CHAPTER 1
INTRODUCTION

S. MAUW & G.J. VELTINK

1.1 AIM AND SCOPE

An important reason why formal description techniques are not appreciated as widely as wished by the developers of such techniques, is that people who actually design and implement software have relatively little knowledge of formal methods. The acceptance of formal techniques not only depends on the existence of techniques that are easy to understand and easy to use, but also on the training of potential users. This implies that there is a need for text books and case-studies. We think that a collection of formal specifications in a restricted area of application may help to get a better understanding of the use of formal techniques. Although the method we use is well suited for formal verification, we concentrate on the act of specification. A first requirement for a formal correctness proof is a formal specification.

We restrict ourselves in this book to a collection of specifications concerning one application area, the field of communication protocols. Although this seems to be an area with a relatively high acceptance of formal techniques, most of the protocols that are actually in use are specified in natural language, if ever specified otherwise than by the actual implementation. Even well-known and accepted standards, such as the token ring protocol, do not have a rigorous formal definition. Informal specifications in this area may lead to misinterpretations and, thus, to different implementations that will not be able to work together. Formal techniques are especially needed for communication protocol design, since these protocols describe distributed systems which have a high degree of non-determinism. This implies

that errors are not easily detected and hard to reproduce, which makes traditional testing techniques too unreliable for validation purposes.

The protocols described in this book cover a wide range. We start with the description of simple point-to-point protocols, such as the Alternating Bit Protocol, which have been specified and verified many times with a variety of formal techniques. The more complex protocols in this book are used in practice and some of them have never been subject of formal specification and verification before.

The main aim of this book is to provide the reader with a collection of protocol descriptions which illustrates how to use algebraic specification techniques, be it in the field of communication protocols or in a related area. Furthermore, we wish to give insight in the design and operation of communication protocols. The specifications in this book have a level of abstraction that is appropriate for a clear understanding of the protocols without having to deal with implementation details.

Although based on a formal theory of concurrent processes, this book can be understood without previous knowledge of process algebras. We think that beginners as well as professionals in the field of communication protocols will benefit from studying the protocol descriptions provided. The sections on simple protocols are tutorial, whereas the more complex protocols are topics of current research and are being used in modern networks.

1.2 Formal Methods

In academia it is generally recognized that formal methods are a useful technique for software design. The main advantage of the use of formal techniques is that a formal specification is a mathematical object which has an unambiguous meaning, therefore mathematical methods may be used to analyse these specifications, such as formal verification of the correctness and completeness of a specification.

Formal methods play a vital rôle, especially in the field of protocol standardization. Standards are needed for providing connectivity among systems. Adhering to a formal specification guarantees that system components from different manufacturers can be interchanged easily. A formal specification can be used to derive test cases automatically, in order to check whether a particular implementation behaves as expected. If the implementation language is a formal language as well, it even makes it possible to verify formally that an implementation satisfies a given specification.

For most formal techniques, software tools have been developed that aid in analyzing a given specification. Apart from syntax-checking and type-checking, most software environments for formal methods also provide a means for rapid prototyping. Even automatic generation of an implementation is sometimes possible. Other tools can generate test sequences that can be used for conformance testing, which is an important validation activity in the construction of software.

A great variety of formal specification techniques exist, some of which are general purpose (such as Z, VDM or COLD), while others are generally used in a specific domain of application (such as LOTOS, SDL and PSF). The mathematical theories

on which these languages are based range from set theory and temporal logic to lambda-calculus and process algebra.

Formal methods have been applied in the design of a great number of systems. Many protocols have been specified and verified formally. Some protocol standards are even defined by means of a formal method.

The specifications in this book are written in the language PSF (Process Specification Formalism). This is a formal specification language based on the Algebra of Communicating Processes (ACP). PSF can be seen as a concrete representation for the process theory ACP. Furthermore, PSF supports the use of abstract data types and has special features for modularization and parameterization of specifications.

ACP is a theory for the specification and verification of concurrent systems. It can be viewed as a generalization of other theories for concurrency such as CCS and CSP. An ACP specification consists of a collection of algebraic process definitions and verification is done by algebraic manipulation of these processes. Its semantics are relatively simple and it can be extended easily with special features such as real time and interrupts. ACP has been applied in the specification and verification of many concurrent systems.

In this book we advocate the use of algebraic techniques for protocol specification. However, we regard the PSF language as one of the many ways to obtain formal protocol specifications and do not claim that it is the specification language best suited for communication protocols in general.

1.3 Computer Networks

The way in which people interact with computers has changed significantly since their introduction. In the early days only one user could be working with a computer at a time. The introduction of so-called *time-sharing* systems made it possible for computers to be used by a number of users at the same time. Nowadays, with decreasing prices of computer hardware, mainframes are being more and more replaced by clusters of smaller machines connected by a *network*. An important reason for this is that the price/performance ratio of a personal computer is much better than that of a mainframe.

A second reason for using networks, is that they allow one to construct systems that share resources like *file servers* and *printers*. Such systems can also achieve an increased reliability by offering duplicated services. If one computer or peripheral device is out of order, other machines can take over its tasks.

The final reason for using computer networks, is that they make it possible to connect computer systems that are spread out geographically. A company with branches all over the world that each have their local computer systems can access information about all different branches through a network that connects the local systems.

1.3.1 LANS & WANS

There are two important classes of networks. A Local Area Network (LAN) is used when the distances between the computers are not too large. A typical example is a network that connects the computers in one building. Each user has his own computer and all users share output devices and storage devices using the LAN.

A Wide Area Network (WAN) is used when the distance between two computers is larger, for example to establish a connection between different cities. An example of a world-wide WAN is *USENET*. One of the services offered by this network is *electronic mail*, which enables people all around the world to exchange messages.

1.3.2 NETWORK STRUCTURE

A network consists of a collection of computers, called *hosts*, and a *subnet* that connects the hosts. The subnet consists of *transmission lines*, the medium through which the data is transported, and special dedicated processors IMPs (Interface Message Processor) which connect two or more communication channels.

There are two important designs for the subnet. The first design uses a so-called *point-to-point* subnet. All IMPs are connected by wires or telephone lines. IMPs that are directly connected can communicate with each other. However, two IMPs that are not directly connected must use one or more intermediate IMPs to communicate. Messages are sent to an intermediate IMP that stores them and waits until the channel to the next IMP is free and sends the message further. Most WANs are implemented using point-to-point subnets.

The second communication architecture, which is mainly used in LANs, uses a mechanism called *broadcasting*. Whenever a message is sent by one machine, all other machines connected to the subnet receive this message. The message contains an address, which indicates for which machine the message was intended. The receiving stations that do not have a matching address, simply discard the message.

1.3.3 OSI REFERENCE MODEL

In the previous section we discussed how computers can be connected on the physical level. Having such a connection does not automatically imply that two computers can *understand* each other. It might well be the case that both computers use a different way of representing information. To make sure that they can communicate they have to adhere to certain agreements of how to exchange data. Such an agreement is called a *protocol*.

In the early days of networks, each manufacturer used his own protocols. Such an approach makes it difficult to connect systems of different manufacturers. To overcome such problems the ISO (International Standards Organization) has made a proposal for an international standard for networking. This standard is called the ISO OSI (Open Systems Interconnection) Reference Model.

The OSI model is divided into seven layers, as shown in Figure 1.1. One important reason for dividing the network into layers is to reduce the design complexity of a network system. Each layer has its own specific task and offers services to a higher layer, based on the services offered by a lower layer. In this way a layer is shielded from the *implementation details* of lower layers.

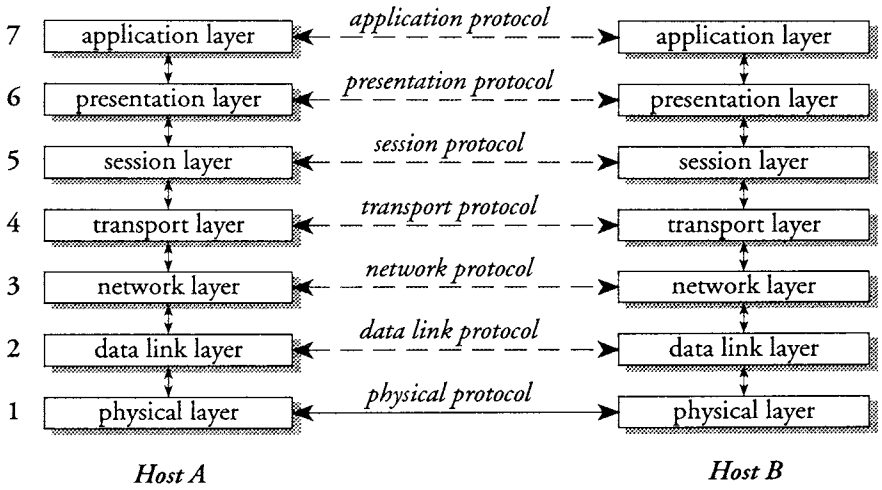

Figure 1.1 The ISO OSI Reference Model

A layer on level n on one computer communicates with the layer on level n on another computer. The set of rules for communication that they adhere to is called the *protocol* of layer n. The processes in the same layer, in two different computers that communicate with each other, are called *peer* processes.

There is no direct transmission of data from layer n on one machine to layer n on the other machine. Instead, data is sent to the layer immediately below, until the physical layer is reached. The physical layer forms the only actual connection between two computers. All other layers are in fact different levels of abstraction of communication.

The seven layers are roughly divided into two groups, levels 1 to 4 are called the *communication layers* and we refer to the levels 5 to 7 as *application layers*. The lower layers take care of the exchange of information between two computers. The upper layers contain, among others, encryption and decryption techniques, transformations between different methods of character representation such as ASCII and EBCDIC, and routines for transferring a file from one file system to another. Only the communication layers of the OSI model will be of interest for the topics treated in this book, so we will briefly describe their functions.

The *physical layer* is dedicated to the transmission of bits over a physical communication channel. In this layer there are standards that describe how the different bit values *0* and *1* must be represented.

The transmission of data on the physical level can be distorted in many ways, because data often has to travel a long way through physically unreliable channels, such as the atmosphere and telephone wires. It is the responsibility of the *data link layer* to transform this unreliable connection into an error-free communication channel for the higher layers.

The main task of the *network layer* is to determine the route by which data has to travel through the network. If data has to travel from one network to another, it

is also the responsibility of the network layer to offer services that enable this coupling of networks.

It is the responsibility of the *transport layer* to decide how a *transport connection*, needed by the higher layers, is mapped onto the available *network connections*. If two hosts are connected by more than one network connection, the transport layer can decide to split the data and send it along the different network connections to achieve better performance. This technique is called *downward multiplexing*. On the other hand it is possible that the upper layers need more transport connections than available network connections. In this case the transport layer can decide to merge the data of several transport connections, so that they can be transported over fewer network connections. This technique is called *upward multiplexing*.

1.3.4 TERMINOLOGY

The protocols that are described in this book, are mainly situated in the data link layer. We recall that the main task of this layer is to offer the network layer an error-free communication channel. In this section we will introduce the terminology used in the specifications of the protocols from the data link layer.

The specifications deal with the transmission of data from a *sender* to a *receiver*, via a *channel*. There are three possibilities of data transfer along a channel with respect to the direction of the data.

- *simplex*: data can be transported in only one direction
- *half-duplex*: data can be transported in both directions but only in one direction at a time
- *full-duplex*: data can be transported in both directions at the same time

To be able to detect transmission errors that may occur in the physical layer, the data stream is augmented with extra information also called a *checksum*. The method normally used to calculate this checksum is the Cyclic Redundancy Code (CRC). The receiver verifies the checksum of the incoming data and if this verification fails it raises a *checksum error* to indicate that something went wrong. One possible action then is to ask the sender to retransmit the garbled data.

In the PSF specifications it is required that a communication channel is *fair*. This means that a channel does not produce an infinite stream of garbled data, but that it will sooner or later transmit a datum correctly.

1.4 OVERVIEW

This book contains seven chapters, most of which can be read and understood independently. A prerequisite for understanding Chapters 3 to 7 is knowledge of the PSF language, which is explained in Chapter 2. All specifications in this book are analyzed with the checking and simulation tools from the PSF-Toolkit. Knowledge

of the PSF-Toolkit, which is also described in Chapter 2, is not necessary for understanding the specifications.

Chapters 3 and 4 are concerned with point-to-point protocols, that is, protocols which communicate information from one fixed location to another (and possibly back) using an unreliable channel. It is advisable to read Chapter 3 before Chapter 4. Chapter 3 contains a specification of three simple protocols for simplex communication. These are the alternating bit protocol (ABP), a protocol with positive acknowledgement and retransmissions (PAR), and the concurrent alternating bit protocol (CABP). They differ in the sense that PAR and CABP can handle "lossy" channels, while ABP can only correct mutilated messages. In the PAR protocol a timer is used to overcome lost messages, while in the CABP this is solved by sending a continuous stream of messages.

Chapter 4 contains three more-evolved point-to-point protocols from the class of sliding window protocols. These protocols are full-duplex. The simplest is the one-bit protocol, which has a sending window and a receiving window of size 1. Such a window is a buffer which contains messages that are not yet acknowledged. A more efficient protocol is the "pipelining with go back N" protocol, which has a sending window size greater than 1. A still more efficient protocol is the "nonsequential receive with selective repeat" protocol which also has a receiving window greater than 1.

Chapter 5 contains a protocol for communication in a distributed operating system; the Amoeba Transaction protocol. It is used in the Amoeba distributed operating system for information exchange between two processes.

Chapters 6 and 7 contain protocols for Local Area Networks. A simple token ring protocol and a simple ethernet protocol are specified in Chapter 6. The simple token ring protocol can be used in a network with a ring architecture. The simple ethernet protocol is a CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol, which is used on a bus architecture.

A specification of a token ring protocol based on the formal IEEE standard (IEEE 8802/4) is given in Chapter 7.

1.5 Bibliographical Notes

There are two books that are strongly related to this book: *Process Algebra* ([BW90]) and *Applications of Process Algebra* ([Bae90]). We consider [BW90] as a basic text on process algebra theory and [Bae90] as an example of the application of this theory, especially in the sense of correctness verification. In our view, this collection of protocol specifications demonstrating the practical use of process algebra will link up nicely with the two aforementioned books.

This book is completely self-contained, in the sense that it can be read without previous knowledge of the process theory explained in [BW90]. Nevertheless we expect that this book will stimulate readers to take an interest in the theoretical backgrounds of concurrency theory. One can take full advantage of the formal method used in this book by studying both the mathematical background from [BW90] and the verification examples of [Bae90].

The PSF language is defined in [MV90]. Other formal methods are described in [FJ92], [VDM88], [SDL84] and [ISO89]. The theory ACP was developed by Bergstra and Klop [BK84]. CCS and CSP are other theories for the specification of concurrent systems. They are described in [Mil89] and [Hoa85].

There are a number of other examples of the use of PSF and ACP concerning the specification of communication protocols. These are, amongst others [MW89], [Mau90], [MM90] and [VW92].

A comprehensive introduction to the OSI model is [HS88]. More general books on computer networks are [Tan89] and [Sch87].

CHAPTER 2
ALGEBRAIC SPECIFICATIONS IN PSF

G.J. VELTINK

2.1 INTRODUCTION

In this chapter we will focus on the specification language used throughout this book: PSF (Process Specification Formalism). We will discuss the mathematical origins of PSF as well as its syntax and semantics. The language itself will be clarified by using a running example, which gets more complicated as new language features are introduced. Apart from giving specifications in PSF we will also describe the implementations that make up the so-called PSF-Toolkit, such as the *term rewriting system* and the *simulator*.

The PSF-Toolkit also embodies a collection of frequently used specifications in the form of the PSF standard library. In this chapter we will explain which modules are part of the library and how they can be used. A full listing of the relevant modules from the PSF standard library can be found in Appendix A.

2.2 ACP

Before we turn our attention to PSF, we will give some information on ACP (Algebra of Communicating Processes). ACP is the theoretical foundation for the process part of PSF, and deserves some explanation as such.

The development of ACP was started in 1982 by J.A. Bergstra and J.W. Klop, at the Centre for Mathematics and Computer Science in Amsterdam. Compared with other concurrency theories like CCS, CSP and Petri Nets, ACP is most closely allied

to CCS. The main difference between ACP and the other approaches is the way in which the semantics is treated.

Most formalisms, like CCS, CSP and Petri Nets are based on one specific model of concurrency. ACP, however, is a theory based on algebraic methods†. The theory is defined by a set of axioms. The collection of possible models of this set of axioms contains most of the models for concurrency that have been proposed. By extending or restricting this set of axioms and by adding new operators, one can change the collection of possible models. This is a more general approach than the one which focuses on a single model. In many cases an algebraic approach towards verification has advantages over the model based approach.

2.3 THE HISTORY OF PSF

Several small tools for ACP were constructed between 1982 and 1986, but the lack of a unifying framework led to many inconsistencies between these tools. In the Autumn of 1987 the PAT (Process Algebra Tools) project was started. This project aimed at constructing an integrated environment of computer tools for studying concurrent systems, especially in the setting of ACP.

The first step towards this goal was the definition of PSF, a computer readable language to specify ACP processes. Although ACP uses data types in an informal way, it was felt that PSF should incorporate data types on a more formal basis. The language definition of PSF was completed in the Spring of 1988.

The definition of data types and modularization concepts in PSF is based on the algebraic specification language ASF (Algebraic Specification Language). The language ASF was developed at the Centre for Mathematics and Computer Science in Amsterdam.

The relation between ACP, ASF and PSF is given by Figure 2.1. The three blocks represent the *building blocks* from which PSF has been constructed.

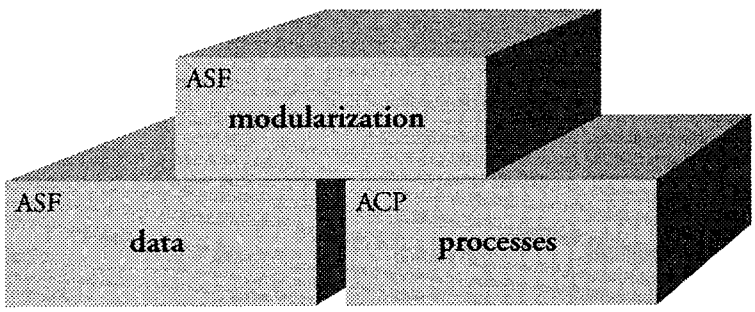

Figure 2.1 The constituent parts of PSF

† We notice that in this text book series two volumes on process algebra in the style of ACP have already appeared. One book describing the mathematical foundations of ACP and one collected volume describing different applications of ACP.