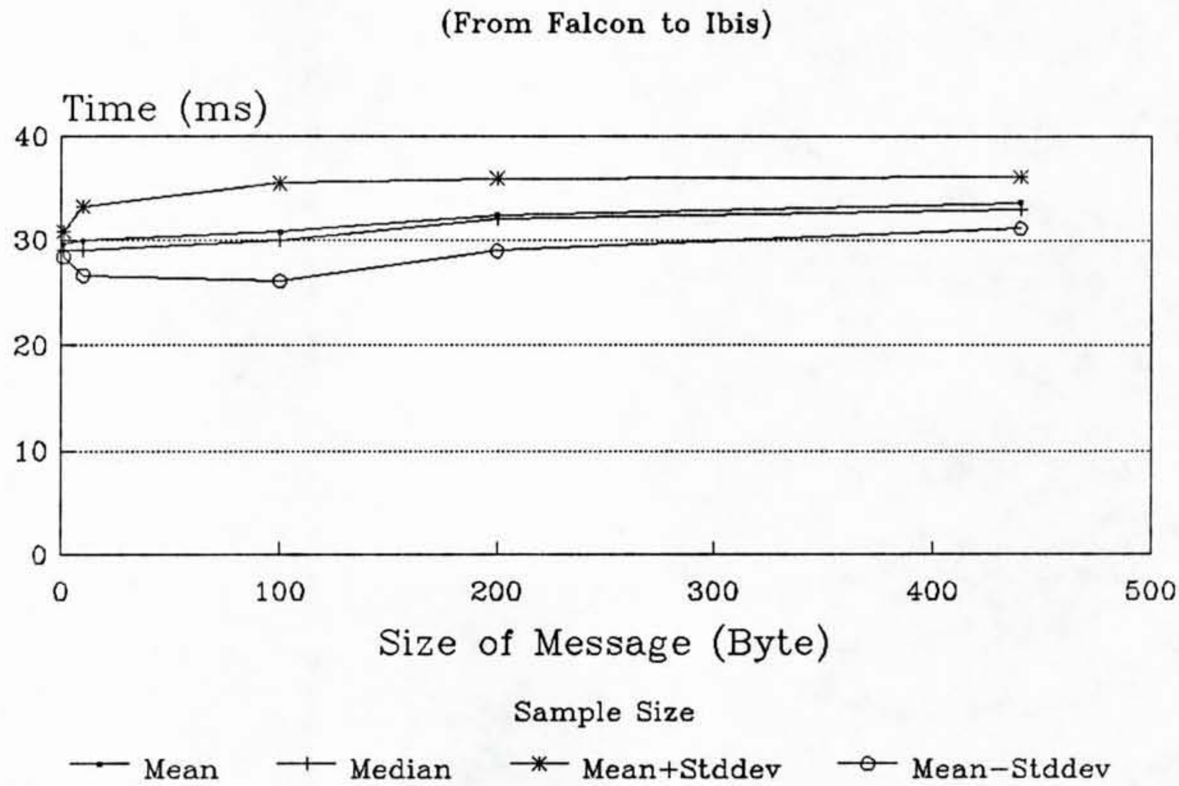

Figure 15. Experiment 2: Message Size vs. Average Round Trip Time

(Sample Size = 500)

CHAPTER IV

CONCLUSIONS

The objectives of this research were the following: describe the Distributed Interactive Simulation (DIS) application and its requirements; introduce the Open System Interconnection (OSI) Reference Model and its applicability to the DIS application; present an overview of the ISO Development Environment (ISODE), its services and facilities; describe a prototype architecture for the DIS application developed using the ISODE; present an experiment and test plan for evaluating the prototype DIS architecture; and discuss the data obtained and analysis thereof.

The DIS application is still an evolving standard which has the potential to bridge the communication barrier for all military simulators and training devices. The Protocol Data Units (PDUs) for describing entity appearance and entity interactions are emerging. However, the communication architecture through which DIS will transmit this information is still being studied and specified. Since the objective of the DIS initiative is to provide an open communications environment in which DIS vendors can develop standard DIS compliant products, the OSI protocol suite and the GOSIP mandate will have a tremendous effect on the communication architecture selected and the protocols developed.

The primary rationale for utilizing the ISODE to implement a prototype DIS architecture was to gain insight into the details of working with an OSI compliant communications protocol stack. Clearly, an actual DIS implementation within a simulator systems using ISODE would be ludicrous. However, the nature of this work was research and from that point of view, the project produced surprising results.

All of the experiments were conducted on workstations running standard (AT&T or Berkley) UNIX operating systems. Due to the real-time nature of the DIS application, standard UNIX would most probably have to be replaced by a more real-time UNIX type of operating system. And even in the experiments conducted as part of this thesis, it is almost impossible to ascertain the impact of UNIX related delays on the statistical data gathered. Perhaps some performance differences between the Motorola and SUN workstations can be attributed to the different implementations of the UNIX operating system. The majority of the performance differences in the SUN SPARC workstation and the Motorola workstations is a consequence of the microprocessors used in each. The SUN workstations use Reduced Instruction Set Computing (RISC) technology which has demonstrated a compilation speed of nearly four times faster than the Motorola workstation [LOPE91].

From a long term perspective, the applicability of OSI to DIS will be demonstrated by means of an evolving and iterative process. Until actual implementations of the DIS protocol are developed, one can only theorize on how far DIS will be able to comply with the OSI guidelines. Moreover, if the DIS does, in fact, become embraced globally in the simulation marketplace as the standard for simulation become embraced globally in the simulation marketplace as the standard for simulation networking, it may very well drive the direction of OSI to some extent. The major problem here is that the market for DIS products is very limited today (e.g., U.S.Government procurements for military training devices) and the incentive for industry to migrate to the DIS voluntarily is minimal. If however, the DIS does continue its evolution along the OSI guidelines, the task of integrating actual OSI-compliant communications systems with DIS systems will be much easier.

Some comments concerning the prototype architecture presented herein are warranted. Due to the nature of the ISODE implementation, all experimental interactions between systems in which data were transmitted over the prototype DIS network stack were done so under connection-oriented constraints. In an actual DIS implementation, connection-less services would be utilized, due to the multicast (sending PDUs from one-to-many instead of one-to-one) nature of the protocol. While time measurements taken as a part of the experimental analysis attempted to isolate the effects of connection establishment and release, it is extremely difficult to guarantee that these effects were totally negated.

There are also several points which should be highlighted concerning the ASN.1 implementation. Under all experiments conducted herein, the DIS Entity State PDUs were "pre-coded" in ASN.1 before they were actually transmitted over the prototype DIS protocol stack. In actual OSI applications, ASN encoding and decoding is performed "on-the-fly", or during the run time of the application. The current viewpoint of the DIS Communications Architecture working group is that the overhead associated with the run-time ASN encoding and decoding of DIS PDUs will be too great for the real-time constraints. Inasmuch as the experiments carried out as part of this thesis adopt the DIS Communications Architecture viewpoint, there are still many experiments which must be carried out to determine whether or not the departure from conventional OSI implementations is warranted.

The performance data presented herein represents a first attempt at utilizing the ISODE system and associated tools to build a prototype and pass data over a communications protocol stack with the networking requirements of DIS (e.g., real-time performance) being held as paramount. The ISODE is a massive software system, in its own right. And the process of obtaining, installing, and utilizing ISODE has taken over two man years worth of engineering effort. The performance measurements give some insight into the delay times associated with the ISODE stack processing of the DIS prototype implementation. However, and more importantly, the fact that any performance statistics were gathered at all demonstrates an in-depth understanding of not only the ISODE system, but the UNIX system as well. And while the ISODE is a good jumping-off point for a research environment, its lack of documentation makes it probably unsuitable for use in an industrial, developmental environment.

APPENDICES

APPENDIX A

DISTRIBUTED INTERACTIVE SIMULATION

PROTOCOL DATA UNIT FORMATS

DIS PDU Header

FIELD SIZE (bits)	PROTOCOL DATA UNIT HEADER FIELDS	
8	PROTOCOL VERSION	8 - bit unsigned integer
8	EXERCISE INDENTIFIER	8 - bit unsigned integer
8	PDU TYPE	8 - bit enumeration
8	PADDING	8 bits unused

Entity State PDU

FIELD SIZE (bits)	ENTITY STATE PDU FIELDS	
48	ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
16	PADDING	16 bits unused
64	ENTITY TYPE	ENTITY KIND - 8 - bit enumeration
		DOMAIN - 8 - bit enumeration
		COUNTRY - 16 - bit enumeration
		CATEGORY - 8 - bit enumeration
		SUBCATEGORY - 8 - bit enumeration
		SPECIFIC - 8 - bit enumeration
		EXTRA - 8 - bit enumeration
32	TIME STAMP	32 - bit unsigned integer
192	ENTITY LOCATION	X - Component - 64 - bit floating point
		Y - Component - 64 - bit floating point
		Z - Component - 64 - bit floating point
96	ENTITY LINEAR VELOCITY	X - Component - 32 - bit floating point
		Y - Component - 32 - bit floating point
		Z - Component- 32 - bit floating point
96	ENTITY LINEAR ACCELERATION	X - Component - 32 - bit floating point
		Y - Component - 32 - bit floating point
		Z - Component - 32 - bit floating point

Entity State PDU (Cont.)

FIELD SIZE (bits)	ENTITY STATE PDU FIELDS (CONT'D)	
96	ENTITY ORIENTATION	Psi 32 - bit BAM
		Theta 32 - bit BAM
		Phi 32 - bit BAM
96	ENTITY ANGULAR VELOCITY	X - Component - 32 - bit signed integer
		Y - Component - 32 - bit signed integer
		Z - Component - 32 - bit signed integer
64	DEAD RECKONING PARAMETERS	64 bits - undefined
32	ENTITY APPEARANCE	32 - bit unsigned integer
96	ENTITY MARKING	CHARACTER SET - 8 - bit enumeration
		11 element character string
32	CAPABILITIES	32 bits of Boolean fields
24	PADDING	24 bits unused
8	# of articulated parts	8 - bit unsigned integer
n x (96m + 32)	ARTICULATED PARTS	See Figure G-1, Appendix G

n = # of articulated parts
m = # of articulation parameters for each part
(i = 1 to n)

Fire PDU

FIELD SIZE (bits)	FIRE PDU FIELDS	
48	FIRING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	TARGET ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	MUNITION ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	EVENT-ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		EVENT - 16 - bit unsigned integer
32	TIME STAMP	32 - bit unsigned integer
192	LOCATION IN WORLD	X-coordinate- 64 - bit floating pt
		Y-coordinate- 64 - bit floating pt
		Z-coordinate- 64 - bit floating pt
128	BURST DESCRIPTOR	MUNITION - See Entity Type Record
		WARHEAD - 16 - bit enumeration
		FUZE - 16 - bit enumeration
		QUANTITY - 16 - bit unsigned integer
		RATE -16 - bit unsigned integer
96	VELOCITY	X-component 32 - bit floating pt
		Y-component 32 - bit floating pt
		Z-component 32 - bit floating pt
32	RANGE	32 - bit floating pt

Detonation PDU

FIELD SIZE (bits)	DETONATION PDU FIELDS	
48	FIRING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	TARGET ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	MUNITION ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	EVENT ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		EVENT - 16 - bit unsigned integer
32	TIME STAMP	32 - bit unsigned integer
192	LOCATION IN WORLD	X-coordinate - 64 - bit floating pt
		Y-coordinate - 64 - bit floating pt
		Z-coordinate - 64 - bit floating pt
128	BURST DESCRIPTOR	MUNITION - See Entity Type Record
		WARHEAD - 16 - bit enumeration
		FUZE - 16 - bit enumeration
		QUANTITY - 16 - bit unsigned integer
		RATE - 16 - bit unsigned integer

Detonation PDU (Cont.)

FIELD SIZE (bits)	DETONATION PDU FIELDS (CONT'D)	
96	VELOCITY	X - component - 32 - bit floating pt.
		Y - component - 32 - bit floating pt.
		Z - component - 32 - bit floating pt.
96	LOCATION IN ENTITY COORDINATES	X - coordinate - 32 - bit floating pt.
		Y - coordinate - 32 - bit floating pt.
		Z - coordinate - 32 - bit floating pt.
8	DETONATION RESULT	8 - bit enumeration
24	PADDING	24 bits unused
32	ENERGY	32 - bit floating pt
32	DIRECTION-ALITY	32 - bit floating pt
32	MOMENTUM	32 - bit floating pt

Update Threshold Request PDU

FIELD SIZE (bits)	UPDATE THRESHOLD REQUEST PDU FIELDS	
48	ISSUING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	RESPONDING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
96	LINEAR THRESHOLD	x - 32 - bit floating pt
		y - 32 - bit floating pt
		z - 32 - bit floating pt
96	ROTATIONAL THRESHOLD	Psi - 32 - bit BAM
		Theta - 32 - bit BAM
		Phi - 32 - bit BAM
32	DURATION OF CHANGE	32 - bit unsigned integer

Update Threshold Response PDU

FIELD SIZE (bits)	UPDATE THRESHOLD RESPONSE PDU FIELDS	
48	RESPONDING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	REQUESTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
8	RESULT	8 - bit enumeration
8	REMAINING TIME	8 - bit unsigned integer
16	PADDING	16 bits unused

Service Request PDU

FIELD SIZE (bits)	SERVICE REQUEST PDU FIELDS	
48	REQUESTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	SERVICING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
8	SERVICE TYPE	8 - bit enumeration
8	Number of (n) Supply types	8 - bit unsigned integer
16	PADDING	16 bits unused
n x 96	SUPPLY QUANTITY	Entity Kind - 8 - bit enumeration
		Domain - 8 - bit enumeration
		Country - 16 - bit enumeration
		Category - 8 - bit enumeration
		Subcategory - 8 - bit enumeration
		Specific - 8 - bit enumeration
		Extra - 8 - bit enumeration
		Quantity - 32 - bit floating pt

Resupply Offer PDU

FIELD SIZE (bits)	RESUPPLY OFFER PDU FIELDS	
48	REQUESTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	SUPPLYING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
8	Number of (n) Supply types	8 - bit unsigned integer
24	PADDING	24 bits unused
n x 96	SUPPLY QUANTITY	Entity Kind - 8 - bit enumeration
		Domain - 8 - bit enumeration
		Country - 16 - bit enumeration
		Category - 8 - bit enumeration
		Subcategory - 8 - bit enumeration
		Specific - 8 - bit enumeration
		Extra - 8 - bit enumeration
		Quantity - 32 - bit floating pt

Resupply Received PDU

FIELD SIZE (bits)	RESUPPLY RECEIVED PDU FIELDS	
48	RECEIVING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	SUPPLYING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
8	Number of (n) Supply types	8 - bit unsigned integer
24	PADDING	24 bits unused
n x 96	SUPPLY QUANTITY	Entity Kind - 8 - bit enumeration
		Domain - 8 - bit enumeration
		Country - 16 - bit enumeration
		Category - 8 - bit enumeration
		Subcategory - 8 - bit enumeration
		Specific - 8 - bit enumeration
		Extra - 8 - bit enumeration
		Quantity - 32 - bit floating pt

Resupply Cancel PDU

FIELD SIZE (bits)	RESUPPLY CANCEL PDU FIELDS	
48	RECEIVING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	SUPPLYING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer

Repair Complete PDU

FIELD SIZE (bits)	REPAIR COMPLETE PDU FIELDS	
48	REQUESTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	REPAIRING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
16	REPAIR	REPAIR TYPE-16 - bit enumeration
16	PADDING	16 bits unused

Repair Response PDU

FIELD SIZE (bits)	REPAIR RESPONSE PDU	
48	REQUESTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	REPAIRING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
8	REPAIR RESULT	8 - bit enumeration
24	PADDING	24 bits unused

Collision PDU

FIELD SIZE (bits)	COLLISION PDU FIELDS	
48	ISSUING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
48	COLLIDING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
32	TIME STAMP	32 - bit unsigned integer
48	EVENT ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		EVENT - 16 - bit unsigned integer
16	PADDING	16 bits unused
96	VELOCITY	x - 32 - bit floating pt
		y - 32 - bit floating pt
		z - 32 - bit floating pt
64	MASS	64 - bit floating pt
96	LOCATION (with respect to Entity)	x - 32 - bit floating pt
		y - 32 - bit floating pt
		z - 32 - bit floating pt

Radar PDU

FIELD SIZE (bits)	RADAR PDU FIELDS	
48	EMITTING ENTITY ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
16	PADDING	16 bits unused
32	TIME STAMP	32 - bit unsigned integer
48	EVENT ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		EVENT - 16 - bit unsigned integer
8	PADDING	8 bits unused
8	Number of Radar Systems (n)	8 - bit unsigned integer
n x ($96m_i + 352$)	LOCATION (w/ respect to entity)	x - 32 - bit floating pt
		y - 32 - bit floating pt
		z - 32 - bit floating pt
	RADAR SYSTEM	32 - bit unsigned integer
	POWER	16 - bit integer
	RADAR MODE	8 - bit enumeration
	# ILLUMINED (m_i)	8 - bit unsigned integer
	SPECIFIC DATA	64 - bit integer
	SWEEP	Azimuth center - 32 - bit BAM
		Azimuth sweep - 32 - bit BAM
		Elevation center - 32 - bit BAM
		Elevation sweep - 32 - bit BAM
	TARGET ID	SITE - 16 - bit unsigned integer
		HOST - 16 - bit unsigned integer
		ENTITY - 16 - bit unsigned integer
	PADDING	16 bits unused
	RADAR DATA	32 - bit unsigned integer

352 bits (LOCATION through SWEEP)

$96 \times m_i$ bits ($i = 1$ to n) (TARGET ID through RADAR DATA)

APPENDIX B

SEND PROGRAMS AND DATABASES

```
##############################################################
###################
#
# isobjects - ISODE Objects Database
#
#    Mappings between object descriptors and object
#    identifiers
#
#
# $Header: /f/osi/config/RCS/objects.local,v 7.0 89/11/23
#  21:26:10 mrose Rel $
#
#
# $Log:   objects.local,v $
# Revision 7.0  89/11/23  21:26:10  mrose
# Release 6.0
#
##############################################################
###################


##############################################################
###################
#
# Syntax:
#
#    <object descriptor> <object id>
#
#    Each token is separated by LWSP, though double-quotes may
#    be used to prevent separation
#
##############################################################
###################


##############################################################
###################
# locally defined objects
#    (this section is usually empty...)
##############################################################
###################


##############################################################
###################
#
# $Header: /f/osi/support/RCS/objects.db,v 7.1 90/01/11
# 18:38:03 mrose Exp $
#
#
# $Log:   objects.db,v $
# Revision 7.1  90/01/11  18:38:03  mrose
# real-sync
```

```
#
# Revision 7.0  89/11/23  22:27:43  mrose
# Release 6.0
#
###############################################################
###################


###############################################################
###################
# ISO ASN.1
###############################################################
###################

# iso standard 8824
"iso asn.1 abstract syntax"   1.0.8824

# iso standard 8825
"iso asn.1 abstract transfer" 1.0.8825

# joint-iso-ccitt asn1(1) basic-encoding(1)
"basic encoding of a single asn.1 type" 2.1.1
# temporary (for backwards compatibility)
"asn.1 basic encoding"        2.1.1


###############################################################
###################
# ISO ASSOCIATION CONTROL
###############################################################
###################

#  joint-iso-ccitt  associationControl(2)  abstractSyntax(1)
apdus(0) version1(1)
"acse pci version 1"          2.2.1.0.1


###############################################################
###################
# ISO/CCITT RELIABLE TRANSFER
###############################################################
###################

# joint-iso-ccitt reliable-transfer (3) apdus (0)
"rtse pci version 1"          2.3.0

# joint-iso-ccitt reliable-transfer (3) aseID (1)
"rtse ase identifier"         2.3.1

# joint-iso-ccitt reliable-transfer (3) abstract-syntax (2)
"rtse abstract syntax"        2.3.2
```

```
################################################################
###################
# ISO/CCITT REMOTE OPERATIONS
################################################################
###################

# joint-iso-ccitt remote-operations (4) notation (0)
"rose notation"                 2.4.0

# joint-iso-ccitt remote-operations (4) apdus (1)
"rose pci version 1"            2.4.1

# joint-iso-ccitt remote-operations (4) notation-extension (2)
"rose notation-extension"       2.4.2

# joint-iso-ccitt remote-operations (4) aseID (3)
"rose ase identifier"           2.4.3

# joint-iso-ccitt remote-operations (4) aseID-ACSE (4)
"rose ase identifier ACSE"      2.4.4


################################################################
###################
# ISO/CCITT DIRECTORY SERVICES
################################################################
###################

# joint-iso-ccitt ds(5) applicationContext(3)
# directoryAccessAC(1)
"directory directoryAccessAC" 2.5.3.1

# joint-iso-ccitt ds(5) applicationContext(3)
# directorySystemAC(2)
"directory directorySystemAC" 2.5.3.2

# joint-iso-ccitt ds(5) abstractService(9)
# directoryAccessAS(1)
"directory directoryAccessAS" 2.5.9.1

# joint-iso-ccitt ds(5) abstractService(9)
# directorySystemAS(2)
"directory directorySystemAS" 2.5.9.2


################################################################
###################
# ISO FTAM
################################################################
###################

# iso standard 8571 abstract-syntax (2) ftam-pci (1)
"ftam pci"                 1.0.8571.2.1
```

```
# iso standard 8571 application-context (1) iso-ftam (1)
"iso ftam"                  1.0.8571.1.1

# nbs-ad-hoc ftam-nil-ap-title (7)
"nil AP title"              1.3.9999.1.7
"null AP title"             1.3.9999.5.1

### BEGIN DIS FTAM ###

# iso standard 8571 transfer-syntax (3) ftam-pci (1)
"ftam pci transfer syntax"      1.0.8571.3.1

### END DIS FTAM ###


###############################################################
###################
# ISO VT
###############################################################
###################

# TEMPORARY
"iso vt pci"            1.17.1.10.1
"iso vt"                1.17.1.10.2


"telnet"                1.3.9999.1.8.0
"forms"                 1.3.9999.1.8.1
"default"               1.3.9999.1.8.2


###############################################################
###################
# ISO CMIP
###############################################################
###################

# iso standard 9596 abstractSyntax(0) cmip-pci(0)
"cmip pci"                  1.0.9596.0.0

# iso standard 9596 cmip(2) version(1) acse(0)
# functional-units(0)
"cmip initialize pci"           1.0.9596.2.1.0.0

# iso standard 9596 cmip(2) version(1) acse(0)
# m-abort-source(1)
"cmip abort pci"            1.0.9596.2.1.0.1

# TEMPORARY application-context until 9596-2 gets its act
# together
"iso cmip"                  1.17.1.11.2
```

```
###############################################################
###################
#
#    ISODE Object Identifiers
#
#    The PCI is an object identifier
#        without asking ISO's permission, we usurp the tree of
#        object identifiers that start with sub-elements
#        "1.17"
#
###############################################################
###################

# reserved for ISODE debug aids: 1.17.0
#    1.17.0.n.1      pci for debug
#    1.17.0.n.2      application context for debug

#    1.17.0.1  isode echo
"isode echo pci"          1.17.0.1.1

#    1.17.0.2  isode sink
"isode sink pci"          1.17.0.2.1

# reserved for ISODE demo programs: 1.17.1
#    1.17.1.n.1      pci for demo
#    1.17.1.n.2      application context for demo

#    1.17.1.1  isode miscellany
"isode miscellany pci"         1.17.1.1.1
"isode miscellany"        1.17.1.1.2

#    1.17.1.2  isode image (obsolete)

#    1.17.1.3  isode callback demo
#              reserved (not actually used yet)
"isode callback demo pci"      1.17.1.3.1
"isode callback demo"          1.17.1.3.2

#    1.17.1.4  isode listen demo
#              reserved (not actually used yet)
"isode listen demo pci"        1.17.1.4.1
"isode listen demo"       1.17.1.4.2

#    1.17.1.5  isode passwd lookup demo
"isode passwd lookup demo pci"      1.17.1.5.1
"isode passwd lookup demo"     1.17.1.5.2

#    1.17.1.6  isode dbm demo
"isode dbm demo pci"      1.17.1.6.1
"isode dbm demo"     1.17.1.6.2

#    1.17.1.7  obsolete
```

```
#    1.17.1.8  isode shell
"isode shell"          1.17.1.8.1
"isode shell pci"      1.17.1.8.2

#    1.17.1.9  isode idist
"isode idist"          1.17.1.9.1
"isode idist pci"      1.17.1.9.2


#    1.17.1.10 VT (temporary)

#    1.17.1.11 CMIP (temporary)

#    1.17.1.12 Z39.50 (temporary)
"IRP Z39.50"           1.17.1.12.2

#    1.17.1.5  isode passwd lookup demo
"isode send string demo pci"  1.17.1.13.1
"isode send string demo"      1.17.1.13.2

# reserved for local ISODE programs: 1.17.2
#    1.17.2.n.1     pci for local program
#    1.17.2.n.2     application context for local program

# additions for local ISODE programs are made to the site's
# objects.local file


# reserved for ISODE FTAM document types: 1.17.3
#    see the isodocuments(5) file


# reserved for application entity titles: 1.17.4
#    1.17.4.0  templates for services
#    1.17.4.1  templates for local services
#    1.17.4.2  examples of specific services
#    1.17.4.3  specific    services    under    different
administrations

# reserved for use with Directory attributes: 1.17.5
#    1.17.5.0  reserved
#    1.17.5.1  attributes under different administrations
#              (numbers parallel to 1.17.4.3 tree)
```

```
###############################################################
###################
#
# isoentities - ISODE Application Entity Title Database
#
#    Application Entity Titles as per ACSE
#
#    This file takes the place of "real" directory services;
#    OIDs are used for AETs, rather than Distinguished Names.
#
#
# $Header: /f/osi/support/RCS/entities.prefix,v 7.0 89/11/23
# 22:27:11 mrose Rel $
#
#
# $Log:   entities.prefix,v $
# Revision 7.0  89/11/23  22:27:11  mrose
# Release 6.0
#
###############################################################
###################


###############################################################
###################
#
# Syntax:
#
#    <host> <service> <aet> <paddr>
#
#    Each token is separated by LWSP, though double-quotes may
#    be used to prevent separation
#
###############################################################
###################


###############################################################
###################
#
# Application entity titles: 1.17.4
#
#    1.17.4.0  templates for services
#    1.17.4.1  templates for local services
#    1.17.4.2  examples of specific services
#    1.17.4.3  specific services under different
#    administrations
#
###############################################################
###################

# templates for services: 1.17.4.0
```

```
default         default         1.17.4.0.0

# this is where ISODE 3.0 FTAM (DIS over DIS) lived
#default  filestore 1.17.4.0.1     #256/

default         "isode/echo"    1.17.4.0.2      #512/

default         "isode/rtse echo" 1.17.4.0.3    #513/

default         "isode/ros_echo" 1.17.4.0.4     #514/

default         "isode/sink"    1.17.4.0.5      #515/

default         "isode/rtse sink" 1.17.4.0.6    #516/

default         "isode/ros_sink" 1.17.4.0.7     #517/

default         "isode miscellany" 1.17.4.0.8 #518/

default         imagestore      1.17.4.0.9      #519/

default         "isode callback demo" 1.17.4.0.10 #520/

default         "isode listen demo" 1.17.4.0.11

default         passwdstore     1.17.4.0.12     #521/

default         dbmstore  1.17.4.0.13     #522/

# temporary until the FTAM/FTP gateway is co-resident with the
# FTAM responder
default         ftpstore  1.17.4.0.13     #523/

default         directory 1.17.4.0.14     #257/

# this is where ISODE 4.0 FTAM (DIS over IS) lived
#default  disfilestore    1.17.4.0.15     #258/

# this is where ISODE 5.0 (and later) FTAM (IS over IS) lives
default         filestore 1.17.4.0.16     #259/

default         shell     1.17.4.0.17     #524/

default         terminal  1.17.4.0.18     #260/

default         "isode idist"   1.17.4.0.19     #525/

default         mib       1.17.4.0.20     #261/


##################################################################
###################
#
```

```
# $Header: /f/osi/config/RCS/entities.local,v 7.0 89/11/23
# 21:26:03 mrose Rel $
#
#
# $Log:   entities.local,v $
# Revision 7.0  89/11/23  21:26:03  mrose
# Release 6.0
#
###############################################################
###################


# templates for local services: 1.17.4.1
#    local additions go here...
default         passwdstore     1.17.4.1.7      #1040/

default         send            1.17.4.1.8      #1041/

#    local additions end here (do not remove this line)


# examples of specific services: 1.17.4.2
#    this section is empty


###############################################################
###################
#
# $Header: /f/osi/support/RCS/entities.db,v 7.2 90/01/11
# 18:37:58 mrose Exp $
#
#
# $Log:   entities.db,v $
# Revision 7.2  90/01/11  18:37:58  mrose
# real-sync
#
# Revision 7.1  89/12/06  17:29:38  mrose
# update
#
# Revision 7.0  89/11/23  22:27:10  mrose
# Release 6.0
#
###############################################################
###################


###############################################################
###################
#
# specific services under different administrations: 1.17.4.3
#
#    1.17.4.3.0     local administration
#    1.17.4.3.1     Northrop Research and Technology Center
```

```
#     1.17.4.3.2     University College London
#     1.17.4.3.3     Nottingham University
#     1.17.4.3.4     National Physical Laboratory
#     1.17.4.3.5     National Computing Centre
#     1.17.4.3.6     INRIA
#     1.17.4.3.7     Swedish Institute of Computer Science
#     1.17.4.3.8     Televerket (Swedish Telecom)
#     1.17.4.3.9     COS
#     1.17.4.3.10    University of Sussex
#     1.17.4.3.11    CNET
#     1.17.4.3.12    University of Cambridge Computer
#                    Laboratory
#     1.17.4.3.13    The Wollongong Group
#     1.17.4.3.14    CHORUS
#     1.17.4.3.15    RARE
#     1.17.4.3.16    Swiss Federal Institute of Technology
#                    Zurich
#     1.17.4.3.17    Tampere University of Technology
#     1.17.4.3.18    Diab Data AB
#     1.17.4.3.19    AU-system
#     1.17.4.3.20    Swedish Defense Material Administration
#     1.17.4.3.21    NCR Sweden
#     1.17.4.3.22    Swedish Agency for Administration
#                    Development
#     1.17.4.3.23    TeleDelta
#     1.17.4.3.24    TeleLOGIC
#     1.17.4.3.25    Upsala University Computing Center
#     1.17.4.3.26    Royal Institute of Technology
#     1.17.4.3.27    Swedish State Power Board
#     1.17.4.3.28    CSIRO
#     1.17.4.3.29    Brunel University
#     1.17.4.3.30    Heriot-Watt University
#     1.17.4.3.31    Oce Research and Development
#     1.17.4.3.32    NYSERnet Inc.
#     1.17.4.3.33    Finnish University and Research Network
#                    Project
#     1.17.4.3.34    German National Research Center for
#                    Computer Science
#     1.17.4.3.35    Erlangen-Nuernberg University
#     1.17.4.3.36    University of Surrey
#     1.17.4.3.37    Rutgers University
#
#
# most sites contain a single entry:
#
#     site default        1.17.4.3.nn.1.0      ""    ""    ""
#                    network addresses
#
###############################################################
###################


# Northrop Research and Technology Center: 1.17.4.3.1
```

```
#    1.17.4.3.1.1   gremlin

gremlin         default         1.17.4.3.1.1.0 \
            Internet=gremlin.nrtc.northrop.com


# University College London: 1.17.4.3.2

ucl         default         1.17.4.3.2.1.0 \

Int-X25(80)=23421920030013¦Janet=00000511160013¦Internet=128
.16.5.1

hubris          default         1.17.4.3.2.5.0  \

Int-X25(80)=23421920030047¦Janet=00000511160047¦Internet=128
.16.8.3

dir         default         1.17.4.3.2.7.0  \
            Janet=00000511320041


# Nottingham University: 1.17.4.3.3

nott        default         1.17.4.3.3.1.0 \

Int-X25(80)=23426020017299¦Janet=000021000018+PID+03010100


# National Physical Laboratory:    1.17.4.3.4

snow        default         1.17.4.3.4.1.0 \
            Int-X25(80)=23421390110298


# National Computing Centre: 1.17.4.3.5

sol         default         1.17.4.3.4.1.0 \
            Int-X25(80)=23426160013967

zeb         default         1.17.4.3.4.2.0 \
            Int-X25(80)=23426160013957


# INRIA: 1.17.4.3.6

inria           default         1.17.4.3.6.1.0 \
            Int-X25(80)=20807802017036


# Swedish Institute of Computer Science: 1.17.4.3.7

tvtf        default         1.17.4.3.7.1.0 \
```

```
          Int-X25(80)=2402001328


# Televerket (Swedish Telecom): 1.17.4.3.8

sics      default          1.17.4.3.8.1.0 \
          Int-X25(80)=2402001203+PID+03010100


# COS: 1.17.4.3.9

echo      default          1.17.4.3.9.1.0 \
          Int-X25(80)=31342023004600¦Internet=echo


# reserved for University of Sussex: 1.17.4.3.10


# CNET: 1.17.4.3.11

cnet      default          1.17.4.3.11.1.0 \
          Int-X25(80)=20809202045601+PID+03010100


# University of Cambridge Computer Laboratory: 1.17.4.3.12

bescot         default          1.17.4.3.12.1.0 \
          Int-X25(80)=23422233939909¦Janet=00000801317701

# Acting as an NRS distribution center

nrs       filestore 1.17.4.3.12.1.1 \
          #256/Int-X25(80)=23422233939909¦Janet=00000801317701


# The Wollongong Group: 1.17.4.3.13
#    1.17.4.3.13.1  gonzo
#    1.17.4.3.13.2  boomer
#    1.17.4.3.13.3  dart
#    1.17.4.3.13.4  philj

gonzo          default          1.17.4.3.13.1.0 \

Internet=gonzo.twg.com¦Int-X25(80)=31344152401010+PID+030101
00¦NS+47000400080001415240101000000¦NS+49005908002000405 3fe00

gonzo          tsbridge  1.17.4.3.13.1.1 \

Internet=gonzo.twg.com+17004¦Int-X25(80)=31344152401010+PID+
03018000

gonzo          "isode listen demo" 1.17.4.3.13.1.2 \
          #521/Internet=gonzo.twg.com+17001
```

```
boomer          default         1.17.4.3.13.2.0 \
        Internet=boomer

dart    default         1.17.4.3.13.3.0 \
        #1/NS+49005902608c425403fe04¦Internet=dart

philj   default                 1.17.4.3.13.4.0 \
        #1/NS+49005902608c884255fe04¦Internet=philj


# Chorus Systems: 1.17.4.3.14

chorus          default         1.17.4.3.14.1.0 \
        Int-X25(80)=208078091969


# reserved for RARE: 1.17.4.3.15


# Swiss Federal Institute of Technology Zurich: 1.17.4.3.16

multimeth  default  1.17.4.3.16.1.0 \
        Int-X25(80)=22849911084131


# reserved for Tampere University of Technology: 1.17.4.3.17


# Diab Data AB: 1.17.4.3.18

diab    default         1.17.4.3.18.1.0 \
        Int-X25(80)=2402000166+PID+03010100


# reserved for AU-system: 1.17.4.3.19


#  reserved  for  Swedish  Defense  Material  Administration:
1.17.4.3.20


# reserved for NCR Sweden: 1.17.4.3.21


# reserved for Swedish Agency for Administration Development:
1.17.4.3.22


# reserved for TeleDelta: 1.17.4.3.23


# reserved for TeleLOGIC: 1.17.4.3.24
```

```
# reserved for Upsala University Computing Center: 1.17.4.3.25

# reserved for Royal Institute of Technology: 1.17.4.3.26

# reserved for Swedish State Power Board: 1.17.4.3.27

# CSIRO: 1.17.4.3.28

ditmela         default         1.17.4.3.28.1.0 \
          Int-X25(80)=5052334300013+PID+03010100

guppy           default         1.17.4.3.28.2.0 \
          Int-X25(80)=5052334300017+PID+03010100


# Brunel University: 1.17.4.3.29

brunel          default         1.17.4.3.29.1.0 \
          Janet=00004114000001+PID+03010100

bru-me          default         1.17.4.3.29.1.0 \
          Janet=00004113150001+PID+03010100

bru-cc          default         1.17.4.3.29.1.0 \
          Janet=00004113150002+PID+03010100


# Heriot-Watt University: 1.17.4.3.30

ra        default         1.17.4.3.30.1.0 \
          Janet=00007024661010

helios          default         1.17.4.3.30.2.0 \
          Janet=00007024661011

solaris         default         1.17.4.3.30.3.0 \
          Janet=00007024661013


# reserved for Oce Research and Development: 1.17.4.3.31

# NYSERNet Inc.: 1.17.4.3.32
#    1.17.4.3.32.1        osi.nyser.net
#    1.17.4.3.32.2        uu.psi.com
#    1.17.4.3.32.4        oclc

osi.nyser.net  default         1.17.4.3.32.1.0 \

Internet=osi.nyser.net|Int-X25(80)=31106070013600+PID+03010100
```

```
osi.nyser.net  Z39.50         1.17.4.3.32.1.1 \
          #1025/Internet=osi.nyser.net

uu.psi.com     default   1.17.4.3.32.2.0 \
          Internet=uu.psi.com

oclc           Z39.50    1.17.4.3.32.4.1 \
          #1025/Int-X25(80)=31106140003659+PID+03010100

# reserved for Finnish University and Research Network
# Project: 1.17.4.3.33

# reserved for German National Research Center for Computer
# Science: 1.17.4.3.34

# Erlangen-Nuernberg University: 1.17.4.3.35

faui45         default        1.17.4.3.35.1.0 \
          Int-X25(80)=26245913144345+PID+03010100


# University of Surrey: 1.17.4.3.36

sur-ee         default        1.17.4.3.36.1.0 \
          Janet=00004800200101+PID+03010100


#
# do not, under any circumstances, remove anything beneath
this line
#

# end of isoentities database
```

```
###########################################################
###################
#
# isoservices - ISODE Services Database
#
#    Mappings between services, selectors, and programs
#
#
# $Header: /f/osi/config/RCS/services.local,v 7.0 89/11/23
# 21:26:17 mrose Rel $
#
#
# $Log:   services.local,v $
# Revision 7.0  89/11/23  21:26:17  mrose
# Release 6.0
#
###########################################################
###################


###########################################################
###################
#
# Syntax:
#
#    <provider>/<entity> <selector> <arg0> <arg1> ... <argn>
#
#    Each token is separated by LWSP, though double-quotes may
#    be used to prevent separation
#
###########################################################
###################


###########################################################
###################
# locally defined services
#    (this section is usually empty...)
###########################################################
###################



# local additions end here (do not remove this line)

###########################################################
###################
#
# $Header: /f/osi/support/RCS/services.db,v 7.1 90/01/11
# 18:38:07 mrose Exp $
#
#
# $Log:   services.db,v $
```

```
# Revision 7.1  90/01/11  18:38:07  mrose
# real-sync
#
# Revision 7.0  89/11/23  22:27:44  mrose
# Release 6.0
#
##############################################################
###################


##############################################################
###################
#
# Entities living above the lightweight presentation service
#
#    Selector is unimportant
#
##############################################################
###################

"lpp/isode miscellany"        ""        lpp.imisc
lpp/mib                  ""        lpp.cmot


##############################################################
###################
#
# Entities living above the transport layer, expressed as TSAP
# IDs
#
#       0         reserved
#     1-127       reserved for GOSIP
#    128-255      GOSIP-style TSAP IDs for ISODE
#    256-511      TSAP selectors for ISO applications
#    512-1023     TSAP selectors for ISODE
#   1024-2047     TSAP selectors reserved for local programs
#   2048-32767    unassigned
#  32768-65535    process-specific
#
##############################################################
###################

# internal server to support asynchronous event INDICATIONs
tsap/isore                   #0       isore

# GOSIP-style addressing
tsap/session                 #1       tsapd-bootstrap

# debug aids
tsap/echo                  #128       isod.tsap
tsap/sink                  #129       isod.tsap

# ISO applications
```

```
# this is where ISODE 5.0 FTAM (IS over IS) lives
"tsap/filestore"           #259       iso.ftam

# this is where ISODE 4.0 FTAM (DIS over IS) lives
"tsap/filestore"           #258       iso.ftam-4.0

# this is where ISODE 3.0 FTAM (DIS over DIS) lived
"tsap/filestore"           #256       iso.ftam-3.0

# QUIPU is a static server
#"tsap/directory"          #257       iso.quipu

"tsap/terminal"            #260       iso.vt

"tsap/mib"                 #261       ros.cmip

"tsap/Z39.50"              #262       iso.z39-50

# ISODE applications
"tsap/isode echo"          #512       isod.acsap
"tsap/isode rtse echo"     #513       isod.rtsap -rtse
"tsap/isode ros_echo"      #514       isod.rtsap -rtse -rose
"tsap/isode sink"          #515       isod.acsap
"tsap/isode rtse sink"     #516       isod.rtsap -rtse
"tsap/isode ros_sink"      #517       isod.rtsap -rtse -rose
"tsap/isode miscellany"    #518       ros.imisc
# imagestore is obsolete
#"tsap/imagestore"         #519       ros.image
"tsap/isode callback demo"      #520       iso.callback
"tsap/passwdstore"         #521       ros.lookup
"tsap/dbmstore"            #522       ros.dbm
# temporary until the FTAM/FTP gateway is co-resident with the
FTAM responder
"tsap/ftpstore"            #523       iso.ftam-ftp
"tsap/shell"               #524       ros.osh
"tsap/isode idist"         #525       ros.idist
"tsap/isode passwd"        #1040      ros.lookup
"tsap/isode send           #1041      ros.send

##############################################################
###################
#
# Entities living above the session layer, expressed as SSAP
# IDs
#
#        0      reserved
#      1-127    reserved for GOSIP
#    128-255    GOSIP-style SSAP IDs for ISODE
#    256-1023   unassigned
#   1024-2047   SSAP selectors reserved for local programs
#   2048-32767  unassigned
#  32768-65535  process-specific
#
```

```
###############################################################################
# GOSIP-style addressing
ssap/presentation       #1      tsapd-bootstrap
ssap/rts                #2      tsapd-bootstrap
ssap/ros                #3      tsapd-bootstrap

# debug aids
ssap/echo             #128      isod.ssap
ssap/sink             #129      isod.ssap


###############################################################################
#
# Entities living above the presentation layer, expressed as
# PSAP IDs
#
#        0      reserved
#      1-127    reserved for GOSIP
#    128-255    GOSIP-style PSAP IDs for ISODE
#    256-1023   unassigned
#   1024-2047   PSAP selectors reserved for local programs
#   2048-32767  unassigned
#  32768-65535  process-specific
#
###############################################################################

# GOSIP-style addressing
psap/ftam               #1      iso.ftam

# debug aids
psap/echo             #128      isod.psap
psap/sink             #129      isod.psap


###############################################################################
#
# Old-style RTS addressing
#
#        0      reserved
#      1-127    reserved for GOSIP
#    128-255    GOSIP-style for ISODE
#
###############################################################################

# mhs
rtsap/p1                1
rtsap/p3                3
```

```
# debug aids
rtsap/echo                    #128      isod.rtsap
rtsap/sink                    #129      isod.rtsap
rtsap/ros_echo                #130      isod.rtsap
rtsap/ros_sink                #131      isod.rtsap
"rtsap/file transfer"              #132      iso.rtf

############################################################
###################
#
# Old-style ROS addressing
#
#       0      reserved
#      1-127   reserved for GOSIP
#    128-255   GOSIP-style for ISODE
#
############################################################
###################

# debug aids
rosap/echo                    #128      isod.rosap
rosap/sink                    #129      isod.rosap
```

```
/*  SEND.C
/*  Program initiator modified from Lookup to perform
*/
/*  the essential communication set-up and funtion definition
*/

#include <stdio.h>
#include <strings.h>
#include <math.h>
#include <sys/time.h>
#include "timer.h"
#include "ryinitiator.h"      /*  for  generic  interctive
initiators */
#include "PDU-ops.h"                  /* operation definitions
*/
#include "PDU-types.h"                   /* type definitions
*/

/*
```

```
   DATA */

/* the following three statements define the name for the
process */
/* and they are related with the ISODE object, entity and
service files */

static char *myservice = "sendstring";
static char *mycontext = "isode send string demo";
static char *mypci = "isode send string demo pci";


                                /* ARGUMENTS */
int  do_send (), do_quit ();

                                /* RESULTS */
int  send_result ();

                                /* ERRORS */
int  send_error ();

/*  This is the declaration of the available processes in this
*/
/*  intiator program with the standard structure dispatch
 */

static struct dispatch dispatches[] = {
    "send", operation_PDU_send,
    do_send, free_PDU_Pdu,
    send_result, send_error,
    "send a pdu",

    "quit", 0,
    do_quit, NULLIFP,
    NULLIFP, NULLIFP,
    "terminate the association and exit",

    NULL
};

/*
```

```
   MAIN */

/* ARGSUSED */

/*  Here is the top level call for the send program. The
ryinitloop */
/*  routine  resides in the ryinitiator.c program
      */

main (argc, argv, envp)
int  argc;
char  **argv,
      **envp;
{
    (void) ryinitloop (argc, argv, myservice, mycontext,
mypci,
               table_PDU_Operations, dispatches, do_quit);

    exit (0);              /* NOTREACHED */
}

/*
```

```
   ARGUMENTS */

/* ARGSUSED */

/* do_send routine checks for the commmand line entered by the
user */
/* and buffers it
       */

static int  do_send (sd, ds, args, arg)
int  sd;
struct dispatch *ds;
char  **args;
register struct type_PDU_Pdu **arg;
{
    char   *cp;

    if ((cp = *args++) == NULL) {
     advise (NULLCP, "usage: send pdu");
     return NOTOK;
    }

    timer(0);
    if ((*arg = str2qb (cp, strlen (cp), 1)) == NULL)
         adios (NULLCP, "out of memory");

    return OK;
}

/*
```

```
 */

/* ARGSUSED */

/* The do_quit routine handles the case when the user request
a */
/* graceful disconnect. Basically it calls the AcRelRequest to
*/
/* perform the disconnection
  */

static int  do_quit (sd, ds, args, dummy)
int  sd;
struct dispatch *ds;
char  **args;
caddr_t *dummy;
{
    struct AcSAPrelease acrs;
    register struct AcSAPrelease   *acr = &acrs;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort *aca = &aci -> aci_abort;

    if (AcRelRequest (sd, ACF_NORMAL, NULLPEP, 0, NOTOK, acr,
aci) == NOTOK)
     acs_adios (aca, "A-RELEASE.REQUEST");

    if (!acr -> acr_affirmative) {
     (void) AcUAbortRequest (sd, NULLPEP, 0, aci);
     adios (NULLCP, "release rejected by peer: %d", acr ->
acr_reason);
    }

    ACRFREE (acr);

    exit (0);
}

/*
```

```
   RESULTS */

/* ARGSUSED */

/*  This is the function that handles the response that comes
back */
/*  from the responder's program. Basically it displays the
message */
/*  string on the terminal
      */

static int  send_result (sd, id, dummy, result, roi)
int  sd,
     id,
     dummy;
register struct type_PDU_Pdu *result;
struct RoSAPindication *roi;
{
    char *tmp;
    tmp=qb2str(result);
    timer(strlen(tmp));
    printf("\n");
    putchar('(');
    printf(tmp);
    printf(") <- echoed back by the responder\n");
    return OK;
}


/*
```

```
   ERRORS */

/* ARGSUSED */

/* send_error routine checks for the error during the message
*/
/* sending
*/

static int  send_error (sd, id, error, parameter, roi)
int  sd,
     id,
     error;
caddr_t parameter;
struct RoSAPindication *roi;
{
    register struct RyError *rye;

    if (error == RY_REJECT) {
     advise (NULLCP, "%s", RoErrString ((int) parameter));
     return OK;
    }

    if (rye = finderrbyerr (table_PDU_Errors, error))
     advise (NULLCP, "%s",  rye -> rye_name);
    else
     advise (NULLCP, "Error %d", error);

    return OK;
}
```

```
/* ryinitiator.c - generic interactive initiator */

#ifndef lint
static char *rcsid = "$Header:
/f/osi/others/lookup/RCS/ryinitiator.c,v 7.0 89/11/23 22:56:41
mrose Rel $";
#endif


#include <stdio.h>
#include <math.h>
#include <varargs.h>
#include "ryinitiator.h"
#include "wait.h"

/*
```

```
   DATA */
static char *myname = "ryinitiator";

extern char *isodeversion;

/*
```

```
   INITIATOR */

ryinitloop (argc, argv, myservice, mycontext, mypci, ops,
dispatches, quit)
int  argc;
char  **argv,
       *myservice,
       *mycontext,
       *mypci;
struct RyOperation ops[];
struct dispatch *dispatches;
IFP  quit;
{
    int  n,i,j;
    int       iloop,
         sd[10];
    char    buffer[BUFSIZ],
        *vec[NVEC + 1];
    register struct dispatch   *ds;
    struct SSAPref sfs;
    register struct SSAPref *sf;
    register struct PSAPaddr *pa[10];
    struct AcSAPconnect accs;
    register struct AcSAPconnect   *acc = &accs;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort *aca = &aci -> aci_abort;
    AEI       aei[10];
    OID       ctx,
         pci;
    struct PSAPctxlist pcs;
    register struct PSAPctxlist *pc = &pcs;
    struct RoSAPindication rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject *rop = &roi -> roi_preject;

    int cnt;
    double tm;
    printf("argc=%d\n",argc);
    if ( tm = atof(argv[argc-1]) )
       argc--;
    else
       tm=0;
    printf("argc=%d  tm=%f\n",argc,tm);
    if ( cnt = atoi(argv[argc-1]) )
       argc--;
    else
       cnt=1;
    printf("argc=%d  cnt=%d\n",argc,cnt);

    if (myname = rindex (argv[0], '/'))
     myname++;
    if (myname == NULL || *myname == NULL)
```

```
     myname = argv[0];

/* checking the command syntax */

    if (argc < 2)
     adios (NULLCP, "usage: %s host [operation [ arguments ...
]]", myname);

/* detecting the number of nodes the connect */
    n=1;
    do {
      n++;
    } while ( (n<argc) && (strcmp(argv[n],"send")) );
    n--;
/**/

    if ((sf = addr2ref (PLocalHostName ())) == NULL) {
     sf = &sfs;
     (void) bzero ((char *) sf, sizeof *sf);
    }

/* this section checks if interactive mode or not and set the
*/
/* value of iloop
*/

    if (argc < n+2) {
     printf ("%s", myname);
     if (sf -> sr_ulen > 2)
         printf (" running on host %s", sf -> sr_udata + 2);
     if (sf -> sr_clen > 2)
         printf (" at %s", sf -> sr_cdata + 2);
        printf (" [%s, ",mycontext);
     printf ("%s]\n", mypci);
     printf ("using %s\n", isodeversion);

        (void) fflush (stdout);
     iloop = 1;
    }
    else {
     for (ds = dispatches; ds -> ds_name; ds++)
         if (strcmp (ds -> ds_name, argv[n+1]) == 0)
          break;
     if (ds -> ds_name == NULL)
         adios    (NULLCP,    "unknown    operation    \"%s\"",
argv[n+1]);

     iloop = 0;
    }

/* This block sets some initial parameters and use tham to
  */
/*  establish   a   connect   and   stores   the   connection
```

```
identification */
/* number in the variable sd
  */

    for (i=1;i<=n;i++)
    { if ((aei[i] = str2aei (argv[i], myservice)) == NULLAEI)
     adios (NULLCP, "%s-%s: unknown application-entity",
          argv[i], myservice);
      if ((pa[i] = aei2addr (aei[i])) == NULLPA)
     adios (NULLCP, "address translation failed");

      if ((ctx = ode2oid (mycontext)) == NULLOID)
     adios  (NULLCP,  "%s:  unknown  object  descriptor",
mycontext);
      if ((ctx = oid_cpy (ctx)) == NULLOID)
     adios (NULLCP, "out of memory");
      if ((pci = ode2oid (mypci)) == NULLOID)
     adios (NULLCP, "%s: unknown object descriptor", mypci);
      if ((pci = oid_cpy (pci)) == NULLOID)
     adios (NULLCP, "out of memory");
      pc -> pc_nctx = 1;
      pc -> pc_ctx[0].pc_id = 1;
      pc -> pc_ctx[0].pc_asn = pci;
      pc -> pc_ctx[0].pc_atn = NULLOID;

      if (iloop) {

     printf ("%s... ", argv[i]);
        (void) fflush (stdout);
      }

      if (AcAssocRequest (ctx, NULLAEI, aei[i], NULLPA, pa[i],
pc, NULLOID,
          0, ROS_MYREQUIRE, SERIAL_NONE , 0, sf, NULLPEP, 0,
NULLQOS,
          acc, aci)
         == NOTOK)
     acs_adios (aca, "A-ASSOCIATE.REQUEST");

      if (acc -> acc_result != ACS_ACCEPT) {
     if (iloop)
         printf ("failed\n");

     adios (NULLCP, "association rejected: [%s]",
          AcErrString (acc -> acc_result));
      }

      if (iloop) {
     printf ("connected\n");
     (void) fflush (stdout);
      }

      sd[i] = acc -> acc_sd;
```

```
       ACCFREE (acc);

/* This function call defines the types of lower layer
services */
/* required by the initiator process (this call can be found
in */
/* the responder side as well in ryresponder.c
  */

       if (RoSetService (sd[i], RoPService, roi) == NOTOK)
      ros_adios (rop, "set RO/PS fails");

     }

/* this block in a infinite loop when interactrive mode is
chosen */
/* and finishes when the user enters the quit command.
Basically  */
/* the program calls the disconnect service to have a graceful
  */
/* disconnection
     */

     if (iloop) {
      for (;;) {
          if (getline (buffer) == NOTOK)
           break;

          if (str2vec (buffer, vec) < 1)
           continue;

          for (ds = dispatches; ds -> ds_name; ds++)
           if (strcmp (ds -> ds_name, vec[0]) == 0)
               break;
          if (ds -> ds_name == NULL) {
           advise (NULLCP, "unknown operation \"%s\"", vec[0]);
           continue;
          }

             for (i=1;i<=n;i++)
                 for (j=1;j<=cnt;j++) {
                   invoke (sd[i], ops, ds, vec + 1);
                   wait(tm);
                 }
      }
     }
     else
         for (i=1;i<=n;i++)
             for (j=1;j<=cnt;j++) {
               invoke (sd[i], ops, ds, argv + n + 2);
               wait(tm);
             }
```

```
    for (i=1;i<=n;i++)
        (*quit) (sd[i], (struct dispatch *) NULL, (char **)
NULL, (caddr_t *) NULL);
}

/*
```

```
 */
/* this routine calls the Rystub to send the name of the
process to */
/* be performed by the responder and the parameter of that
process  */

static   invoke (sd, ops, ds, args)
int  sd;
struct RyOperation ops[];
register struct dispatch *ds;
char  **args;
{
    int        result;
    caddr_t in;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    in = NULL;
    if (ds -> ds_argument && (*ds -> ds_argument) (sd, ds,
args, &in) == NOTOK)
     return;

    switch (result = RyStub (sd, ops, ds -> ds_operation,
RyGenID (sd), NULLIP,
                     in, ds -> ds_result, ds -> ds_error,
ROS_SYNC,
                     roi)) {
     case NOTOK:          /* failure */
         if (ROS_FATAL (rop -> rop_reason))
          ros_adios (rop, "STUB");
         ros_advise (rop, "STUB");
         break;

     case OK:        /* got a result/error response */
         break;

     case DONE:           /* got RO-END? */
         adios (NULLCP, "got RO-END.INDICATION");
         /* NOTREACHED */

     default:
         adios  (NULLCP,  "unknown  return  from  RyStub=%d",
result);
         /* NOTREACHED */
    }

    if (ds -> ds_free && in)
     (void) (*ds -> ds_free) (in);
}

/*
```

```
 */
/* routine that get the line entered by the user and buffers
it */

static int  getline (buffer)
char   *buffer;
{
    register int    i;
    register char  *cp,
                   *ep;
    static int  sticky = 0;

    if (sticky) {
     sticky = 0;
     return NOTOK;
    }

    printf ("%s> ", myname);
    (void) fflush (stdout);

    for (ep = (cp = buffer) + BUFSIZ - 1; (i = getchar ()) !=
'\n';) {
     if (i == EOF) {
         printf ("\n");
         clearerr (stdin);
         if (cp != buffer) {
          sticky++;
          break;
         }

         return NOTOK;
     }

     if (cp < ep)
         *cp++ = i;
    }
    *cp = NULL;

    return OK;
}

/*
```

```
 */

void ros_adios (rop, event)
register struct RoSAPpreject *rop;
char   *event;
{
    ros_advise (rop, event);

    _exit (1);
}


void ros_advise (rop, event)
register struct RoSAPpreject *rop;
char   *event;
{
    char    buffer[BUFSIZ];

    if (rop -> rop_cc > 0)
     (void) sprintf (buffer, "[%s] %*.*s", RoErrString (rop ->
rop_reason),
          rop -> rop_cc, rop -> rop_cc, rop -> rop_data);
    else
     (void)  sprintf  (buffer,  "[%s]",  RoErrString  (rop  ->
rop_reason));

    advise (NULLCP, "%s: %s", event, buffer);
}

/*
```

```
 */

void acs_adios (aca, event)
register struct AcSAPabort *aca;
char   *event;
{
    acs_advise (aca, event);

    _exit (1);
}


void acs_advise (aca, event)
register struct AcSAPabort *aca;
char   *event;
{
    char    buffer[BUFSIZ];

    if (aca -> aca_cc > 0)
     (void) sprintf (buffer, "[%s] %*.*s",
          AcErrString (aca -> aca_reason),
          aca -> aca_cc, aca -> aca_cc, aca -> aca_data);
    else
     (void) sprintf (buffer, "[%s]", AcErrString (aca ->
aca_reason));

     advise (NULLCP, "%s: %s (source %d)", event, buffer,
          aca -> aca_source);
}

/*
```

```
 */

#ifndef   lint
void _advise ();


void adios (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _advise (ap);

    va_end (ap);

    _exit (1);
}
#else
/* VARARGS */

void adios (what, fmt)
char   *what,
       *fmt;
{
    adios (what, fmt);
}
#endif


#ifndef   lint
void advise (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _advise (ap);

    va_end (ap);
}


static void  _advise (ap)
va_list   ap;
{
    char    buffer[BUFSIZ];

    asprintf (buffer, ap);

    (void) fflush (stdout);
```

```
    fprintf (stderr, "%s: ", myname);
    (void) fputs (buffer, stderr);
    (void) fputc ('\n', stderr);

    (void) fflush (stderr);
}
#else
/* VARARGS */

void advise (what, fmt)
char   *what,
       *fmt;
{
    advise (what, fmt);
}
#endif


#ifndef   lint
void ryr_advise (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _advise (ap);

    va_end (ap);
}
#else
/* VARARGS */

void ryr_advise (what, fmt)
char   *what,
       *fmt;
{
    ryr_advise (what, fmt);
}
#endif
```

/* ryinitiator.h - include file for the generic interactive initiator */

```
/*
 *  $Header:  /f/osi/others/lookup/RCS/ryinitiator.h,v  7.0
89/11/23 22:56:43 mrose Rel $
 *
 *
 * $Log:  ryinitiator.h,v $
 * Revision 7.0  89/11/23  22:56:43  mrose
 * Release 6.0
 *
 */

/*
 *                        NOTICE
 *
 *    Acquisition, use, and distribution of this module and
related
 *    materials are subject to the restrictions of a license
agreement.
 *    Consult the Preface in the User's Manual for the full
terms of
 *    this agreement.
 *
 */


#include "rosy.h"


static struct dispatch {
    char   *ds_name;
    int        ds_operation;

    IFP        ds_argument;
    IFP        ds_free;

    IFP        ds_result;
    IFP        ds_error;

    char   *ds_help;
};


void adios (), advise ();
void acs_adios (), acs_advise ();
void ros_adios (), ros_advise ();

int  ryinitiator ();
```

```
/* ryresponder.c - generic idempotent responder */

#ifndef   lint
static char *rcsid = "$Header:
/f/osi/others/lookup/RCS/ryresponder.c,v 7.0 89/11/23 22:56:44
mrose Rel $";
#endif

/*
 *   $Header:  /f/osi/others/lookup/RCS/ryresponder.c,v  7.0
89/11/23 22:56:44 mrose Rel $
 *
 *
 * $Log:  ryresponder.c,v $
 * Revision 7.0  89/11/23  22:56:44  mrose
 * Release 6.0
 *
 */

/*
 *                    NOTICE
 *
 *     Acquisition, use, and distribution of this module and
related
 *    materials are subject to the restrictions of a license
agreement.
 *     Consult the Preface in the User's Manual for the full
terms of
 *    this agreement.
 *
 */


#include <stdio.h>
#include <setjmp.h>
#include <varargs.h>
#include "ryresponder.h"
#include "tsap.h"          /* for listening */

/*
```

```
   DATA */

int  debug = 0;

static LLog _pgm_log = {
    "responder.log", NULLCP, NULLCP,
    LLOG_FATAL | LLOG_EXCEPTIONS | LLOG_NOTICE, LLOG_FATAL,
-1,
    LLOGCLS | LLOGCRT | LLOGZER, NOTOK
};
LLog *pgm_log = &_pgm_log;

static char *myname = "ryresponder";


static jmp_buf toplevel;


static IFP     startfnx;
static IFP     stopfnx;

int  ros_init (), ros_work (), ros_indication (), ros_lose ();


extern int  errno;

/*
```

```
   RESPONDER */
/* this top level routine acts as a responder to the incoming
message */
/* string. Basically it responds toto connection request and
sets the */
/* lower layer services
        */

int  ryresponder (argc, argv, host, myservice, dispatches,
ops, start, stop)
int  argc;
char  **argv,
        *host,
        *myservice;
struct dispatch *dispatches;
struct RyOperation *ops;
IFP  start,
     stop;
{
    register struct dispatch   *ds;
    AEI       aei;
    struct TSAPdisconnect   tds;
    struct TSAPdisconnect  *td = &tds;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    if (myname = rindex (argv[0], '/'))
     myname++;
    if (myname == NULL || *myname == NULL)
     myname = argv[0];

    isodetailor (myname, 0);
    if (debug = isatty (fileno (stderr)))
     ll_dbinit (pgm_log, myname);
    else {
     static char  myfile[BUFSIZ];

     (void) sprintf (myfile, "%s.log",
              (strncmp (myname, "ros.", 4)
                       && strncmp (myname, "lpp.", 4))
                   || myname[4] == NULL
                  ? myname : myname + 4);
     pgm_log -> ll_file = myfile;
     ll_hdinit (pgm_log, myname);
    }

    advise (LLOG_NOTICE, NULLCP, "starting");

    if ((aei = str2aei (host, myservice)) == NULLAEI)
     adios (NULLCP, "%s-%s: unknown application-entity", host,
myservice);
```

```
    for (ds = dispatches; ds -> ds_name; ds++)
     if (RyDispatch (NOTOK, ops, ds -> ds_operation, ds -> 
ds_vector, roi)
          == NOTOK)
         ros_adios (rop, ds -> ds_name);

    startfnx = start;
    stopfnx = stop;

/* this routine handles the initial connection establishment, 
the */
/* message transmissions and the connection release calling 
    */
/* ros_init, ros_work and ros_lose respectively 
    */

    if (isodeserver (argc, argv, aei, ros_init, ros_work, 
ros_lose, td)
         == NOTOK) {
     if (td -> td_cc > 0)
         adios (NULLCP, "isodeserver: [%s] %*.*s",
              TErrString (td -> td_reason),
              td -> td_cc, td -> td_cc, td -> td_data);
     else
         adios (NULLCP, "isodeserver: [%s]",
              TErrString (td -> td_reason));
    }

    return 0;
}

/*
```

```
 */
/* this routine respondes to the connection establishment
request */

static int  ros_init (vecp, vec)
int  vecp;
char  **vec;
{
    int       reply,
         result,
         sd;
    struct AcSAPstart   acss;
    register struct AcSAPstart *acs = &acss;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort   *aca = &aci -> aci_abort;
    register struct PSAPstart *ps = &acs -> acs_start;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    if (AcInit (vecp, vec, acs, aci) == NOTOK) {
     acs_advise (aca, "initialization fails");
     return NOTOK;
    }
    advise (LLOG_NOTICE, NULLCP,
          "A-ASSOCIATE.INDICATION: <%d, %s, %s, %s, %d>",
          acs -> acs_sd, oid2ode (acs -> acs_context),
          sprintaei (&acs -> acs_callingtitle),
          sprintaei  (&acs  ->  acs_calledtitle),  acs  ->
acs_ninfo);

    sd = acs -> acs_sd;

    for (vec++; *vec; vec++)
     advise  (LLOG_EXCEPTIONS,  NULLCP,  "unknown  argument
\"%s\"", *vec);

    reply = startfnx ? (*startfnx) (sd, acs) : ACS_ACCEPT;

    result = AcAssocResponse (sd, reply,
          reply  !=  ACS_ACCEPT  ?  ACS_USER_NOREASON  :
ACS_USER_NULL,
          NULLOID,  NULLAEI,  NULLPA,  NULLPC,  ps  ->
ps_defctxresult,
          ps -> ps_prequirements, ps -> ps_srequirements,
SERIAL_NONE,
          ps -> ps_settings, &ps -> ps_connect, NULLPEP, 0,
aci);

    ACSFREE (acs);

    if (result == NOTOK) {
```

```
     acs_advise (aca, "A-ASSOCIATE.RESPONSE");
     return NOTOK;
    }
    if (reply != ACS_ACCEPT)
     return NOTOK;

    if (RoSetService (sd, RoPService, roi) == NOTOK)
     ros_adios (rop, "set RO/PS fails");

    return sd;
}

/*
```

```
 */
/* This routine handles the incoming message strings sent by
the   */
/* initiator. It waits for the process request sent from the
     */
/* initiator
     */

static int  ros_work (fd)
int  fd;
{
    int       result;
    caddr_t out;
    struct AcSAPindication  acis;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    switch (setjmp (toplevel)) {
     case OK:
         break;

     default:
         if (stopfnx)
          (*stopfnx) (fd, (struct AcSAPfinish *) 0);
     case DONE:
         (void) AcUAbortRequest (fd, NULLPEP, 0, &acis);
         (void) RyLose (fd, roi);
         return NOTOK;
    }

    switch (result = RyWait (fd, NULLIP, &out, OK, roi)) {
     case NOTOK:
         if (rop -> rop_reason == ROS_TIMER)
          break;
     case OK:
     case DONE:
         ros_indication (fd, roi);
         break;

     default:
          adios    (NULLCP,    "unknown    return    from
RoWaitRequest=%d", result);
    }

    return OK;
}

/*
```

```
 */
/* This routine handles the possible errors that might happen
during */
/* the message transmission
        */

static int ros_indication (sd, roi)
int  sd;
register struct RoSAPindication *roi;
{
    int       reply,
         result;

    switch (roi -> roi_type) {
     case ROI_INVOKE:
     case ROI_RESULT:
     case ROI_ERROR:
         adios (NULLCP, "unexpected indication type=%d", roi
-> roi_type);
         break;

     case ROI_UREJECT:
         {
          register  struct  RoSAPureject     *rou  =  &roi  ->
roi_ureject;

          if (rou -> rou_noid)
              advise (LLOG_EXCEPTIONS, NULLCP,
                   "RO-REJECT-U.INDICATION/%d: %s",
                   sd, RoErrString (rou -> rou_reason));
          else
              advise (LLOG_EXCEPTIONS, NULLCP,
                   "RO-REJECT-U.INDICATION/%d: %s (id=%d)",
                   sd, RoErrString (rou -> rou_reason),
                   rou -> rou_id);
         }
         break;

     case ROI_PREJECT:
         {
          register  struct  RoSAPpreject     *rop  =  &roi  ->
roi_preject;

          if (ROS_FATAL (rop -> rop_reason))
              ros_adios (rop, "RO-REJECT-P.INDICATION");
          ros_advise (rop, "RO-REJECT-P.INDICATION");
         }
         break;

     case ROI_FINISH:
         {
          register   struct   AcSAPfinish  *acf   =   &roi   ->
roi_finish;
```

```
          struct AcSAPindication  acis;
          register struct AcSAPabort *aca = &acis.aci_abort;

          advise  (LLOG_NOTICE,  NULLCP,
"A-RELEASE.INDICATION/%d: %d",
               sd, acf -> acf_reason);

          reply = stopfnx ? (*stopfnx) (sd, acf) : ACS_ACCEPT;

          result  =  AcRelResponse  (sd,  reply,  ACR_NORMAL,
NULLPEP, 0,
                   &acis);

          ACFFREE (acf);

          if (result == NOTOK)
              acs_advise (aca, "A-RELEASE.RESPONSE");
          else
              if (reply != ACS_ACCEPT)
               break;
          longjmp (toplevel, DONE);
         }
     /* NOTREACHED */

     default:
         adios (NULLCP, "unknown indication type=%d", roi ->
roi_type);
    }
}

/*
```

```
 */

static int  ros_lose (td)
struct TSAPdisconnect *td;
{
    if (td -> td_cc > 0)
     adios (NULLCP, "TNetAccept: [%s] %*.*s",
          TErrString (td -> td_reason), td -> td_cc, td ->
td_cc,
          td -> td_data);
    else
     adios (NULLCP, "TNetAccept: [%s]", TErrString (td ->
td_reason));
}

/*
```

```
   ERRORS */

void ros_adios (rop, event)
register struct RoSAPpreject *rop;
char   *event;
{
    ros_advise (rop, event);

    longjmp (toplevel, NOTOK);
}


void ros_advise (rop, event)
register struct RoSAPpreject *rop;
char   *event;
{
    char     buffer[BUFSIZ];

    if (rop -> rop_cc > 0)
     (void) sprintf (buffer, "[%s] %*.*s", RoErrString (rop ->
rop_reason),
             rop -> rop_cc, rop -> rop_cc, rop -> rop_data);
    else
     (void)  sprintf  (buffer,  "[%s]",  RoErrString  (rop  ->
rop_reason));

    advise (LLOG_EXCEPTIONS, NULLCP, "%s: %s", event, buffer);
}

/*
```

```
 */

void acs_advise (aca, event)
register struct AcSAPabort *aca;
char   *event;
{
    char    buffer[BUFSIZ];

    if (aca -> aca_cc > 0)
     (void) sprintf (buffer, "[%s] %*.*s",
          AcErrString (aca -> aca_reason),
          aca -> aca_cc, aca -> aca_cc, aca -> aca_data);
    else
     (void) sprintf (buffer, "[%s]", AcErrString (aca ->
aca_reason));

    advise (LLOG_EXCEPTIONS, NULLCP, "%s: %s (source %d)",
event, buffer,
          aca -> aca_source);
}

/*
```

```
 */

#ifndef   lint
void adios (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _ll_log (pgm_log, LLOG_FATAL, ap);

    va_end (ap);

    _exit (1);
}
#else
/* VARARGS2 */

void adios (what, fmt)
char   *what,
       *fmt;
{
    adios (what, fmt);
}
#endif


#ifndef   lint
void advise (va_alist)
va_dcl
{
    int       code;
    va_list ap;

    va_start (ap);

    code = va_arg (ap, int);

    _ll_log (pgm_log, code, ap);

    va_end (ap);
}
#else
/* VARARGS3 */

void advise (code, what, fmt)
char   *what,
       *fmt;
int  code;
{
    advise (code, what, fmt);
}
```

```
#endif


#ifndef   lint
void ryr_advise (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _ll_log (pgm_log, LLOG_NOTICE, ap);

    va_end (ap);
}
#else
/* VARARGS2 */

void ryr_advise (what, fmt)
char   *what,
       *fmt;
{
    ryr_advise (what, fmt);
}
#endif
```

```
/* ryresponder.h - include file for the generic idempotent
responder */

/*
 *  $Header:  /f/osi/others/lookup/RCS/ryresponder.h,v  7.0
89/11/23 22:56:46 mrose Rel $
 *
 *
 * $Log:  ryresponder.h,v $
 * Revision 7.0  89/11/23  22:56:46  mrose
 * Release 6.0
 *
 */

/*
 *                      NOTICE
 *
 *    Acquisition, use, and distribution of this module and
related
 *    materials are subject to the restrictions of a license
agreement.
 *    Consult the Preface in the User's Manual for the full
terms of
 *    this agreement.
 *
 */


#include "rosy.h"
#include "logger.h"


static struct dispatch {
    char   *ds_name;
    int         ds_operation;

    IFP         ds_vector;
};


extern int  debug;


void adios (), advise ();
void acs_advise ();
void ros_adios (), ros_advise ();
void ryr_advise ();

int  ryresponder ();
```

```
/* timer.h - timer utility -- common subroutines */

#ifndef   lint
#endif

/*
 *   $Header:  /f/osi/others/lookup/timer.c,v  7.0  89/11/23
22:10:50 mrose Rel $
 *
 *
 * Revision 7.0  89/11/23  22:10:50  mrose
 * Release 6.0
 *
 */

/*
 *                        NOTICE
 *
 *     Acquisition, use, and distribution of this module and
related
 *    materials are subject to the restrictions of a license
agreement.
 *     Consult the Preface in the User's Manual for the full
terms of
 *    this agreement.
 *
 */

#include <varargs.h>
#if defined(SYS5) && !defined(HPUX)
#include <sys/times.h>
#define   TMS
#endif


/*
```

```
 */

#ifndef   NBBY
#define   NBBY 8
#endif


#ifndef   TMS
timer (cc)
int     cc;
{
    long    ms;
    float   bs;
    struct timeval  stop,
                    td;
    static struct timeval   start;

    if (cc == 0) {
     (void) gettimeofday (&start, (struct timezone *) 0);
     return;
    }
    else
     (void) gettimeofday (&stop, (struct timezone  *) 0);

    tvsub (&td, &stop, &start);
    ms = (td.tv_sec * 1000) + (td.tv_usec / 1000);
    bs = (((float) cc * NBBY * 1000) / (float) (ms ? ms : 1))
/ NBBY;

    printf  ("round  trip  of:  %d  bytes  in  %d  ms  (%.2f
Kbytes/s)\n                                   (%.2f bits/s)\n
                              (%.2f ms/bit)",
         cc, td.tv_usec / 1000, bs / 1024,
            bs / 0.128, 128 / bs);
}


static  tvsub (tdiff, t1, t0)
register struct timeval *tdiff,
               *t1,
               *t0;
{

    tdiff -> tv_sec = t1 -> tv_sec - t0 -> tv_sec;
    tdiff -> tv_usec = t1 -> tv_usec - t0 -> tv_usec;
    if (tdiff -> tv_usec < 0)
     tdiff -> tv_sec--, tdiff -> tv_usec += 1000000;
}

#else
long times ();
```

```
static    timer (cc)
int  cc;
{
    long    ms;
    float   bs;
    long    stop,
         td,
         secs,
         msecs;
    struct tms tm;
    static long start;

    if (cc == 0) {
     start = times (&tm);
     return;
    }
    else
     stop = times (&tm);

    td = stop - start;
    secs = td / 60, msecs = (td % 60) * 1000 / 60;
    ms = (secs * 1000) +  msecs;
    bs = (((float) cc * NBBY * 1000) / (float) (ms ? ms : 1))
/ NBBY;

    printf("1-round trip of: %d bytes in %d.%02d seconds (%.2f
Kbytes/s)",
         cc, secs, msecs / 10, bs / 1024);
}
#endif
```

APPENDIX C

ABSTRACT SYNTAX NOTATION ONE (ASN.1) MODULES

```
-- SEND.RY - the IST message transmission data specification


PDU DEFINITIONS ::=

BEGIN

-- operations

               -- send pdu
send OPERATION
     ARGUMENT  Pdu
     RESULT         Pdu
     ERRORS         { noSuchUser, congested }
     ::=       0


-- errors

               -- no matching user in the database
noSuchUser ERROR
     ::=       0

               -- congestion at responder
congested ERROR
     ::=       1


-- types

               -- pdu
Pdu ::=
    [APPLICATION 1]
     IMPLICIT SEQUENCE {
          echo[0]
               IMPLICIT GraphicString
     }

END
```

-- PDU-OPS.C --

```
/* automatically generated by rosy 6.0 #3 (falcon), do not
edit! */

#include <stdio.h>
#include "PDU-ops.h"


#include "PDU-types.h"

                              /* OPERATIONS */

                              /* OPERATION send */
int  encode_PDU_Pdu (),
     decode_PDU_Pdu (),
     free_PDU_Pdu ();
int  encode_PDU_Pdu (),
     decode_PDU_Pdu (),
     free_PDU_Pdu ();

static struct RyError *errors_PDU_send[] = {
    &table_PDU_Errors[0],
    &table_PDU_Errors[1]
};


struct RyOperation table_PDU_Operations[] = {
                              /* OPERATION send */
    "send", operation_PDU_send,
     encode_PDU_send_argument,
     decode_PDU_send_argument,
     free_PDU_send_argument,
     1, encode_PDU_send_result,
        decode_PDU_send_result,
        free_PDU_send_result,
     errors_PDU_send,

    NULL
};


                              /* ERRORS */

struct RyError table_PDU_Errors[] = {
                              /* ERROR noSuchUser */
    "noSuchUser", error_PDU_noSuchUser,
     encode_PDU_noSuchUser_parameter,
     decode_PDU_noSuchUser_parameter,

     free_PDU_noSuchUser_parameter,
                              /* ERROR congested */
    "congested", error_PDU_congested,
     encode_PDU_congested_parameter,
```

```
     decode_PDU_congested_parameter,

     free_PDU_congested_parameter,
    NULL
};
```

-- PDU-STUB.C --

```
/* automatically generated by rosy 6.0 #3 (falcon), do not
edit! */

#include <stdio.h>
#include "PDU-ops.h"
#include "PDU-types.h"


#ifdef    lint

int  stub_PDU_send (sd, id, in, rfx, efx, class, roi)
int  sd,
     id,
     class;
struct type_PDU_Pdu* in;
IFP  rfx,
     efx;
struct RoSAPindication *roi;
{
     return    RyStub    (sd,    table_PDU_Operations,
operation_PDU_send, id, NULLIP,
          (caddr_t) in, rfx, efx, class, roi);
}

int  op_PDU_send (sd, in, out, rsp, roi)
int  sd;
struct type_PDU_Pdu* in;
caddr_t *out;
int     *rsp;
struct RoSAPindication *roi;
{
    return    RyOperation    (sd,    table_PDU_Operations,
operation_PDU_send,
          (caddr_t) in, out, rsp, roi);
}
#endif
```

```
-- PDU-ASN.PY --
-- automatically generated by rosy 6.0 #3 (falcon), do not
edit!

PDU DEFINITIONS ::=

BEGIN

Pdu ::=
    [APPLICATION 1]
        IMPLICIT SEQUENCE {
            echo[0]
                IMPLICIT GraphicString
        }

END
```

-- PDU-TYPE.C --

```
/* automatically generated by posy 6.0 #4 (falcon), do not
edit! */

#ifndef   _module_PDU_defined_
#define   _module_PDU_defined_

#include "psap.h"
#ifndef   PEPYPATH
#define   PEPYPATH
#endif
#include "../pepy/UNIV-types.h"


#define   type_PDU_Pdu   type_UNIV_GraphicString
#define   free_PDU_Pdu   free_UNIV_GraphicString
#endif
```

```
-- PDU-TYPE.PH --
-- automatically generated by posy 6.0 #4 (falcon), do not
edit!

PDU DEFINITIONS ::=

%{
#include <stdio.h>
#include "PDU-types.h"
%}

PREFIXES encode decode print

BEGIN


ENCODER encode

Pdu [[P struct type_PDU_Pdu *]] ::=
    [APPLICATION 1]
        IMPLICIT SEQUENCE
            %{
            %}
        {
            echo[0]
                IMPLICIT GraphicString
                [[p parm ]]
        }

DECODER decode

Pdu [[P struct type_PDU_Pdu **]] ::=
    [APPLICATION 1]
        IMPLICIT SEQUENCE
            %{
            %}
        {
            echo[0]
                IMPLICIT GraphicString
                [[p &((*parm))]]
        }

END

%{

%}
```

-- SEND.C --

```
/* automatically generated by pepy 6.0 #4 (falcon), do not
edit! */

#include "psap.h"

#define   advise   ryr_advise

void advise ();

/* Generated from module PDU */

#include <stdio.h>
#include "PDU-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;

/* ARGSUSED */

int  encode_PDU_Pdu (pe, explicit, len, buffer, parm)
register PE     *pe;
int  explicit;
integer   len;
char   *buffer;
struct type_PDU_Pdu * parm;
{
    PE    p0_z = NULLPE;
    register PE *p0 = &p0_z;

    if (((*pe) = pe_alloc (PE_CLASS_APPL, PE_FORM_CONS, 1)) ==
NULLPE) {
        advise (NULLCP, "Pdu: %s", PEPY_ERR_NOMEM);
        return NOTOK;
    }
    {
# line 20 "PDU-types.py"


    }
    (*p0) = NULLPE;

    {     /* echo */
        if (encode_UNIV_GraphicString (p0, 0, NULLINT, NULLCP,
parm ) == NOTOK)
            return NOTOK;
        (*p0) -> pe_class = PE_CLASS_CONT;
        (*p0) -> pe_id = 0;

#ifdef DEBUG
        (void) testdebug ((*p0), "echo");
```

```
#endif

    }

    if ((*p0) != NULLPE)
        if (seq_add ((*pe), (*p0), -1) == NOTOK) {
            advise (NULLCP, "Pdu %s%s", PEPY_ERR_BAD_SEQ,
                    pe_error ((*pe) -> pe_errno));
            return NOTOK;
        }

#ifdef DEBUG
    (void) testdebug ((*pe), "PDU.Pdu");
#endif


    return OK;
}
```

BIBLIOGRAPHY

[CCITT88a] Integrated Services Digital Network (ISDN). International Telegraph and Telephone Consultative Committee, November, 1988. Recommendations 1.430 and 1.431.

[CCITT88b] Integrated Services Digital Network (ISDN). International Telegraph and Telephone Consultative Committee, November, 1988. Recommendation Q.921.

[CCITT88c] Integrated Services Digital Network (ISDN). International Telegraph and Telephone Consultative Committee, November, 1988. Recommendation Q.931.

[CCITT88d] Message Handling: System and Service Overview. International Telegraph and Telephone Consultative Committee, November, 1988. Recommendation X.400.

[CCITT88e] The Directory - Overview of Concepts, Models, and Services. International Telegraph and Telephone Consultative Committee, November, 1988. Recommendation X.500.

[CENG91] Cengeloglu, Y.; Ng, H.K.; and Pourparviz, G., "Investigating the Use fo the Distributed Interactive Simulation Protocol Standards for Real Time Simulation Networking", Institute for Simulation and Training, Publication Number IST-TR-91-4, February 15, 1991.

[FIPS87] Federal Information Processing Standards, "GOSIP Draft", National Bureau of Standards Federal Information Processing Standards Publication (FIPS PUB), National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161, 1987.

[GAUD89] Gaudette, P., "A Tutorial on ASN.1", U.S. Department of Commerce, National Institute of Standards and Technology, Technical Report NCSL/SNA-89/12, May 1989.

[ISI81a] Information Sciences Institute, University of Southern California, Internet Protocol. Request for Comment 791, DDN Network Information Center, SRI International, September 1981.

[ISI81b] Information Sciences Institute, University of Southern California, Transmission Control Protocol. Request for Comment 793, DDN Network Information Center, SRI International, September 1981.

[ISO84] Information Processing Systems - Open Systems Interconnection - Basic Reference Model. International Organization for Standardization and International Electrotechnical Committee, 1984. International Standard 7498.

[ISO86] Information Processing Systems - Data Communications - High-Level Data Link Control Procedures - Description of the X.25 LAPB-Compatible DTE Data Link Procedures. International Organization for Standardization and International Electrotechnical Committee, 1986. International Standard 7776.

[ISO87a] Information Processing Systems - Local Area Networks - Logical Link Control. International Organization for Standardization and International Electrotechnical Committee, 1987. Draft International Standard 8802-2.

[ISO87b] Information Processing Systems - Data Communications - Network Service Definition. International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8343.

[ISO87c] Information Processing Systems - Open Systems Interconnection - Protocol for Providing the Connectionless-Mode Transport Service. International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8602.

[ISO87d] Information Processing Systems - Open Systems Interconnection - Basic Connection Oriented Session Protocol Specification. International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8327.

[ISO88a] Information Processing Systems - Data Communications - Protocol for Providing the Connectionless-Mode Network Service. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8473.

[ISO88b] Information Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8073.

[ISO88c] Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Protocol Specification. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8823.

[ISO88d] Information Processing Systems - Open Systems Interconnection - The Directory. International Organization for Standardization and International Electrotechnical Committee, 1988. Draft International Standard 9594.

[ISO88e] Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8571.

[ISO89a] Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 8824 with Draft Addendum 1.

[ISO89b] Information Processing Systems - Open Systems Interconnection - Application Layer Structure. International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 9545.

[ISO89c] Information Processing Systems - Local Area Networks - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 8802-3.

[ISO89d] Information Processing Systems - Local Area Networks - Part 4: Token-Passing Bus Access Method and Physical Layer Specifications. International Organization for Standardization and International Electrotechnical Committee, 1989. Draft International Standard 8802-4.

[ISO89e] Information Processing Systems - Local Area Networks - Part 5: Token Ring Access Method and Physical Layer Specifications. International Organization for Standardization and International Electrotechnical Committee, 1989. Draft International Standard 8802-5.

[ISO89f] Information Processing Systems - Fibre Distributed Data Interface (FDDI). International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 9314.

[ISO89g] Information Processing Systems - Data Communications - X.25 Packet Level Protocol for Data Terminal Equipment. International Organization for Standardization and International Electrotechnical Committee, 1989. Draft International Standard 8208.

[ISO89h] Information Processing Systems - Open Systems Interconnection - Virtual Terminal Protocol. International Organization for Standardization and International Electrotechnical Committee, 1989. Draft International Standard 9041.

[LOPE91] Loper, M.L.; Shen, D.; Thompson, J.; and Williams, H., "ISODE System Installation Manual", Institute for Simulation and Training, Internal Report, January, 1991.

[MCDO90] McDonald, B.L.; Pinon, C.; Glasgow, R.; and Danisas, K., "Rationale Document Protocol Data Units for Distributed Interactive Simulation", Institute for Simulation and Training, Publication Number IST-PD-90-1, June 15, 1990.

[PAIG90] Paige, LT. J.L., "SAFENET - A Navy Approach to Computer Programming", IEEE 15th Conference on Local Computer Networks, Minneapolis, Minn., October 1-3, 1990.

[PISC86] Piscitello, D.M.; Weissberger, A.J.; Stein, S.A.; and Chapin, A.L., "Internetworking in an OSI Environment", Data Communications, May 1986, pp. 118-136.

[ROSE87] Rose, M.T., "ISODE: Horizontal Integration in Networking", ConneXions, The Interoperability Report, vol. 1, no. 1, May 1987.

[ROSE88] Rose, M.T., "Building Distributed Applications in an OSI Framework", ConneXions, The Interoperability Report, vol. 2, no. 3, March 1988.

[ROSE90a] Rose, M.T., "The Open Book: A Practical Prespective on OSI", Prentice Hall, Englewood Cliffs, N.J., 1990.

[ROSE90b] Rose, M.T., "The ISO Development Environment: User's Manual, Volumes 1-5", Performance Systems International, Inc., January 14, 1990.

[SHEN91] Shen, D., "ISODE Network Performance Experimental Analysis", Institute for Simulation and Training, Internal Report, February 15, 1991.

[STAL87] Stallings, W., Local Networks, Second Edition, Macmillan Publishing Company, N.Y., 1987.

[TANE88] Tanenbaum, A.S., Computer Networks, Second Edition, Prentice Hall, Englewood Cliffs, N.J., 1988.

0000007