ABSTRACTION AND SPECIFICATION OF LOCAL AREA NETWORKS

By

Chang-Hyun Jo

Bachelor of Economics

Sung Kyun Kwan University

Seoul, Korea

1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1988

ABSTRACTION AND SPECIFICATION

OF LOCAL AREA NETWORKS

Thesis Approved:

Donald D Fisher
Thesis Advisor

J P Chandler

Thomas C. Collins

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express sincere appreciation to my advisor, Dr. D. D. Fisher for his warm encouragement and helpful advisement throughout my graduate study. I extend sincere thanks to committee member, Dr. K. M. George who provided valuable suggestions and comments on this thesis. I would also like to express my gratitude to Dr. J. P. Chandler for his invaluable support and advisement. The helpful comments of Dr. C. Hutchens on LANs are also sincerely appreciated. I am also grateful to Dr. M. Samadzadeh for his warm and constant guidance.

Throughout my life, my parents provided constant support and understanding without which this thesis could never have been done.

My deepest appreciation is extended to my wife, Ae-Kyung and to my sons, Hyun-Soo and Jin-Soo for their patience and love.

Chang-Hyun Jo
OSU
July, 1988

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

An abstraction is a methodology for program construction. A large program problem can be decomposed into a number of small programs called modules. Abstraction is a way to do decomposition. Using an abstraction methodology, complicated objects can be simplified by decomposing the original objects into sub-modules until we can abstract each module as a specific function. We use here three kinds of abstractions [Liskov 1986]. Data types are the kinds of data such as integer, real, logical and character. Data objects are the elements in the set of data. Data structures consist of the operations applicable to the data objects and the data objects themselves. An abstract data type is a the mathematical model with a collection of operations defined on the data objects [Cleaveland 1986] [Guttag 1980]. Data abstraction allows us to defer decisions about a data structure during implementation until the data structure is fully specified. A procedural abstraction is a mapping from the arguments to the results with possible modification of some of the arguments. A behavioral or a functional abstraction is a description of the behaviors of modules when they are

invoked. An abstraction is intangible. We have some difficulties to understand what an abstraction is, without any description. The specification is this description. A specification is the only tangible record of an abstraction.

A local area network(LAN) is a communications network that provides interconnection of a variety of data communicating devices within a small area [Stallings 1987]. LAN makes possible a form of computing that is distributed in several ways and provides several communications media and channels for data, images, and voice communication.

Specification for a computer network protocol is a formal description of its service function. Recently, a few papers introduced the specification, testing and verification techniques for distributed systems and communication protocols [Gehani 1986] [Lam 1984] [Sunshine 1981]. Those are formally described by Petri Net models, attributed grammars and some dedicated languages. However, those are not machine-readable, hence they cannot be normally used as input to design automation or simulation of network systems.

In this paper, we use the abstraction methodology to describe local area networks. This work includes an abstract model for a LAN and the specification of LAN communication protocols using a specification template developed based on the LAN abstract model. The advantage of such an approach is that the specification could be

machine-readable. In LAN abstraction, we discuss the functional properties of a data structure and its operations, then specify it in abstract template constructs. This paper includes an abstract specification of the functional capabilities of its physical components, the data structure, the nature of control and information flow between components in local area networks. Chapter 2 deals with general concepts for abstraction and specification. Chapter 3 provides an overview of local area network. Chapter 4 introduces LAN abstract model and the formal model of LAN communication and objects. Chapter 5 presents the specification template and its examples. Further applications of abstract specification to OSI model LAN and DOD model LAN are shown in Chapter 6. This paper ends with the conclusions in Chapter 7.

CHAPTER II

ABSTRACT SPECIFICATION

2.1 Abstraction

An abstraction provides a systematic tool used in classifying and solving problems. Abstraction is a mathematical modeling of the system. Discussion about abstraction is also becoming abstract. To avoid this abstraction, a formal or an informal model is used. The functions of the system are fixed and are already defined. The complexity of the functions, however, is beyond comprehension of the users who manipulate the system functions and even the implementors who facilitate the system. Using abstraction, complexity is reduced and the problem is generalized in the process of reducing the redundancy and omitting irrelevant details and reconstructing ambiguous flow. A good abstraction is well balanced between the level of the abstraction and its complexity. It must also have a good balance between the machine dependent modeling and the machine independent modeling. Some abstractions are portable if they can be implemented on several independent systems without losing compatibility. The harmonious abstraction between the

logical environment and the physical environment is a very difficult problem.

2.2 Specification

Specification for a computer network protocol is a formal description of its service function. A number of communication protocol specification techniques are as follows [Gehani 1986] [Guttag 1978] [Sunshine 1981]:

1) abstract machine model,

2) formal languages model,

3) Petri Nets model,

4) abstract data types.

Not only can abstract machine automata diagram easily network structures [Bochmann 1978], but Petri Nets also describe well abstract concurrent systems [Merlin 1979] [Keller 1976]. However, those are not machine-readable, hence they cannot be normally used as input to design automation or simulation of network systems. Here we show how to use abstraction methodologies for the specification of communication systems in terms of local area networks which can be a machine-readable input for network software design with some proper modification.

2.3 Abstract Data Types

2.3.1 Data Types

Data types are specific programming language

constructs used to describe and define data structures. A type is a set of values. A data type is a set of values and a set of operations on the values. Types specify how to interpret the values. The relationships between the different types depend on the point of view. Every object belongs to exactly one type, however, objects of one type can represent objects of other types. Types are helpful to understand objects and to detect errors. (Type checking is one of the most powerful error detection capabilities of a compiler.)

2.3.2 Abstract Data Types

Abstract data types(ADT) were probably the most important advance in programming languages during the 1970's [Cleaveland 1986] [Guttag 1980]. The major conceptual idea of ADT is to separate the use of a type from the representation and implementation of a type. The use of a type should depend on the set of values and operations. It should not depend on either its representation or its implementation. The use of data types can be specified by the syntax and semantics. The languages provide a rich syntax for expressing data types and ADT. An abstract data type may have various implementations. Interchangeability of implementations is another merit of abstract data types. Since it is not necessary that the application programmer need to know the encapsulated algorithms, it is only necessary for him to learn how to

specify the interface parameters which is generally a much simpler task. This encapsulation is a major merit of abstract data types.

2.4 The Characteristics of Abstraction

Abstraction provides a way of organizing and designing programs that are both more reliable and easier to change. We cannot call just a collection of related procedures a data abstraction. A merit for using abstraction is the inter-changeability of implementation. If a more efficient implementation is found it can readily be substituted for the older implementation. Three kinds of abstractions are used for an abstraction here. Those are data abstraction, procedural abstraction and behavioral abstraction. Liskov [1986] describes well about data abstraction and procedural abstraction. But how does behavioral abstraction differ from those two kinds of abstraction? Behavioral abstraction is a high level language like description of an invoked module which consists of data abstraction and procedural abstraction. When such a module is invoked, necessary operations are held spontaneously. If the abstract module is invoked as a main module, such an abstraction is absolutely necessary. An abstraction consists of two parts. The first part is the specification and the second one is the implementation. In this paper, we introduce the specification scheme with an abstraction model of a LAN, and then we show how this specification technique is used

for the implementation.

2.4.1 Data Abstraction

Data abstraction consists of a set of objects and a set of operations characterizing the behavior of the objects. Data abstraction separates the use of a data type from the implementation of a data type.

2.4.2 Procedural Abstraction

A procedure provides a transformation from input arguments to output arguments. Procedural abstraction allows us to explain data objects in terms of input and output parameters.

2.4.3 Behavioral Abstraction

Behavioral abstraction specifies the behaviors of invoked modules in which both data abstraction and procedural abstraction are composed. When each module is invoked by calling its quantifiers, its functional behavior is defined by its behavioral abstraction.

2.5 Abstract Specifications

A specification says exactly what a data type is and how its operation behaves. This information enables a programmer to implement the data type and the operation, and it enables users to use the data type and the operations. Precision and communication are the two most

important qualities of specifications to understand and to interpret; specifications must be precise and unambiguous. Specifications are used to communicate between the user and the implementor. Natural language is easier to read but not precise, and most formal languages are precise but difficult to read. Precision and readability are conflicting goals of abstract specifications. Abstract specification provides a means for defining abstractions. There are are several methods for defining abstractions including the following [Cleaveland 1986].

1) Algorithmic specification
2) Operational specification
3) Logical specification
4) Functional specification
5) Algebraic specification

Now let us illustrate how to define the abstractions in terms of each specification.

2.5.1 Algorithmic Specification

The algorithmic specification can define the data types and the operations as a plain English sentence which is a brief and precise description.

```
Operations
SENDMSG(destin_proc_id) : send a message to the system buffer, and
                          change the state.
Exceptions
sendmsg : when dead state, no operation occurs and output error
          message.
```

2.5.2 Operational Specification

The operational specification is the most high_level language like implementation.

```
Datatypes
system_msg_buffer : array[1..max_que_size] of character;
Operations
SEND_MSG(proc_id) : system_msg_buffer[msg_no] = sending_msg;
```

2.5.3 Logical Specification

Logical specifications use input and output assertions written in predicative calculus to describe the conditions before and after execution of statements and procedures.

```
Parameter : destin_proc_id;
Invariant Assertion : state ^= dead_state;
Output Assertions :
SEND_MSG : (system_msg_queue= sending_msg)^(status =
ready_to_send);
```

2.5.4 Functional Specification

Functional specifications describe the output in terms of input using mathematical constructions such as sets, functions, and sequences for representing objects traditionally.

```
system_msg_buffer = set of messages;
SENDMSG :  message x system_msg_buffer -> system_msg_buffer;
```

2.5.5 Algebraic Specification

The algebraic approach views data types as algebra,and to specify a type the axioms that describe such an algebra can be used.

```
Axioms
RECEIVE_MSG (STATUS_CHK (SYSTEM_MSG_QUE (STATUS_CHK (SEND_MSG))..)
= message;
```

2.6 Abstract Model

In the context of a formal model, the abstraction for a certain problem solving can be done as illustrated in Figure 1.

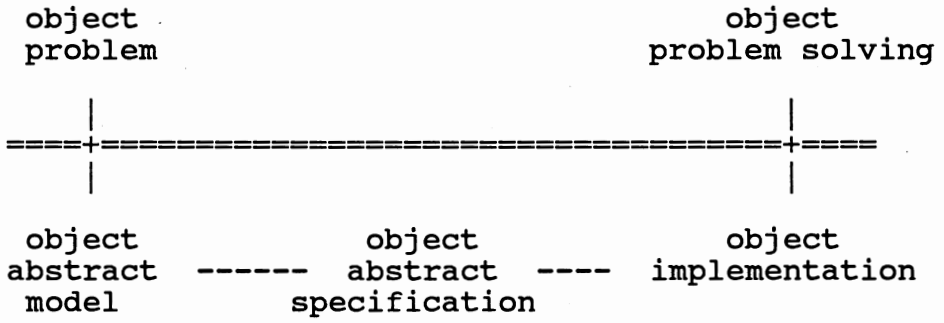


Figure 1. Abstract Modeling

Properly combining all of the above methods to specify abstractions, we give an abstract model of LAN for its communication and its objects in Chapter 4 after we review some LAN models in Chapter 3.

2.7 Abstract Specification

The abstract specification describes how all the operations work. It can be a procedural abstraction and it

may be represented by a programming language. An abstract specification may be implemented in various ways. The inter-changeability of certain implementations and the encapsulation of data objects are the major motivations of the abstraction. The abstract specification of LAN systems are shown in Chapter 5 and 6.

CHAPTER III

LOCAL AREA NETWORKS

3.1 Purpose of LAN

A local area network is a communications network that provides interconnection of a variety of data communicating devices within a small area [Stallings 1987]. The main purpose of a LAN is for distributed computing and resource sharing among associated devices. LAN makes possible a form of computing that is distributed in several ways and provides several communications media and channels for data, images, and voice communication [Figure 2].

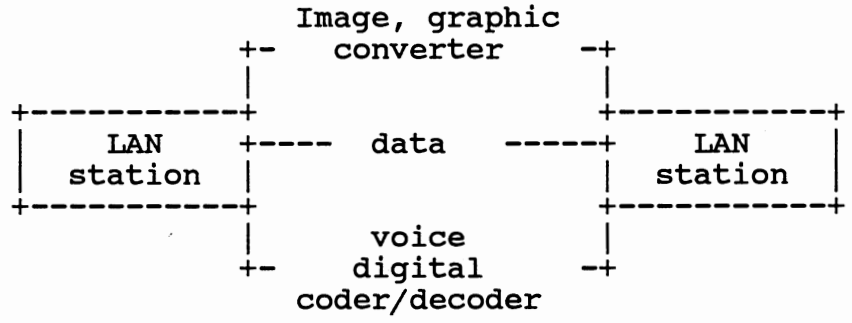


Figure 2. Data Communication Model

3.2 LAN Standards

In general LAN computers from different vendors are different from each other, thus it is desirable to have a set of standards for LAN. All specifications for LAN in this paper try to follow the OSI(Open Systems Interconnection) Reference Model standardized by International Organization for Standardization(ISO) and IEEE 802 LAN standards. The OSI Reference Model [ISO 1981] contains the following seven layers [Figure 3].

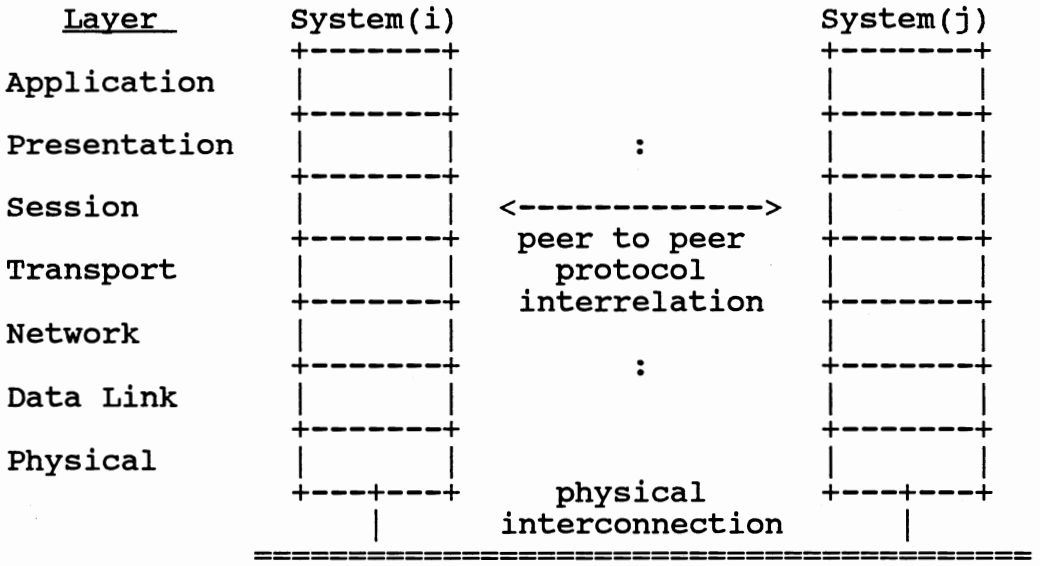


Figure 3. Seven-Layer OSI Reference Model

Protocols are the rules that communicating processes follow when they exchange messages and control information.

The protocols specify the format of the data and control information and give the procedures that the sender and receiver follow. Protocols are characterized by a cost and a reliability.

3.3 LAN Topologies

Topology refers to the way in which the end points, or stations, attached to the network are interconnected. The common LAN topologies are the following [Figure 4].

1) Ring network
2) Bus network
3) Star network
4) Tree network
5) Mesh network

Among those topologies, the standardizations of LAN focus on the bus, ring and star topologies. Bus networks use the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) scheme, while the ring network uses token passing scheme. A transmission scheme says how to control the transmission of traffic in a network. Two transmission techniques are used for the bus topology: baseband and broadband. Baseband uses digital signaling on twisted pair wire or coaxial cable. Baseband transmission is bidirectional. A broadband scheme allows many different stations to have messages in the network at the same time. Broadband uses analog signaling on coaxial cable. It covers much greater distance than baseband. But unlike baseband,

broadband is unidirectional transmission scheme. The ring topology is major alternative of the bus topology. The ring consists of a number of repeaters connected on unidirectional transmission links. Data are transmitted in packets and transferred sequentially around a ring.

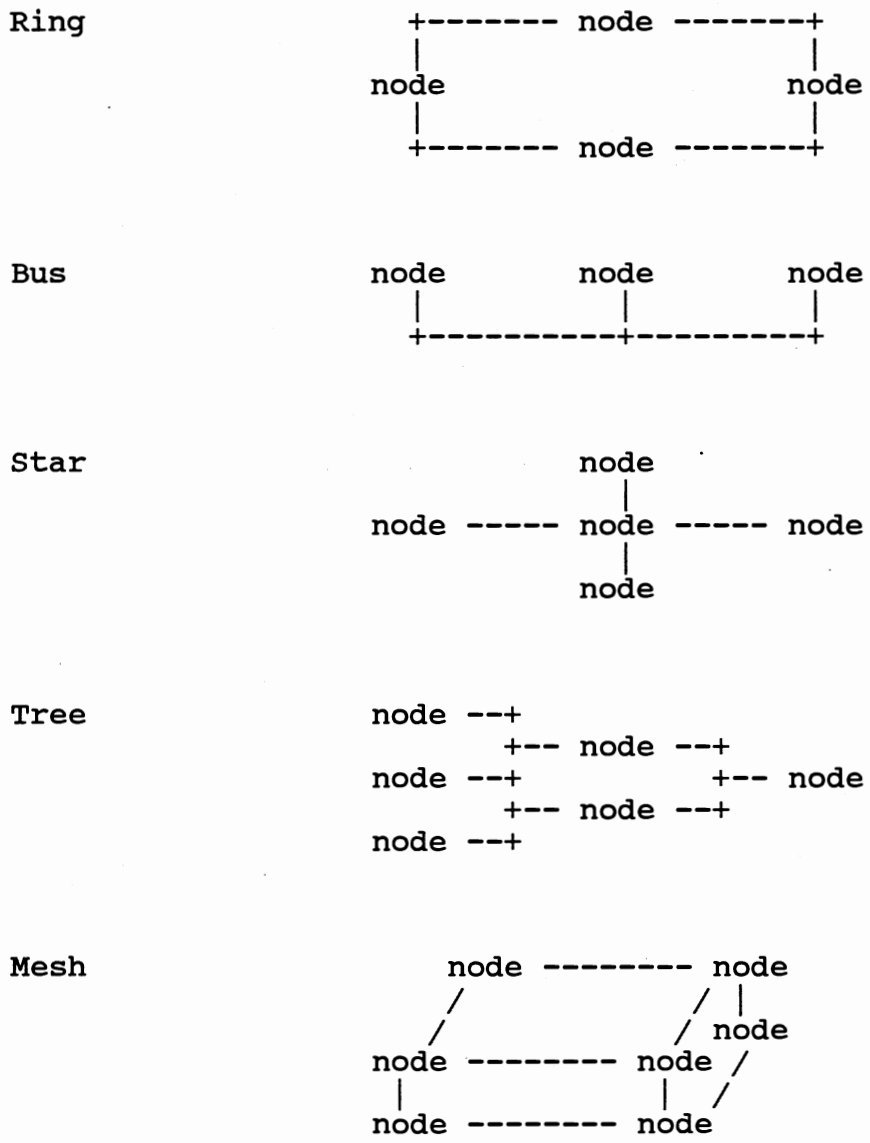


Figure 4. Network Topologies

3.3.1 The Ring Topology

A ring network consists of nodes that are connected by transmission links to form a series of nodes in a link(loop).

3.3.2 The Bus Topology

The nodes are attached to a bus that provides a bidirectional transmission facility.

3.3.3 The Star Topology

All nodes are connected to a centralized controller in which nodes exchange data through the central node.

3.4 LAN Transmission Media

The transmission medium is the physical path between transmitter and receiver. The media used for local area networks are twisted pair, coaxial cable and optical fiber. A twisted pair is used for normal voice communication. Twisted pair can be a low-cost solution for a small network, but the transmission distance for signals on twisted pair is relatively short without the aid of repeaters. A coaxial cable can be used for transmitting both analog video and digital data signals. Coaxial cable has a much wider bandwidth. The shield on a coaxial cable is used to reduce noise intrusion. Since it is easy to install and it supplies a higher transmission speed, it is

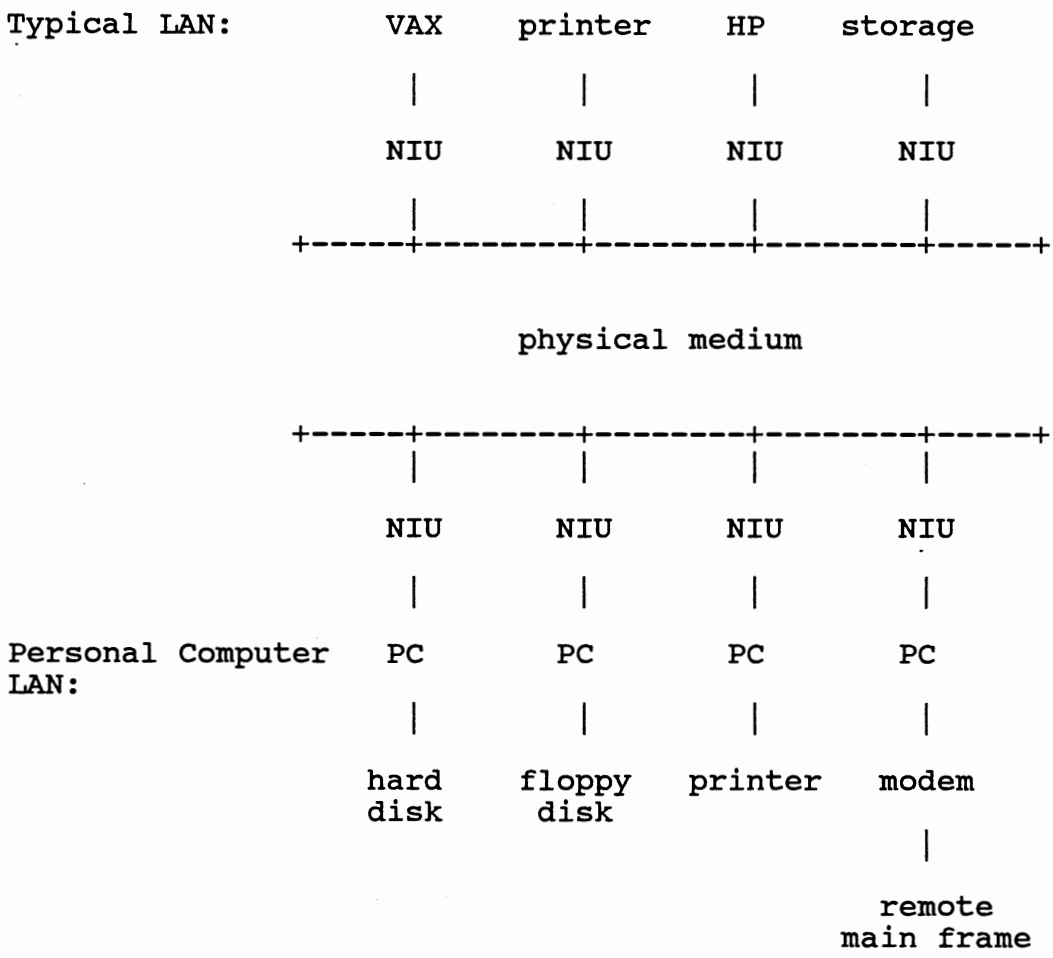
widely used in Cable TV industry. Optical fiber media is now used in LAN systems with several advantages over other media. It has lower power loss, and greater bandwidth potential. But optical fiber is more expensive than other transmission medium.

3.5 LAN Architecture

Computer networks consist of the host and the subnets. The subnets include the switching elements and the transmission line. Most computer networks are organized in layers or levels, each built upon its predecessor level. A computer network consists of two layers, a physical communication level and a virtual communication level. The virtual communication level is divided into several layers like the OSI model. Each layer performs a certain task and provides services for the next layer. The set of interaction rules between the layers is called a protocol. The functions performed by each layer are the following.

1) Initiation of entity interaction
2) Data transmission
3) Data manipulation
4) Information control
5) Interaction termination

Figure 5 shows a typical LAN architecture.



NIU: Network Interface Unit

Figure 5. LAN Architecture

3.6 Internetwork Communication

Sometimes it is necessary that a LAN access another network like a nation-wide network as well as another LAN. A network interface unit(NIU) implements and provides an interface capability for interconnection between nodes in different LANs. It acts as a communication controller. A network interface unit can support several communication terminals. An intermediate node which provides an interface between a LAN and another LAN or a WAN(Wide Area Network) for communication is called a bridge or a gateway. A bridge consists of two network interface units linked together. A bridge serves as a connector to connect separate homogeneous networks [Figure 6]. A gateway is also a device for connecting two systems that use different protocols [Figure 7]. It behaves as a protocol converter. Thus a bridge may not change the format of protocol content while a gateway may modify the protocol format. An example of an inter-network data packet is given in Figure 8. The inter-network protocol has responsibility for multiplexing and demultiplexing a data packet. The well known inter-network protocol is Internet Protocol(IP) which was developed by ISO and by DOD and it is used with Transmission Control Protocol(TCP). IP provides a connectionless data transfer service to other IP users.

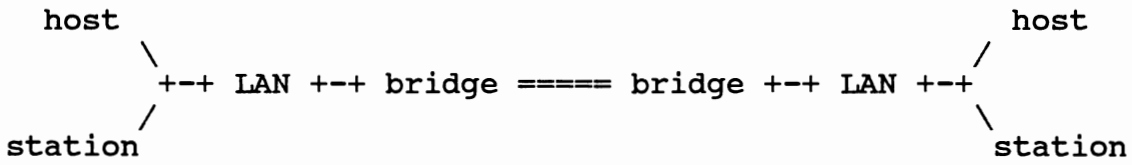


Figure 6. Homogeneous Inter-network

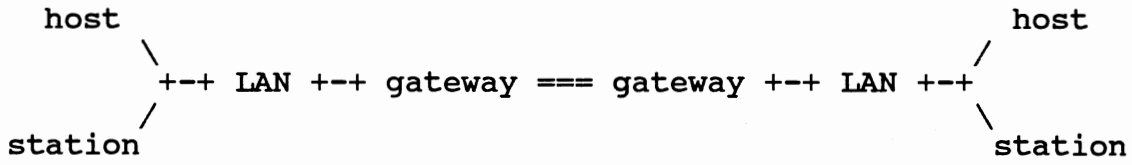
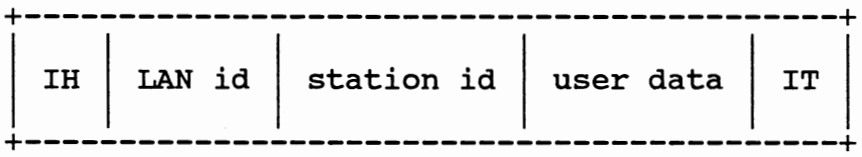


Figure 7. Heterogeneous Inter-network



IH : inter-network data packet header
IT : inter-network data packet trailer

Figure 8. Inter-Network Data Packet Format

CHAPTER IV

ABSTRACTION OF LOCAL AREA NETWORKS

Using protocols, two users can communicate with each other from different LAN stations [Figure 9]. Protocols involve interactions with users or programs in order to get certain functions performed. How these functions are actually performed by the protocol is not really of concern; only the end result matters. The users regard the protocols as black boxes. Each protocol level makes use of the services provided by the next lower level. Data transmission is accomplished by passing data between adjacent lsyers. We call this an inter-layer transmission. In this chapter, we discuss an abstract modeling of LAN communication using such an inter-layer data transmission. A formal model of LAN data objects are also described.

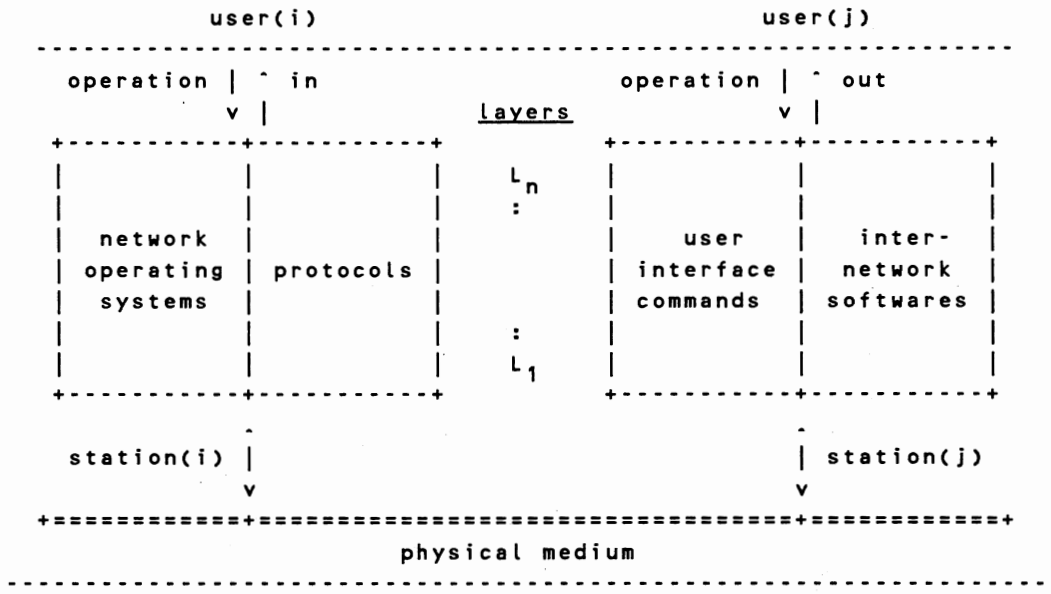


Figure 9. Communication System

4.1 Abstract Model of LAN

LAN abstract model consists of an abstraction of LAN communication and a formal model for LAN objects. A LAN abstraction can be modelled as illustrated in Figure 10. In this section we describe basic concepts for LAN abstact modeling. Then the formal model of abstract LAN communication in terms of inter-layer transmission and abstract modeling for data object used in a LAN are followed in the next sections.

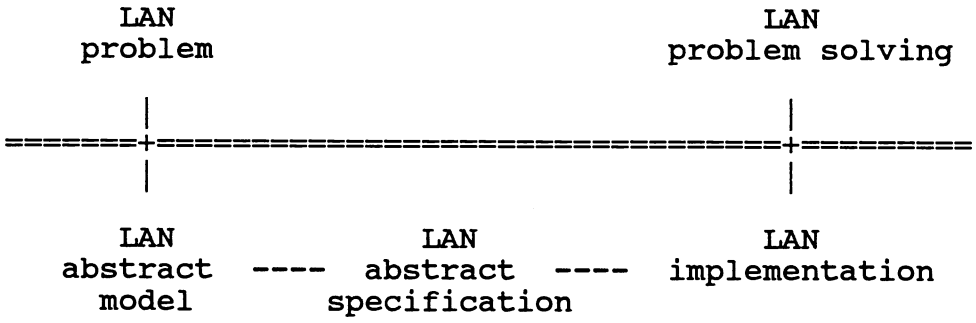


Figure 10. LAN Abstract Modeling

In the layer set of L, there are upward and downward inter-layer communication relations. Suppose L is an ordered set of layers, and let R represent an inter-layer communication relation on the set L. Thus R is

$$R = \{ (L_i, L_j) \mid 1 <= i,j <= n \}$$

where $L_i, L_j \in L$,

$1 < .. i, j .. < n$, $j = i \pm 1$,

n is the number of layers,

and the meaning of the ordered pair(L_i, L_j) is that layer L_i communicates with layer L_j.

The directed graph representation of the bidirectional inter-layer communication relations $R(L_i,L_j)$ is the following [Figure 11].

$$L_n \underset{\longrightarrow}{\longleftarrow} L_{n-1} \underset{\longrightarrow}{\longleftarrow} \cdots\cdots \underset{\longrightarrow}{\longleftarrow} L_2 \underset{\longrightarrow}{\longleftarrow} L_1$$

Figure 11. Inter-Layer Communication Relation Graph

For a particular local area network, the matrix representation M of these relations is

$M(i,j) = 1$ (true) if $R(i,j) \in R$ or

0 (false) if $R(i,j) \notin R$

for all $L_i, L_j \in L$(an ordered set of layers), and (Li)R(Lj) <=> (Lj)R(Li) since the inter-layer communication relation shows the symmetric property in the matrix [Figure 12].

	L1	L2	L3	L4	L5	L6	L7	....
L1	0	1	0	0	0	0	0	
L2	1	0	1	0	0	0	0	
L3	0	1	0	1	0	0	0	
L4	0	0	1	0	1	0	0	
L5	0	0	0	1	0	1	0	
L6	0	0	0	0	1	0	1	
L7	0	0	0	0	0	1	0	
:								
:								

Figure 12. Inter-Layer Communication Relation Matrix

In LAN, the services are provided by interfaces between adjacent layers. To transfer data or control information, it is transferred to the next layer by invocation of the services of the next layer with given primitives. Transmitted data reaches the physical layer, and is delivered to a destination entity through the transmission medium. Upward service invocations are then invoked until the data finally arrives at the user interface layer in the destination entity. Now let us define the formal model of LAN communication in terms of such inter-layer service communication.

4.2 The Formal Model of LAN Communication

The formal model of LAN communication is defined in terms of inter-layer service communication described above.

We need the following definitions:

1) L represents an ordered set of layers
   { L_n, L_{n-1}, ..., L_j, L_i, ..., L_1 }
   where L_n is the highest layer and L_1 is the lowest layer;
2) || represents inter-layer communication-flow concatenation
   (left-associative and non-commutative binary operator);
3) $$ represents inter-node physical connection;
4) ## represents inter-network communication connection;

5) %% represents inter-network converter
{ gateway, bridge };

6) R = { (L_i, L_j) | 1 <= i,j <= n and i = j $\pm$ 1 }
inter-layer communication relationship between layers L_i,L_j.
The meaning of the ordered pair (L_i, L_j) is that layer L_i communicates with layer L_j;

7) D = { downward transmission, upward transmission };

8) An ordered pair (r,d) where r $\in$ R and d $\in$ D is called a transmission.

Definition 2.1

Communication in a homogeneous LAN can be represented by the string:

$((L_n, L_{n-1}), R)$ || $((L_{n-1}, L_{n-2}), R)$ || ..... ||
$((L_j, L_i), R)$ || ..... || $((L_3, L_2), R)$ ||
$((L_2, L_1), R)$ $$ $((L_1, L_2), R)$ || $((L_2, L_3), R)$
|| ..... || $((L_{n-1}, L_n), R)$

Communication in a heterogeneous LAN can be represented by the string:

$((L_n, L_{n-1}), R)$ || $((L_{n-1}, L_{n-2}), R)$ || ..... ||
$((L_j, L_i), R)$ || ..... || $((L_3, L_2), R)$ ||
$((L_2, L_1), R)$ ## %% ## $((L_1, L_2), R)$ ||
$((L_2, L_3), R)$ || ..... || $((L_{m-1}, L_m), R)$

where (i) n,m > 0, n and m need not be same and

(ii) %%(inter-network converter) may have some protocol layers.

Definition 2.2

Let T : R x T -> { Ts, Te, Tr }

T = Ts if transmission is successful,

Te if an error occurs,

Tr if re-transmission is needed.

(Tr eventually is either Ts of Te.)

with the property, T distributes over '||' in a communication.

With the above definition of T we have the following:

if C = s1 || s2 || .....

where si = $((L_i, L_{i\pm1}), d)$,

then T(C) = T(S1) || T(S2) || ..... .

Definition 2.3

A communication consists of a series of transmissions, such a communication can be defined by the concatenation of inter-layer transmissions:

The operation '||' can be defined for values of T as follows:

Ts || Ts = Ts,

Ts || Tr = Tr,

Ts || Te = Te,

undefined otherwise

where left-associative and non-commutative

binary operator '||' means inter-layer communication-flow concatenation.

We deduce the result of whole communication from substitution of partial inter-layer transmissions. If we have a communication of

Ts || Tr || T(Si) || T(Sj) .... .

By substitution, it would be

Tr || T(Si) || T(Sj) .... .

If Tr is eventually Te, then the string is

Te || T(Si) || T(Sj) .... .

Here, we do not have any definition for (Te || ...), it means such a transmission is out of the question; Note that the transmission to the next adjacent layers is impossible after the transmission error between the certain layers. Non-definition stands for an impossible transmission or a transmission error. The result of the above transmission is

Te ....

which indicates a transmission error.

What is the valid communication in terms of the data transmission T? A communication is successful if and only if a transmission stream reduced to Ts. If we have the transmissions

Ts || Tr || T(Si) || T(Sj) ....

in a certain communication, such transmissions can be reduced by substitution of transmission definitions.

Tr || T(Si) || T(Sj) ....

:

Such a communication may result in

Ts || T(Si) || T(Sj) ....

:

Ts

which means a valid communication with the re-transmission during the data transmissions.

So far, we have defined an abstract LAN model using the mathematical modeling methodologies. In the next section we discuss the formal model of LAN objects.

4.3 The Formal Model of LAN Objects

Now we introduce the formal model of LAN objects [Figure 13]. Data packet, file, message buffer and status vector are the LAN objects. The model is based on the elements of a LAN such as communication layers, topology, transmission medium, data structure of LAN objects and communication details.

In this LAN formal model, the topologies and media used in a LAN can be defined in **'TOPOLOGY'** and **'MEDIUM'**. **'DATA STRUCTURE'** defines the data and data type used in a LAN structure. **'COMMUNICATION LAYERS'** shows a LAN structure which consists of several layers. Transmission scheme, the functions of layers, and the primitives used with the layers can be described in **'COMMUNICATION DETAIL'**. In Figure 13, '->' and '<-' mean downward and upward transmissions respectively.

ABSTRACT_LAN

TOPOLOGY => topology_list
topology_list => { ring, bus, star }

MEDIUM => medium_list
medium_list => { coaxial cable, fiber optic cable, twisted pairs }

DATA STRUCTURE => data_list
data_list => data : data_type

COMMUNICATION LAYERS => layer_list
layer_list => { L_n, L_{n-1}, ... , L_j, L_i, ... , L_1 }
L_n : highest layer
L_i, L_j : middle layers
L_1 : physical layer

COMMUNICATION DETAIL => communication_flow
communication_flow => { (L_n, L_{n-1}, { ->, <- }),
(.., .., { ->, <- }),
(L_j, L_i, { ->, <- }),
(.., .., { ->, <- }),
(L_2, L_1, { ->, <- }) }

END ABSTRACT_LAN

Figure 13. Formal Model of LAN

Using this model, we introduce the LAN abstract specification template in which the LAN objects and their functions can be described [Figure 14 in Chapter 5]. The relationship between the LAN formal model and the LAN abstract specification template is that the LAN specification template is an instance of the LAN formal model. LAN specification template is high level language like. Each object or protocol can be specified using this template. Each specified template can be implemented in a module. Topology and medium in a formal model for a LAN can

be specified in this template. To specify communication detail, necessary interface operations are modulized in a template.

CHAPTER V

SPECIFICATION FOR LAN ABSTRACTION

Now we combine properly abstract specifications which have been shown in Chapter 2, LAN in Chapter 3, and LAN abstraction and formal model of LAN in Chapter 4.

5.1 LAN Abstract Specification Template

Using the LAN formal model, we introduce the high-level language like LAN abstraction template for specifying LAN objects. Figure 14 shows a template for an abstract specification which is associated with a LAN. There are seven dimensions in the space of an abstract specification for a LAN. A dimension of the network abstraction space relates to the distinction between structures and behaviors. The header **'ABSTRACT_LAN'** introduces an abstract object name and **'OVERVIEW'** defines an overall description and describes the operations with the objects. **'TOPOLOGY'** shows all the possible topologies in the network structure and **'MEDIUM'** specifies possible transmission media. **'DATA STRUCTURE'** shows all the data objects and their types used in the module. **'INTERFACE'** lists all the operations used for the interface in the object. **'OPERATION'** specifies each operation involved in the abstract object. **'BEHAVIOR'**

describes the inherited functional behaviors of the modules when they are invoked. It is similar to the high-level language descriptions. A template ends with '**END ABSTRACT_LAN**' trailer. Examples are shown in the next sections.

```
ABSTRACT_LAN  object_name

     OVERVIEW
                  abstract description of this module

     TOPOLOGY
                  possible topologies
                  (ring | bus | tree | star)

     MEDIUM
                  possible media
                  (twisted pair | coaxial cable |
                   optical fiber)

     DATA STRUCTURE
                  data objects : data types

     INTERFACE
                  interface operations

     OPERATION
                  operation : procedural description

     BEHAVIOR
                  functional abstraction of the object
                  behavior

END ABSTRACT_LAN
```

Figure 14. LAN Abstract Specification Template

5.2 LAN Abstract Specification Examples

Any object in a LAN can be specified in the LAN abstract specification template. The following example of an abstract specification for a subset of object modules to implement a LAN includes abstract modules named osu_lan, status, file, message, batch_job, pdu and buffer. Figure 15 shows the relations among these object modules by constructing a graph.

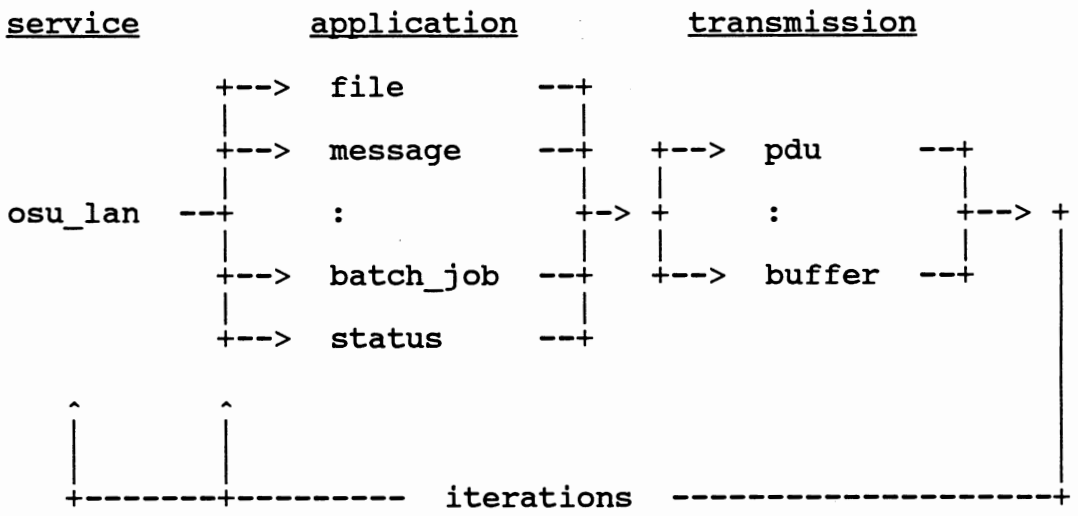


Figure 15. Relations among Object Modules

In Figure 15, 'A -> B' means that the module A may call module B to request services or applications.

In this example 'osu_lan' LAN abstract module can be interpreted to be a main module for implementation [Figure 16].

Actually, the application of abstract specification template to the logical or physical model leads to some problems. Not all the spaces can be specified well all the time. Then we left such a space as not applicable.

Also sometimes, we need access to LAN system status information. Then the module 'status' can be used [Figure 17].

The application layer of LAN supports file transfer [Figure 18], message transmission [Figure 19], and distributed batch jobs [Figure 20].

As we mentioned, there are seven layers in the OSI Reference Model. There is no direct communication between the peer layers except for the physical layer. Communication between applications need services of the lower layers. In the most common way, the data transmission between two entities is accomplished by both encapsulation and segmentation. A header is appended to or detached from the data as it is passed by each layer. The application layer of the receiving entity regains the same data type that the application layer of sending entity sent. This formatted data is called PDU(protocol data unit). PDU can be specified as shown in Figure 21.

In each layer and especially in the physical layer, the data to be transmitted may be delayed due to

synchronization. A buffer is used for that purpose. Any waiting data can be buffered if necessary. The abstract module 'buffer' is called at that time [Figure 22].

```
ABSTRACT_LAN  osu_lan

     OVERVIEW
          LAN is an abstract representation of the
          communication model used in a small area. Using
          set_up and shut_down, the object LAN is feasible.
          The communication between users at different
          stations can occur in a LAN. By calling the LAN
          abstract modules, the communication is
          accomplished.

     TOPOLOGY      ring, star, bus

     MEDIUM        coaxial cable, fiber optic cable,
                   twisted pairs

     DATA STRUCTURE
                   service_type : service data type,
                   user_id : integer type,
                   power, busy, idle : boolean type,
                   user() : function type.

     INTERFACE     set_up, service, shut_down, user.

     OPERATION
                   set_up    : PROC() RETURN(boolean);
                   service   : PROC(service_type)
                                    RETURN(boolean);
                   shut_down : PROC() RETURN(boolean);
                   user      : PROC(user_id)
                                    RETURN(boolean);
                    :
     BEHAVIOR
                   initialize;
                     :
                   do while (power = on);
                     :
                     if (user(user_id))
                       busy;
                        :                  /* set status     */
                       osu_lan.set_up();
                        :                  /* lan invocation */
                       osu_lan.service();
                        :                /* call application */
                       osu_lan.shut_down();
                        :                  /* lan relinquish */
                     else idle;
                                           /* set status     */
                     :
                   end;

END ABSTRACT_LAN
```

Figure 16. Abstract Specification for OSU_LAN

ABSTRACT_LAN status

OVERVIEW

Status vector informs us of the current status of the LAN system. The status shows the current status of the assigned process-id and the change allows a process to change its current status.

TOPOLOGY not applicable

MEDIUM not applicable

DATA STRUCTURE

status_vector : bit array type,
process_id : integer type.

INTERFACE look_status, change_status.

OPERATION

```
look_status    : PROC(process_id)
                   RETURN(status_vector);
change_status  : PROC(process_id, status_vector)
                   RETURN(status_vector);
```

BEHAVIOR

:

END ABSTRACT_LAN

Figure 17. Abstract Specification for LAN Status

ABSTRACT_LAN file

OVERVIEW

In LAN, the files can be created or deleted by each entity and can be transmitted to each other and also copied among the entities in the LAN communication.

TOPOLOGY
not applicable

MEDIUM
not applicable

DATA STRUCTURE

file : logical records of the basic types,
source : integer type,
destination : integer type.

INTERFACE create_file, delete_file, copy_file, send_file, recv_file.

OPERATION

```
create_file : PROC(file) RETURN(pointer);
delete_file : PROC(file) RETURN(boolean);
copy_file   : PROC(file) RETURN(pointer);
send_file   : PROC(source,destination,file)
                RETURN(boolean);
recv_file   : PROC(source,destination,file)
                RETURN(pointer);
```

BEHAVIOR
:

END ABSTRACT_LAN

Figure 18. Abstract Specification for Object File

```
ABSTRACT_LAN  message

     OVERVIEW

          Electronic mail can be created or deleted by each
          entity and can be transmitted to each other and
          also copied among the processes in the LAN.

     TOPOLOGY
          not applicable

     MEDIUM
          not applicable

     DATA STRUCTURE

          message : record type or file type,
          source : integer type,
          destination : integer type.

     INTERFACE      create_msg, delete_msg, copy_msg,
                    send_msg, recv_msg.

     OPERATION

          create_msg : PROC(message) RETURN(pointer);
          delete_msg : PROC(message) RETURN(boolean);
          copy_msg   : PROC(message) RETURN(pointer);
          send_msg   : PROC(source, destination, message)
                            RETURN(boolean);
          recv_msg   : PROC(source, destination, message)
                            RETURN(pointer);

     BEHAVIOR
                    :

END ABSTRACT_LAN
```

Figure 19. Abstract Specification for Object Message

```
ABSTRACT_LAN  batch_job

     OVERVIEW

          In LAN, the batch processing job is possible
          among the processes in the LAN.

     TOPOLOGY
          not applicable

     MEDIUM
          not applicable

     DATA STRUCTURE

          job : logical structure of record type or file
                type,
          source : integer type,
          destination : integer type,
          host : integer type,
          station : integer type.

     INTERFACE      send_job, recv_job, run_job.

     OPERATION

          send_job    : PROC(source, destination, job)
                                          RETURN(boolean);
          recv_job    : PROC(source, destination, job)
                                          RETURN(pointer);
          run_job     : PROC(host, station, job)
                                          RETURN(boolean);

     BEHAVIOR
                      :

END ABSTRACT_LAN
```

Figure 20. Abstract Specification for Object Batch_job

```
ABSTRACT_LAN  pdu(protocol data unit)

     OVERVIEW

          The pdu is the data format used to transmit
          information among the processes in the LAN.

     TOPOLOGY
          applicable to all

     MEDIUM
          applicable to all

     DATA STRUCTURE

          data : pointer to pdu,
          pdu : < header : bit type,
                  data control : bit type,
                  information : record type,
                  trailer : bit type >,
          layer : integer type.

     INTERFACE     attach_header, detach_header, gen_header.

     OPERATION

          attach_header : PROC(gen_header :
                               PROC(layer, data))
                                        RETURN(data);
          detach_header : PROC(layer, data)
                                        RETURN(data);
          gen_header    : PROC(layer, data)
                                        RETURN(pointer);

     BEHAVIOR
                        :

END ABSTRACT_LAN
```

Figure 21. Abstract Specification for Object PDU

Here we can see the procedure gen_header is used as the parameter of the procedure attach_header.

ABSTRACT_LAN buffer

OVERVIEW

The buffer is for the buffering the information or the data inserted for synchronization when they are transmitted so that they are finally removed from a buffer for delivery.

TOPOLOGY
applicable to all

MEDIUM
applicable to all

DATA STRUCTURE

```
buffer : pointer type,
data : pointer type.
```

INTERFACE insert, remove.

OPERATION

```
insert : PROC(data, buffer) RETURN(pointer);
remove : PROC(data, buffer) RETURN(boolean);
```

BEHAVIOR
:

END ABSTRACT_LAN

Figure 22. Abstract Specification for Object Buffer

So far, we have specified some objects which can be implemented in a particular local area network. To do this, we left the abstraction for the protocols of each layer in LAN. The abstract specifications for the protocols in the existing LAN model are given in Chapter 6 and Appendixes. Now we confront the question of how we can use these abstraction modules. Figure 23 shows the relationship between the LAN abstract modules specified above and the service operations necessary in a certain LAN. In this figure, '-> A.B' means a function call to an abstract module for LAN object 'A' using its interface operation as a qualifier 'B'.

service operation		abstract operation
system initiation	-->	osu_lan.set_up
request service	-->	osu_lan.service
status control	-->	status.look_status status.change_status
application		
message transfer	-->	message.create_msg message.send_msg message.recv_msg
file transfer	-->	file.create_file file.send_file file.recv_file
batch_job	-->	batch_job.send_job batch_job.recv_job batch_job.run_job
synchronization	-->	buffer.insert buffer.remove
pdu manipulation	-->	pdu.attach_header pdu.detach_header
system termination	-->	osu_lan.shut_down

Figure 23. Relations between Service and Abstraction

Combining all these LAN abstract modules, now we show how this LAN abstraction methodology can be applied to the existing LAN standard protocols. The details are provided in the next chapter.

CHAPTER VI

LAN ABSTRACT SPECIFICATION APPLICATION

6.1 Application to OSI Model LAN

In a multivendor LAN environment, LAN standards are necessary to achieve compatibility. While the LAN related device companies have tried to collaborate for standards with some constraints for standardization such as economic interests, information security and political considerations, some users are also developing their own implementations using the LAN standards. It rids users of unnecessarily expensive and inefficient implementations. Among them, two big users are General Motors and Boeing company. General Motors developed a specification for communication standards in factory environment called Manufacturing Automation Protocol(MAP) [Kaminski 1986] and Boeing addressed a specification of communication network standards for the technical and office environment called Technical and Office Protocols(TOP) [Farowich 1986]. The application of the TOP includes electronic mail, word processing, file transfer, database management and distributed batch jobs. Here, we show how the specification of the abstraction associated with the LAN standard

protocol like MAP or TOP can be actually implemented in the LAN simulation program or LAN software using the abstract specification modules with some proper modifications. The IEEE 802.5 Token Ring and MAC protocol and its service function are used for the specification of the physical layer and the IEEE 802.2 Logical Link Control service and protocol are used for the specification of the data link layer. In this LAN specification, however, the network, transportation, session and presentation layers are supposed to be included in application layer. There is relatively tight relationship among these layers of the OSI model and those are not well defined for LAN. Each communication entity has its own protocol layers. With each of these, the abstractions for its service functions are specified in Appendix A. Also, we use some of the associated objects specified in the previous chapter which are necessary in a LAN system. The upper-level-layer abstraction in Appendix A shows the LAN application layer abstraction. Among the corresponding sub-layers in a certain local area network, the data link layer corresponds with the application layer as the highest sub-layer. Such IEEE 802.2 logical link control layer is shown in data-link-control abstraction in Appendix A. With LLC layer, we need a lowest layer protocol such as the IEEE 802.5 token ring. We need its peer MAC services and physical layer specifications. Abstractions of these are also specified in Appendix A.

Using abstraction modules, we show an example of implementation. In connection-mode service, a service following the user application operation in a communication system consists of three phases [Stallings 1987]: (1) connection establishment, (2) data transfer ,and (3) connection release. In Figure 24, we show how the LAN abstract module works with the possible operations when the connection-mode service is offered in a LAN. Data transmission is one of the operations. The numbers on this figure are associated with those in Figure 25.

```
    service operation                abstract operation
-------------------------    -------------------------------

system initiation             osu_lan.set_up()

service request               osu_lan.service(type)

in lan.service

  connection establishment    application.a_associate.request() --> (1)
                              application.a_associate.confirm() <-- (2)

  +-> data transfer
  |                           application.a_data.request()      --> (3)
  |                           application.a_data.confirm()      <-- (4)
  +--(loop)

  connection release          application.a_release.request()   --> (5)
                              application.a_release.confirm()   <-- (6)

system termination            osu_lan.shut_down()
```

Figure 24. Operations in Connection-Mode Service

Figure 25 shows the peer entity operations with above operations. The numbers should be matched with peer operations(refer to Figure 24 also).

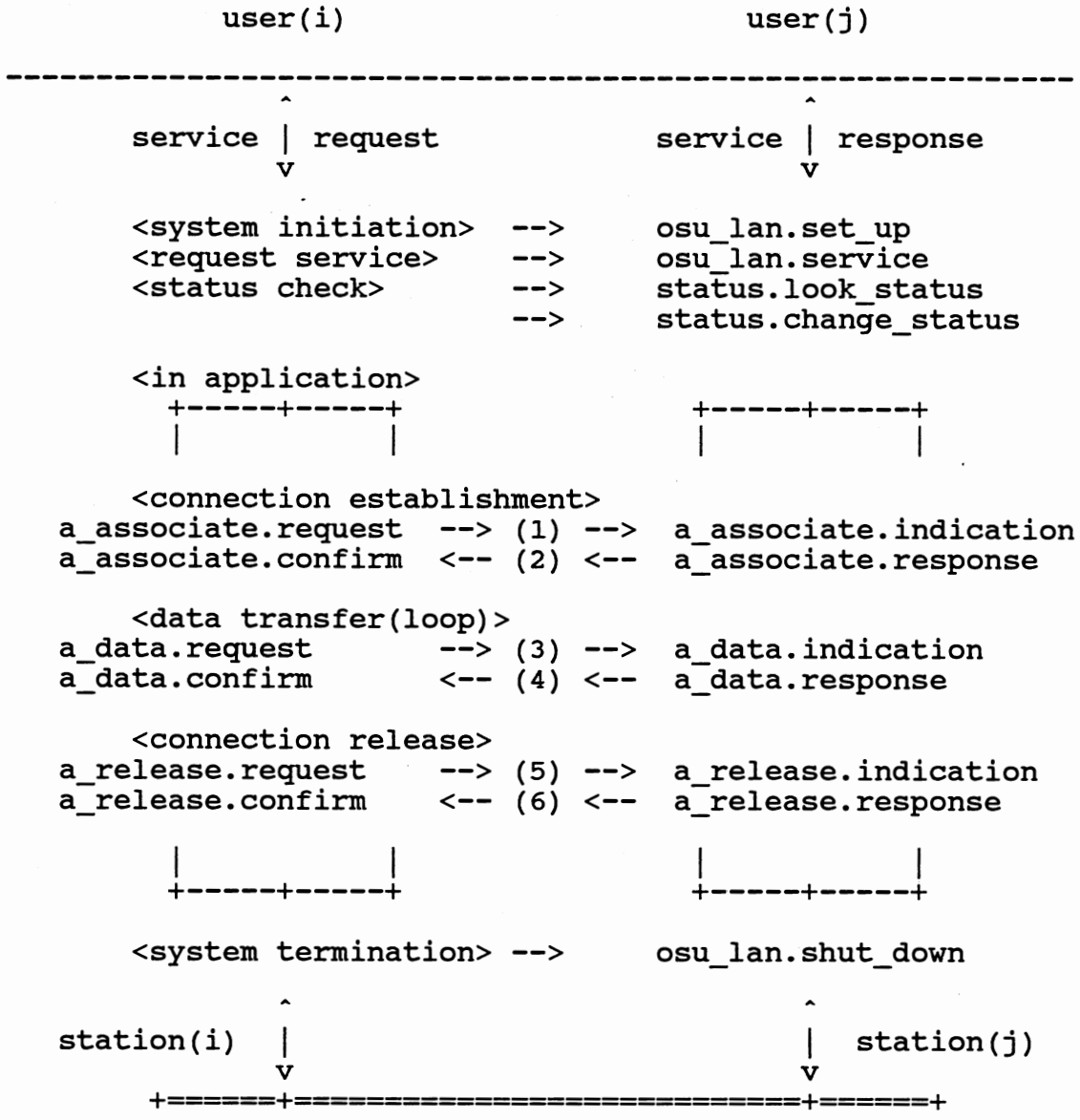


Figure 25. LAN Communication Service

Now we examine the trace of service in detail when a LAN is invoked and requested to transfer data. Most of the codes consist of functional module call using abstracted module which specifies its function in abstract specification model and the associated codes. Figure 26 shows which service is necessary and how abstract module can be called. In this figure, the "->" sign means abstract module calling.

```
lan invoked:
     -> osu_lan.set_up;                    /* lan invocation */

service requested:
     -> osu_lan.service;                   /* application    */

status check:
     -> status.look_status;
     -> status.change_status;

connection_mode service:
   try connection establishment:
   from sender:
     -> application.a_associated.request; /* from sender */
     -> link.dl_connect.request;
     -> ring.ma_unitdata.request;
     -> ring.ph_data.request;
   goes through physical medium to peer entity(receiver):
     -> ring.ph_data.indication;          /* to receiver */
     -> ring.ma_unitdata.indication;
     -> link.dl_connect.indication;
     -> application.a_associate.indication;
   in receiver entity, response connection:
     -> application.a_associate.response;
     -> link.dl_connect.response;
     -> ring.ma_unitdata.request;
     -> ring.ph_data.request;
   to sender again:
     -> ring.ph_data.indication;
     -> ring.ma_unitdata.indication;
     -> link.dl_connect.confirm;
     -> application.a_associate.confirm;
     -> status.change_status;
   connection established:

data transfer service:
     :
     -> message.send_msg;
     :
     -> pdu.attach_header;
     :
     -> buffer.insert;
     :

connection release:
     : (similar codes)

lan freed:
     -> osu_lan.shut_down;
```

Figure 26. Trace of service

6.2 Application to DOD Model LAN

We have thought about the specification of OSI model LAN abstraction. We apply abstract specification model to another LAN structure, namely, the TCP/IP model. To do this, we show how this model can be applied to DOD computer network which uses TCP/IP as sub-layer protocols. The TCP/IP are mandatory for use in DOD packet switching networks [Defense 1983]. TCP provides similar services as the transport layer does in OSI model. It provides reliable connection-oriented communication between processes in networks. TCP requires IP as sub-layer protocol. The IP provides services to transport layer and relies on the services of the lower-layer protocols. For LAN, IEEE 802 standards can be a model of such lower-layer. Figure 27 shows DOD communication architecture for LAN. Abstract specification for DOD LAN Model is in Appendix B.

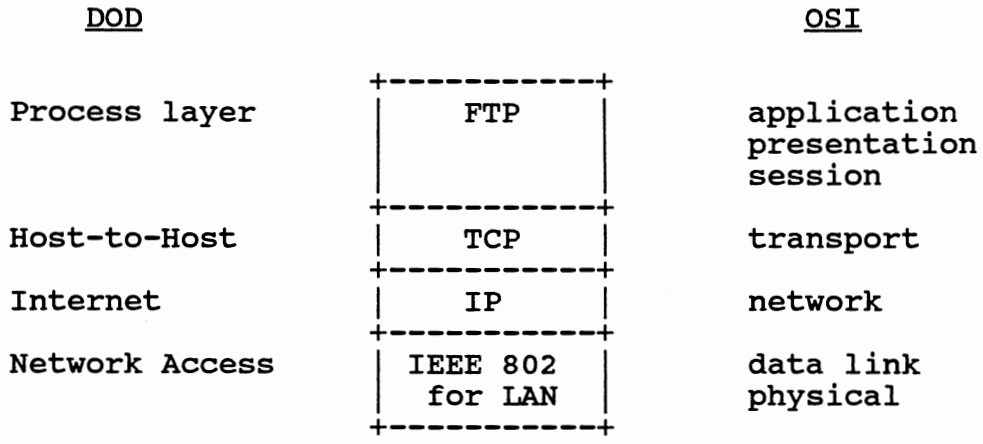


Figure 27. DOD Communication Architecture for LAN

How can DOD LAN abstraction modules be used? Figure 28 shows how abstract modules are invoked to offer the services in a DOD LAN model.

```
Service requested from port sender to port receiver:
for connection establishment:

     from sender:
     -> ulp.active_open(); /* ulp: Upper Level Protocols */
     -> tcp.active_open(); /* tcp: Trans Control Protocol*/
     -> ip.send            /* ip : Internet Protocol     */
     -> llc.connect.request();
                           /* llc: Lower Level Protocols */
     control data goes through internet:
     -> ulp.wait(connection established);

     from receiver:
     -> ulp.active_open(); or ulp.passive_open();
     similar codes:

     open matching:
     connection established:

for data transfer:

     from sender:
     -> ulp.send_data();
     -> tcp.send();
     -> ip.send();
     -> llc.data.request();
     data goes through physical medium:
     to receiver:
     -> llc.data.response();
     -> ip.deliver();
     -> tcp.deliver();
     -> ulp.deliver();
     data_transferred:
     acknowledgment:

connection termination:
     similar codes:
```

Figure 28. Trace of DOD Communication

CHAPTER VII

CONCLUSIONS

We have shown the abstraction methodology and its use in the specification of local area networks. We have presented a formal model for LAN communication in terms of the inter-layer transmission and an abstract LAN specification template. Using this abstract specification template, all levels of communication protocols are defined. Each of these is an abstract module which contains each service function of each protocol layer. An abstract module also can define an object used in a LAN system. To use each object, its abstract module is called. Data, procedural and behavioral abstractions of the LAN objects are specified with this object module. How the abstract specification can be used when it is implemented is illustrated by tracing the service function calling and showing the inter-relations between the abstract modules. The OSI Basic Reference Model and IEEE 802 LAN standard protocols have been chosen as examples for LAN abstract specifications. The same abstraction model has been applied to TCP/IP in DOD packet switching network model for showing the usability and compatibility of the abstraction model.

LAN simulation using this abstract specification is

one of the possible topics for research. To do this implementation, the selection of the proper programming language and the other decisions such as the process state transition, data packet format, error condition and the buffer size considering the LAN performance are prerequisite. The selection of a high-level language for the implementation of a LAN abstract specification module is also a topic for research. Other research topics include network operating systems, network software and network user interface commands with their interpreter and compiler using their abstract specification schemes.

BIBLIOGRAPHY

Berntsen, J.A., Davin, J.R., Pitt,D. A., and Sullivan, N.G. MAC layer interconnection of IEEE 802 local area networks. Computer Networks and ISDN Systems 10, 5(1985), 259-273.

Berztiss, A. T. and Thatte, S. Specification and implementation of abstract data types. Advances in Computers, 22(1983) Academic Press, 295-353.

Blumer, T. P. and Tenney, R. L. A formal specification technique and implementation method for protocols. Computer Networks 6, 3(1982), 201-217.

Bochmann, G. V., Finite state description of communication protocols. Computer Networks 2, 4/5(1978), 361-372.

Bochmann, G. V. and Sunshine, C. A. Formal methods in communication protocol design. IEEE Trans. on Comm. COM-28, 4(1980), 624-631.

Cleaveland, J. C. An Introduction to Data Types. Addison-Wesley Publishing Co.(1986).

Comer, Douglas. Operating System Design - Vol. 2. Internetworking with Xinu. Prentice-Hall, Inc.(1987).

Dasgupta, S. Computer design and description languages. Advances in Computers,21(1982),Academic Press, 91-154.

Derfler, F. J. Jr. and Stallings, W. A Manager's Guide to Local Networks. Prentice-Hall, Inc., NJ(1983).

Defense Communication Agency. Military Standard: Internet Protocol. MIL-STD-1777(1983)., Transmission Control Protocol. MIL-STD-1778(1983)., File Transfer Protocol. MIL-STD-1780(1984)., Simple Mail Transfer Protocol. MIL-STD-1781(1984)., Telnet Protocol. MIL-STD-1782 (1984).

Ehrich, H.-D. On the theory of specification, implementation, and parametrization of abstract data types. JACM, Vol.29, No.1(January 1982), 206-227.

Engels, C., Pletat, U., and Ehrich H.-D. An operational semantics for specifications of abstract data types with error handling. Acta Informatica 19, 3(1983), 235-253.

Farowich, S. A. Communicating in the technical office. IEEE Spectrum Vol.23(April 1986), 63-67.

Gehani, N. and McGettrick, A. D. Software Specification Techniques. Addison-Wesley, International Computer Science Series(1986)

Graube, M. and Mulder, M. C. Local area networks. IEEE Computer, Vol.17, No.10(Oct. 1984), 242-247.

Guttag, J. V. and Horning J. J. The algebraic specification of abstract data types. Acta Informatica 10, 1(1978), 27-52.

Guttag, J. V., Horowitz, E. and Musser, D. R. Abstract data types and software validation. Comm. of the ACM 21, 12(1978), 1048-1064.

Guttag, John V. Notes on type abstraction(Version 2). IEEE Trans. on Software Eng., Vol.SE-6, 1(Jan. 1980), 13-32.

Hawe, B., Kirby, A. and Lauck, A. An architecture for transparently interconnecting IEEE 802 LAN. DEC-TR-322, Digital Equipment Corporation(1984).

Hawe, W. R. and Varghese, G. Extended local area network management principles. DEC-TR-324, Digital Equipment Corporation(1984).

ISO/TC97/SC16. Data Processing - Open Systems Interconnection - Basic Reference Model. ISO/TC97/SC16, ANSI, 1430 Broadway, New York, NY. Computer Networks 5,1(1981), 81-118.

Kaminski, M. A. Jr. Protocols for communicating in the factory. IEEE Spectrum Vol.23(April 1986), 56-62.

Keller, R. M. Formal verification of parallel programs. Comm. of the ACM 19, 7(1976), 371-384.

Kessler, G. Ethernet vs. IEEE 802.3. LAN Magazine(July 1987).

Kummer, P., Tasker, R., Linge, N. and Ball, E. A protocol-less scheme for bridging between IEEE 802 Local Area Networks. Computer Networks and ISDN Systems 12, 2(1987), 81-87.

Lam, Simon S. Tutorial: Principles of Communication and Networking Protocols. IEEE Computer Society, IEEE Catalog No. EHO216-2 (1984).

Liskov, B. and Guttag J.
Abstraction and Specification in Program Development. The MIT Press, McGraw-Hill Book Co.(1986).

Liskov, B. and Zilles, S. Specification techniques for data abstractions. IEEE Trans. on Software Eng. 1,1(1975), 9-19.

Mclean, J. A formal method for the abstract specification of software. JACM, Vol.31,No.3(July 1984), 600-627.

McQuillan, J. M. Local network technology and the lessons of history. Computer Networks 4,5(1980), 235-238.

Merlin, P. M. Specification and validation of protocols. IEEE Trans. on Comm. COM-27, 11(1979), 1671-1680.

Milner, Robin. A Calculus of Communicating Systems. Springer-Verlag, Berlin, Lecture Notes in Computer Science 92(1980).

Nourani, C. F.
Abstract implementations and their correctness proofs. JACM, Vol.30,No.2(April 1983), 343-359.

Saltzer, J. H., Pogran, K. T. and Clark, D. D.
Why a ring? Computer Networks 7,4(1983), 223-231.

Sincovec, R. F. and Wiener, R. S. Data Structures using Modula-2. John Wiley & Sons, New York(1986).

Stallings, William. Data and Computer Communications. Macmillan Publishing Co. New York(1985).

Stallings, William. Handbook of Computer-Communications Standards. Vol.1 The Open Systems Interconnection(OSI) Model and OSI-Related Standards, Vol.2 Local Networks Standards and Vol.3 Department of Defense(DOD) Protocol Standards. Macmillan Publishing Co. New York (Vol.1,2 : 1987, Vol.3 : 1988).

Stallings, William. Local networks.
Computing Survey, Vol.16,No.1(March 1984),3-41.

Stallings, William. Local Networks. 2nd Ed.
Macmillan Publishing Co. New York(1987).

Stallings, William. Tutorial: Local Network Technology. 2nd Ed. IEEE Computer Society, IEEE Catalog No. EHO234-5 (1985).

Sunshine, C. A. and Dalal, Y. K. Connection management in Transport protocols. Computer Networks 2, 6(1978), 454-473.

Sunshine, C. A. Formal Modeling of Communication Protocols. ISI/RR-81-89, USC(March 1981).

Sunshine, C. A. Formal techniques for protocol specification and verification. Tutorial: Principles of communication and networking protocols. IEEE Computer Society(ISBN 0-8186-0582-0),(1984), 467-473.

Tanenbaum, A. S. Computer Networks. Prentice-Hall, New Jersey(1981).

Tanenbaum, A. S. Network protocols. ACM Computing Surveys, Vol.13,No.4(December 1981), 453-489.

Wirsing, M., Pepper, P., Partsch, H., Dosch, W., and Broy, M. On hierarchies of abstract data types. Acta Informatica 20,1(1983), 1-33.

APPENDIXES

APPENDIX A

OSI MODEL ABSTRACT MODULES

For upper_level_layer(ISO Application Layer):

ABSTRACT_LAN application

OVERVIEW
Application layer is the highest layer in a network system. Application layer offers the service of user oriented high level protocol such as electronic mail, a file transfer, and remote job manipulation.

TOPOLOGY ring

MEDIUM twisted pair

DATA STRUCTURE
pdu : pointer type, calling p_id : integer type,
called p_id : integer type,
application_context : pointer type,
responding p_id : integer type, result: bit type,
user_information : bit type.

INTERFACE

a_associate.request,	a_associate.indication,
a_associate.response,	a_associate.confirm,
a_data.request,	a_data.indication,
a_data.response,	a_data.confirm,
a_release.request,	a_release.indication,
a_release.response,	a_release.confirm,

OPERATION

```
PROC  a_associate.request(calling p_id : INTEGER;
        called  p_id : INTEGER;
        application_context : POINTER);
PROC  a_associate.indication(calling p_id :
        INTEGER; called p_id : INTEGER;
        application_context: POINTER);
PROC  a_associate.response(responding p_id :
        INTEGER; application_context: POINTER;
        result : BIT);
PROC  a_associate.confirm(responding p_id :
        INTEGER; application_context: POINTER;
        result : BIT);
PROC  a_data.request(calling p_id : INTEGER;
        called p_id : INTEGER; pdu : POINTER);
PROC  a_data.indication(calling p_id : INTEGER;
        called p_id : INTEGER; pdu : POINTER);
PROC  a_data.response(calling p_id : INTEGER;
        called p_id : INTEGER; result: BIT);
PROC  a_data.confirm(calling p_id : INTEGER;
        called p_id : INTEGER; result : BIT);
PROC  a_release.request(user_information : BIT);
PROC  a_release.indication(user_information:BIT);
```

```
        PROC  a_release.response(user_information : BIT;
                result : BIT);
        PROC  a_release.confirm(user_information : BIT;
                result : BIT);

    BEHAVIOR

        prologue;

        switch

         case(a_associate.request)
           link.dl_connect.request;
         case(a_associate.indication)
           while (busy = status.look_status) wait();
           application.a_associate.response;
         case(a_associate.response)
           link.dl_connect.response;
         case(a_associate.confirm)
           status.change_status(source_address, ready);
           status.change_status(destination_address,
             ready);

         case(a_data.request)
           while((^end_of_data)&(error))
             segment data into pdu;
             link.dl_data.request;
             wait(a_data.confirm);     /* confirm  ACK */
             retransmit or error;
         case(a_data.indication)
           application.a_data.response;/* response ACK */
         case(a_data.response)
           link.dl_data.response;

         case(a_release.request)
           link.dl_disconnect.request;
         case(a_release.indication)
           status.change_status(process_id,idle);
           application.a_release.response;
         case(a_release.response)
           link.dl_disconnect.response;
         case(a_release.confirm)
           status.change_status(process_id,ready);

         case(null)
           idle state;

        end_switch;

        epilogue;

END ABSTRACT_LAN
```

For data_link_control(IEEE 802.2 Logical Link Control):

```
ABSTRACT_LAN  link                                  /* IEEE 802.2 */

     OVERVIEW
          Logical link control(LLC) is the highest layer in
          a single local area network architecture. LLC
          layer provides connection oriented service
          between LLC users across a MAC controlled link.

     TOPOLOGY      ring

     MEDIUM        twisted pair

     DATA STRUCTURE
          pdu : pointer type, source_address: integer type,
          destination_address : integer type,
          priority : integer type, data : pointer type,
          amount : integer type, status : bit type.

     INTERFACE
          dl_connect.request,     dl_connect.indication,
          dl_connect.response,    dl_connect.confirm,
          dl_data.request,        dl_data.indication,
          dl_data.response,       dl_data.confirm,
          dl_disconnect.request,  dl_disconnect.indication,

     OPERATION
          PROC  dl_connect.request(source_address :
                  INTEGER; destination_address : INTEGER;
                  priority : INTEGER);
          PROC  dl_connect.indication(source_address :
                  INTEGER; destination_address : INTEGER;
                  priority : INTEGER);
          PROC  dl_connect.response(source_address :
                  INTEGER; destination_address : INTEGER;
                  priority : INTEGER);
          PROC  dl_connect.confirm(source_address :
                  INTEGER; destination_address : INTEGER;
                  priority : INTEGER);
          PROC  dl_data.request(source_address : INTEGER;
                  destination_address : INTEGER ;
                  data : POINTER);
          PROC  dl_data.indication(source_address :
                  INTEGER; destination_address : INTEGER;
                  data : POINTER);
          PROC  dl_data.response(source_address : INTEGER;
                  destination_address:INTEGER; status:BIT);
          PROC  dl_data.confirm(source_address : INTEGER;
                  destination_address:INTEGER; status:BIT);
```

```
        PROC  dl_disconnect.request(source_address :
                INTEGER; destination_address:INTEGER);
        PROC  dl_disconnect.indication(source_address :
                INTEGER; destination_address:INTEGER);
        PROC  dl_reset.request(source_address : INTEGER;
                destination_address:INTEGER);
        PROC  dl_reset.indication(source_address :
                INTEGER; destination_address:INTEGER);
        PROC  dl_reset.response(source_address : INTEGER;
                destination_address:INTEGER);
        PROC  dl_reset.confirm(source_address : INTEGER;
                destination_address:INTEGER);
        PROC  dl_connection_flowcontrol.request
                (source_address : INTEGER;
                 destination_address:INTEGER;
                 amount : INTEGER);
        PROC  dl_connection_flowcontrol.indication
                (source_address : INTEGER;
                 destination_address:INTEGER;
                 amount : INTEGER);

BEHAVIOR

        prologue;

        switch

          :
          case(dl_connect.request) | (dl_data.request)
               ring.ma_unitdata.request;

          case(dl_connect.indication) |
               application.a_associate.indication;

          case(dl_connect.response) | (dl_data.response)
               ring.ma_unitdata.request;

          case(dl_connect.confirm)
               application.a_associate.confirm;

          case(dl_data.indication)
               application.a_data.indication;

          case(dl_data.confirm)
               application.a_data.confirm;
          :
          case(null)
               idle state;

        end_switch;

        epilogue;

END ABSTRACT_LAN
```

For medium_access_control(IEEE 802.5 MAC)
and physical_layer(IEEE 802.5 Token Ring):

```
ABSTRACT_LAN  ring                              /* IEEE 802.5 */

     OVERVIEW
          The token ring medium access control(MAC)
          protocol is for token topology. MAC protocol,
          service, and physical layer is specified.

     TOPOLOGY      ring only

     MEDIUM        shielded twisted pair

     DATA STRUCTURE
          pdu : pointer type, frame_control : bit type,
          source_address : integer type, symbol : bit type,
          destination_address : integer type,
          m_sdu : pointer type, frame_control : bit type,
          reception_status : bit type,
          service_class : integer type,
          transmission_status : bit type.

     INTERFACE
          ma_unitdata.request, ma_unitdata.indication,
          ma_unitdata.status,
          ph_data.request, ph_data.indication,
          ph_data.confirmation.

     OPERATION

          PROC  ma_unitdata.request(frame_control: BIT;
                  destination_address: INTEGER;
                  m_sdu : POINTER;
                  service_class : INTEGER);
          PROC  ma_unitdata.indication
                  (frame_control: BIT;
                   destination_address: INTEGER;
                   source_address : INTEGER
                   m_sdu : POINTER;
                   reception_status: BIT);
          PROC  ma_unitdata.status (transmission_status:
                  BIT; service_class : INTEGER);

          PROC  ph_data.request(symbol: BIT);
          PROC  ph_data.indication(symbol: BIT);
          PROC  ph_data.confirmation(symbol: BIT);
```

```
BEHAVIOR

     prologue;

     switch

       case(ma_unitdata.request)
            ring.ph_data.request(symbol);

       case(ma_unitdata.indication)
            if (connection_establishment)
              if (connection_request)
                link.dl_connect.indication;
              if (connection_response)
                link.dl_connect.confirm;
            if (data_transfer)
              if (data_request)
                link.dl_data.indication;
              if (data_response)
                link.dl_data.confirm;
       :
       case(ph_data.request)
            goes through media to entity;
            ring.ph_data.indication;

       case(ph_data.indication(symbol))
            ring.ma_unitdata.indication;

       :
       case(null)
            idle state;

     end_switch;

     epilogue;

END ABSTRACT_LAN
```

APPENDIX B

DOD MODEL ABSTRACT MODULES

For upper_level_layer(process layer):

ABSTRACT_LAN ulp(upper level protocol)

OVERVIEW

ULP is high level protocol like file transfer protocols. FTP(File Transfer Protocol) supports file transfer between processes in communication networks. It provides high level service associated with the next lower transport layer protocols.

TOPOLOGY ring, bus, star

MEDIUM twisted pair, coaxial cable, fiber optic cable

DATA STRUCTURE

```
source_address : integer type,
destination_address : integer type,
file : pointer to record type
source_port : integer type,
destination_port : integer type,
local_connection_name : integer type,
data : pointer to record type,
data_length : integer type,
connection_state : integer type,
description : integer type,
passive/active_flag : boolean type,
close/abort_flag : boolean type.
```

INTERFACE

connection, send_data, termination, check_status, open_ID, open_failure, open_success, deliver_data, closing, terminate, status_response, error.

OPERATION

```
PROC  connection(source_port : INTEGER;
                 destination_port : INTEGER;
                 passive/active_flag : BIT);
PROC  send_data(local_connection_name : INTEGER;
                data : POINTER);
PROC  termination(local_connection_name: INTEGER;
                  close/abort_flag : BIT);
PROC  check_status(local_connection_name :
                   INTEGER);
```

```
PROC  open_ID(local_connection_name : INTEGER;
              source_port : INTEGER;
              destination_port : INTEGER);
PROC  open_failure(local_connection_name :
                   INTEGER);
PROC  open_success(local_connection_name :
                   INTEGER);
PROC  deliver_data(local_connection_name :
                   INTEGER; data : POINTER;
                   data_length : INTEGER);
PROC  closing(local_connection_name : INTEGER);
PROC  terminate(local_connection_name : INTEGER;
                description : INTEGER);
PROC  status_response(
                   local_connection_name : INTEGER;
                   source_port : INTEGER;
                   destination_port : INTEGER;
                   connection_state : INTEGER);
PROC  error(local_connection_name : INTEGER;
            description : INTEGER);
```

BEHAVIOR

```
prologue;

switch

/* service request */
     case(connection) /* connection establishment */
          if(passive)
             tcp.passive_open();
          if(active)
             tcp.active_open();
          wait(connection_established);


       case(send_data)   /* connection maintenance */
            tcp.send();


       case(termination) /* connection termination */
            if(close)
               tcp.close();
            if(abort)
               tcp.abort();
            wait(connection_terminated);

       case(check_status)     /* connection status */
            tcp.status();
```

```
/* service response */
              case(open_ID)    /* connection establishment */
                   connection_assigned;

              case(open_failure)
                   failure_of_Active_Open;

              case(open_success)
                   completion_of_pending_Open_request;

              case(deliver_data)
                                 /* connection maintenance */
                   acknowledgment;

              case(closing)      /* connection termination */
                   connection_closed;
              case(terminate)
                   connection_aborted;

              case(status_response)
                                /* current connection info.*/
                   current_connection_status;

              case(error)                      /* error case */
                   error_process(retry | abort);
              case(null)                     /* null service */
                   idle state;

            end_switch;

            epilogue;

END ABSTRACT_LAN
```

For TCP(Transmission Control Protocol):

ABSTRACT_LAN tcp

OVERVIEW

TCP is connection_oriented transport protocol for use in packet_switched communication networks in which data transfer is reliable, ordered, full_duplex, and flow controlled.

TOPOLOGY ring, bus, star

MEDIUM twisted pair, coaxial cable, fiber optic cable

DATA STRUCTURE

```
source_port : integer type,
destination_port : integer type,
local_connection_name : integer type,
data : pointer to record type,
data_length : integer type,
connection_state : integer type,
description : integer type.
```

INTERFACE

```
/* service request primitives */
passive_open, active_open,
send, allocate, close, abort, status
/* service response primitives*/
open_ID, open_failure, open_success,
deliver, closing, terminate, status_response,
error.
```

OPERATION

```
PROC  passive_open(source_port : INTEGER;
                   destination_port : INTEGER);
PROC  active_open(source_port : INTEGER;
                  destination_port : INTEGER);
PROC  send(local_connection_name : INTEGER;
           data : POINTER);
PROC  close(local_connection_name : INTEGER);
PROC  abort(local_connection_name : INTEGER);
PROC  status(local_connection_name : INTEGER);
PROC  open_ID(local_connection_name : INTEGER;
              source_port : INTEGER;
              destination_port : INTEGER);
PROC  open_failure(local_connection_name :
                   INTEGER);
```

```
PROC  open_success(local_connection_name :
                   INTEGER);
PROC  deliver(local_connection_name : INTEGER;
              data : POINTER; data_length :
              INTEGER);
PROC  closing(local_connection_name : INTEGER);
PROC  terminate(local_connection_name : INTEGER;
                description : INTEGER);
PROC  status_response(
                local_connection_name : INTEGER;
                source_port : INTEGER;
                destination_port : INTEGER;
                connection_state : INTEGER);
PROC  error(local_connection_name : INTEGER;
            description : INTEGER);
```

BEHAVIOR

```
prologue;

switch

  /* service request primitives */

  case(passive_open)
       ip.send();
              /* listen for connection attempt */

  case(active_open)
       ip.send();        /* request connection */

  case(send)
       ip.send();             /* transfer data */

  case(close)
       ip.send();
                /* close connection gracefully */

  case(abort)
       ip.send();
                 /* close connection abruptly */

  case(status)
       ip.send();   /* query connection status */
```

```
            /* service response primitives */
            /* ulp : Upper Level Protocols */
            case(deliver)
                 if(open_ID)
                    ulp.open_ID();
                                 /* informs lcn assigned */
                 if(open_failure)
                    ulp.open_failure();
                              /* failure of Active Open */

                 if(open_success)
                    ulp.open_success();
                           /* completion of Open request*/
                 if(deliver)
                    ulp.deliver();    /* arrival of data */
                 if(closing)
                    ulp.closing();
                           /* peer ULP issued a CLOSE */
                 if(terminate)
                    ulp.terminate();
                           /* remote connection reset */
                 if(status_response)
                    ulp.status_response();
                                    /* current status */
                 if(error)
                    ulp.error();
                           /* illegal service request */
            case(null)
                 idle state;

          end_switch;

          epilogue;

END ABSTRACT_LAN
```

For IP(Internet Protocol):

ABSTRACT_LAN ip

OVERVIEW

IP supports the interconnection of communication sub_networks. IP provides services to the upper transport layer protocols and relies on the services of the lower network layer protocols.

TOPOLOGY ring, bus, star

MEDIUM twisted pair, coaxial cable, fiber optic cable

DATA STRUCTURE

source_port : integer type,
destination_port : integer type,
local_connection_name : integer type,
data : pointer to record type,
data_length : integer type,
connection_state : integer type,
description : integer type.

INTERFACE

send, deliver.

OPERATION

```
PROC  send(source_address : INTEGER;
           destination_address : INTEGER;
           data_length : INTEGER;
           data : POINTER);

PROC  deliver(source_address : INTEGER;
              destination_address : INTEGER;
              data_length : INTEGER;
              data : POINTER);
```

```
BEHAVIOR

     prologue;

     switch

       case(send)
            if((active_open) | (passive_open))
               llc.connect.request();
                        /* llc : Low Level Protocols */
            if(send_data)
               llc.data.request();

            if((close) | (abort))
               llc.disconnect.request();

            if(status)
               llc.status.request();

       case(deliver)
            if((open_ID) |
               (open_failure) |
               (open_success) |
               (deliver_data) |
               (close) |
               (abort) |
               (status_response) |
               (error))
               tcp.deliver();

       case(null)
            idle state;

     end_switch;

     epilogue;

END ABSTRACT_LAN
```

For sub_layers:

Similar codes with IEEE 802 LAN and the OSI Reference Model in Appendix A.

VITA

Chang-Hyun Jo

Candidate for the Degree of

Master of Science

Thesis: ABSTRACTION AND SPECIFICATION OF LOCAL AREA NETWORKS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Pusan, Korea, April 25, 1958, the son of Soon-Kyu and Boon-Sun Jo.

Education: Graduated from Myong Ji Senior High School, Seoul, Korea, in February, 1976; received Bachelor of Economics in Statistics from Sung Kyun Kwan University, Seoul, Korea, in February, 1984; completed requirements for the Master of Science degree at Oklahoma State University in July,1988.

Professional Experience: Software Engineer, Electronic Research Lab., Hyo Sung Corp., Seoul, Korea, December, 1983, to April, 1985; Teaching Assistant, Department of Computing and Information Sciences, Oklahoma State University, August, 1987, to July, 1988;