PROTOCOL CONVERSION--CORRECTNESS PROBLEMS*

Simon S. Lam
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract

Consider the problem of providing a logical channel for message exchange between two user processes in a network environment. When is protocol conversion needed? To answer this question, we first define a model of layered architectures. Specifically, three stepwise refinement rules are given. Any architecture that can be obtained by a sequence of applications of the stepwise refinement rules is said to be well-structured. We show that this class of well-structured architectures has several correctness properties. It is also very general and includes many well-known networking and internetworking architectures in the literature. Logical connectivity in such an architecture is defined recursively. As a result, to determine if a logical channel can be provided between two user processes, it is sufficient to examine peer protocols specified for each level of the architecture's hierarchy of processes one at a time. Thus the original problem reduces to the problem of determining if a set of processes will interoperate.

When protocol conversion is needed to achieve interoperability between processes that implement different protocols, how should it be done? How does one prove that a conversion is correct? What is meant by a correct conversion? We propose the use of projections and image protocols (previously developed by Lam and Shankar for protocol verification [10]) for specifying conversions and for reasoning about the correctness of conversions. Given two processes implementing different protocols P and Q, our objective is to find the largest protocol that is an image protocol of P as well as Q. The correctness of the conversion is a consequence of the correctness properties of image protocols.

There are several open problems. Most importantly, heuristics are used for finding the necessary image protocol for conversion. Although, an image protocol common to both P and Q can always be found, it may not be easy to find one with useful functionality. There are also some implementation and design issues to be addressed, such as: the construction of converters that are transparent and converters that add functionality to an image protocol common to P and Q.

1. Introduction

With the proliferation of network architectures and communication protocols, it becomes increasingly difficult to ensure that users connected to different networks can communicate. It may be argued that the solution to this problem is simply to agree upon one worldwide standard protocol architecture, say Open Systems Interconnection [17], or one internetting protocol, say TCP/IP or X.25/X.75, to be used by all suppliers of hardware and software [2, 5, 16]. In a recent article, Green [6] reviewed the protocol conversion problem from the architectural point of view, reviewed current ad hoc solutions, and argued convincingly that protocol conversions will be a permanent fact of life. He gave two main reasons. First, it is already too late to try to get everyone to adhere to the same standard. There is an installed base of over 20,000 IBM SNA networks, over 2000 DECnet networks, several hundred DoD TCP/IP networks, as well as many other vendor-specific networks. Second, convergence to a global standard implies that all tradoffs are understood and all inventions are made and assimilated, which is obviously not the case in the relatively young field of computer communications.

Even in the absence of architectural mismatches, the problem of achieving interoperability between different variants of the same protocol is a nontrivial task. Many protocol standards developed with the intention of fostering compatibility ended up as families of different standards [1, 2]. A standard as basic as RS-232 has many variants [11]. The data link protocol standard HDLC has many siblings: SDLC, ADCCP, LAP, LAP B, LAP B Multilink, etc. Even HDLC itself defines, in addition to a basic repertoire of commands and responses, a wide variety of optional capabilities for implementors to pick and choose from (thus fostering incompatibility between independently implemented versions of the protocol) [8, 9].

*This work was supported by National Science Foundation Grant No. ECS 83-04734.

To date, there have been a few protocol conversions attempted [6, 7]. However, there is no theory for understanding the protocol conversion problem. When is a conversion feasible? Correct? Useful? Acceptable? Or Successful? How do we synthesize a conversion?

We shall limit our attention to networks that implement layered protocol architectures and employ the method of encapsulation for the interaction of peer processes and maintaining data transparency. In today's networks, these requirements are quite reasonable and not at all restrictive. To tackle the protocol conversion problem in general, Green [6] discussed two kinds of mismatches that have to be considered: architectural mismatches and protocol mismatches. We shall address both.

In Section 2, we present a model of layered architectures. In this model, checking for architectural mismatches becomes checking for logical connectivity of the access path between two user processes. Since logical connectivity is defined recursively in the model, it is sufficient to examine peer protocols in an architecture one at a time. Architectures in this model have several correctness properties and are said to be well-structured. The class of well-structured architectures is very general and includes many well-known internetworking architectures. In fact, the concept of logical connectivity in this model eliminates any need to make a distinction between architectures for networking and internetworking. Section 3 illustrates this point. In Section 4, we address the problem of synthesizing a conversion between processes that implement different protocols. A formal model is introduced. The model is based upon protocol projections and image protocols previously developed by Lam and Shankar for protocol verification [10]. The model is well suited for reasoning about matches (or mismatches) between different protocols and also about the meaning of a correct conversion. Some design issues and open problems are discussed in Section 5.

2. A Model of Layered Architectures

Layering has been used as the basis for almost all computer network architectures. It is a powerful "structured programming" technique. Layering allows the allocation of network functions to different layers. Program modules in a layer can be implemented and subsequently modified independent of the details in the implementation of other layers. There is generally some duplication of functions in different layers of an architecture. However, it is fairly clear that any loss of efficiency due to functional duplication is outweighed by the advantages of modularity and hierarchical construction.

We shall present a model in which a layered network architecture is specified by a hierarchy of processes together with a set of protocols. Processes in the hierarchy are interconnected in some fashion by physical channels. We say that a set of processes will *interoperate* or a process-channel pair will interoperate if there is a protocol specified in the architecture for their interaction. (However, we do not need to know the functional specification of the protocol at this time.) The processes at one level of the hierarchy together with the protocols specified for them constitute one layer of the architecture.

An architecture is constructed recursively by applying several stepwise refinement rules. The class of architectures constructed in this fashion is general enough to include many well-known network and internetwork architectures. Our layering concept is not new. Our formal approach, however, is helpful for writing down unambiguously the requirements for processes to communicate across a network or across an interconnection of networks, for understanding what is meant by protocol conversion, and for determining if protocol conversion is necessary for a given architecture. (Our model is an elaboration of an earlier attempt by us in [9].) It is clear from reading Green's paper that a formal model of structured protocols is needed for understanding the protocol conversion problem.

Let us begin by considering Figure 1 showing two processes interconnected by a (full-duplex) channel. P_1 sends messages from a message set into the channel to be delivered to P_2. P_2 sends messages from a message set into the channel for delivery to P_1. In order for P_1 and P_2 to communicate, three protocols are necessary. Obviously, a protocol is needed between P_1 and P_2. Since P_1 and P_2 interact by messages, this protocol can be specified by specifying an event for the sending and receiving of each message by the processes. (Additionally, some internal events not associated with the sending and receiving of messages are also needed, e.g., timeouts.) Each event is specified by its enabling condition and the action of the process during the event execution. Such enabling conditions and actions define operationally the protocol between P_1 and P_2. For the purposes of this section, it is not necessary to know what the specific protocol functions are.

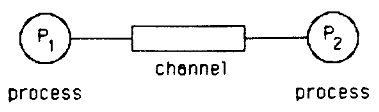


Figure 1. Two communicating processes.

Two other protocols are also necessary in Figure 1, one each for P_1 and P_2 to interoperate with their channel interfaces. The specification of these protocols depends upon the channel. If P_1 and P_2 reside in the same node and the intranode channel is provided by some interprocess communication (IPC) facility, these protocols are simply some IPC send and receive operations. If P_1 and P_2 are processes in different nodes interconnected by a physical link, then these protocols may be realized in the form of signals on a set of interface wires [11].

(In our figures, we use a rectangular box to denote a channel. Such a channel may be realized as a physical channel, intranode or internode, or as a logical channel. We use a dashed line to denote a logical channel and a solid line to denote an intranode physical channel.)

Next, we define some stepwise refinement rules with which we can generate protocol architectures beginning with the configuration in Figure 1.

Abstraction rule. A logical channel between P_1 and P_2 provided by a protocol in a lower layer replaces a channel. Applying the abstraction rule transforms the configuration in Figure 1 to the configuration in Figure 2.

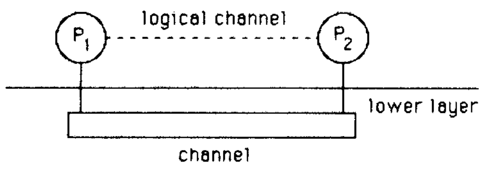


Figure 2. Replacing a physical channel with a logical channel.

Refinement rule. A channel (in Figure 1 or Figure 2) is refined to be a network of processes. The network can have the same configuration as the one in Figure 1. Or it can be a store-and-forward network with a mesh topology, a ring network, a coaxial cable network and others. The only requirement for it is the existence of a path between P_1 and P_2 consisting of a single process or a sequence of processes interconnected by channels as shown in Figure 3. Such a path may change adaptively for each message travelling between P_1 and P_2. It may be cyclic as long as it is finite. Given this refinement, P_1 (P_2) actually interacts with a process rather than interacting with a channel interface; such a process shall be referred to as a boundary process of the network realizing the channel. The boundary process that P_1 (P_2) connects to is assumed to reside in the same node as P_1 (P_2) and they interact by IPC operations. As part of this refinement step, a new protocol is specified for the set of processes in the network to interoperate. Protocols are also specified for each process in the network to interoperate with channels that it sends into or receives from.

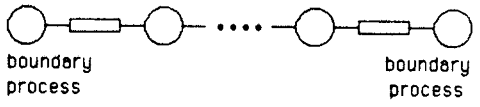


Figure 3 A network path.

Recursion rule. Each channel in a network of processes generated by the refinement rule can be replaced by a logical channel by applying the abstraction rule. Each channel in the hierarchy of processes under construction can be refined into a network of processes.

Starting with either the configuration in Figure 1 or the one in Figure 2, and by applying the above three rules in some order, a very general class of structured protocol architectures can be generated.

A protocol specified for a network of processes generated at a refinement step is said to be a *peer protocol*. (These processes are peers because they are at the same level of the process hierarchy being constructed, i.e., they are within the same layer in the protocol architecture.) These processes interact by sending messages to one another. The method of encapsulation is assumed in our model for the delivery of these protocol messages. Encapsulation is the method used in practically all layered architectures that we know of. It is described next.

Consider a peer protocol implemented by a network of processes such as shown in Figure 4. Messages given by an external user process to a boundary process in the network for delivery will be referred to as service data units (SDUs) of the protocol. Each SDU is treated as transparent data, i.e., a sequence of bits which is not interpreted by any process in the protocol. Each SDU received by a boundary process for delivery is "wrapped" with a header (and possibly a trailer as well). Control messages that processes in the peer protocol send to one another are encoded in the header. A wrapped SDU is referred to as a protocol data unit (PDU) of the peer protocol. Let SDU_x and PDU_x denote a SDU and a PDU of a protocol named x. Let H_x denote the header wrapped by protocol x on its SDUs. We define the following notation:

$$PDU_x = H_x(SDU_x)$$

We also define the unwrapping of a PDU by the two functions:

$$Header(PDU_x) = H_x$$

$$Data(PDU_x) = SDU_x$$

Note that some PDUs may not have any SDU wrapped inside and are created solely by the protocol processes for their own interaction. In this case, $Header(PDU_x) = H_x$ and $Data(PDU_x) = nil$. At each intermediate process along the path of a PDU, the PDU is unwrapped so that Header(PDU) can be read and interpreted. Header(PDU) may be updated prior to delivery to the next process in the path. At the destination boundary process, Data(PDU) is delivered to the destination user process.

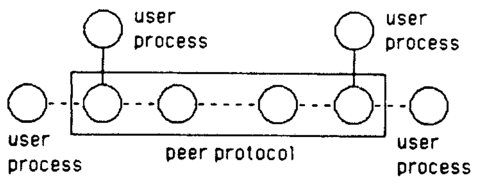


Figure 4 A peer protocol implemented by a network of processes

A peer protocol provides a logical channel between user processes in a higher layer and/or user processes in the same layer as the protocol. Note that processes within the same layer of an architecture may be connected by physical channels (intranode or internode) or by logical channels. Such a logical channel is itself realized by a lower-layer protocol or by a multilayered architecture. Thus, if two processes in protocol x are connected by a logical channel provided by a lower-layer protocol named y, then we have

$$SDU_y = PDU_x$$

and

$$PDU_y = H_y(SDU_y) = H_y(PDU_x) = H_y(H_x(SDU_x)).$$

As a message from P_1 travels to P_2 through the hierarchy of processes that realizes the channel in Figure 1, wrapping and unwrapping of different protocol headers will occur many times.

We state several properties of protocol architectures constructed by applying the stepwise refinement rules in any finite sequence.

Property 1 (*Freedom from unspecified receptions*).

(i) If a message is delivered to P_2, the message was sent by P_1.

(ii) Whenever a process unwraps a PDU, Header(PDU) is for a protocol specified in the architecture for a set of processes including this process.

Note that Property 1 is a safety property. Progress of messages and PDUs are not guaranteed. (That is why the protocol function and network topology need not be specified in applying the refinement rule.) Property 1 is also a fairly weak safety property because it does not preclude losses, duplications, and reorderings in the delivery of messages and PDUs. It is, however, a property of the architecture. Progress and the stronger safety properties are the responsibilities of the protocols which have been specified for logical connectivity in the architecture but not yet specified functionally. Property 1 implies that the method of encapsulation can be used safely in this class of architectures.

Property 1 can be proved in a straightforward manner using induction with the bottom layer in the architecture as the base case. It is necessary to assume that physical channels do not deliver PDUs with undetected errors. (If a PDU is corrupted by errors, the corruption is detected and the PDU is lost.) Also, processes and physical channels do not generate spurious messages.

Property 2 (*Physical connectivity*).

There is a finite sequence of processes between P_1 and P_2 and all adjacent processes in the sequence including P_1 and P_2 are connected by physical channels (intranode or internode).

This property is rather obvious since the initial configuration in Figure 1 has physical connectivity if the channel is realized by a physical channel. And each application of the stepwise refinement rules preserves physical connectivity.

Recall that a set of processes and channels in an architecture are said to interoperate if the same protocol is specified in the architecture for their interaction.

Definition. For two processes connected by a physical channel, as shown in Figure 1, we say that they are logically connected if (1) the processes interoperate, and (2) each process interoperates with its channel interface.

Definition. For two user processes connected by a network of processes, as shown in Figure 4, we say that they are *logically connected* if (1) the user processes interoperate, (2) the processes of the network interoperate, (3) each user process is logically connected to a boundary process in the network, and (4) there is a sequence of processes in the network with the boundary processes at the beginning and end of the sequence, wherein adjacent processes are logically connected.

Property 3 (*Logical connectivity*).

P_1 and P_2 are logically connected.

Again, this property is obvious given that the initial configuration in Figure 1 has logical connectivity. Logical connectivity is preserved by each application of the stepwise refinement rules.

A protocol architecture is said to be *well-structured* if it can be obtained by a sequence of applications of the stepwise refinement rules.

(It should be clear that although we use Figure 1 as the initial configuration for stepwise refinement, we can just as easily use Figure 5 as the initial configuration. By considering two processes instead of N processes, we can represent a network by a path in the refinement step. Thus, we avoid having to draw three-dimensional pictures for process hierarchies.)

A simple application of the above method to generate a well-structured architecture is illustrated in Figure 6. The hierarchy of processes is generated by 3 successive applications of the abstraction rule followed by an application of the refinement rule to get the processes labeled 5. The abstraction rule is then applied twice to each channel connecting processes labeled 5. As each rule is applied, peer protocols are specified; processes that are specified to implement the same peer protocol are labeled by the same number. Protocols are also specified for the interaction of process pairs or process-channel pairs joined by a solid line in the hierarchy picture.

Most readers probably recognize the architecture in Figure 6 to be the OSI architecture [17]. But why have we labeled the layers from the top down instead of from the bottom up? We have purposely deviated from convention here to make the following point. For an architecture to be well-structured, the magic number 7 and the symmetry of the OSI architecture are both unnecessary. (Of course, the OSI architecture is still a good thing to have as a framework for standards activities.) Take a look at Figure 7. Logical connectivity between the user processes is assured by the peer protocol at level 5 given that adjacent processes at level 5 are logically connected. However, logical channels between processes at level 5 can be provided by vastly different protocols or layered architectures, i.e., each logical channel can be realized by any well-structured architecture. For example, 7 and 7′ denote two different physical layers, 6 and 6′ denote processes that implement different data link protocols, 6″ denotes processes that implement not a data link protocol but a peer protocol that uses a network for message delivery. The architecture in Figure 7 is well-structured. It is hard to say how many layers it has since the network in the bottom layer may in turn have a multilayered architecture (obtained by a sequence of stepwise refinements). It also illustrates why it is more natural, for the class of well-structured architectures, to count layers from the top down rather than from the bottom up. The flexibility in the architecture of Figure 7 is not without some cost. For logical connectivity, three protocols are needed over intranode channels between processes in layer 5 and layer 6 as compared to a single protocol in Figure 6. And more such protocols are needed between layers 6 and 7.

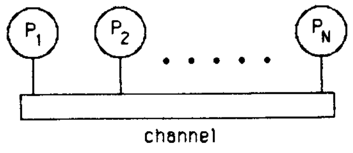


Figure 5. N communicating processes.

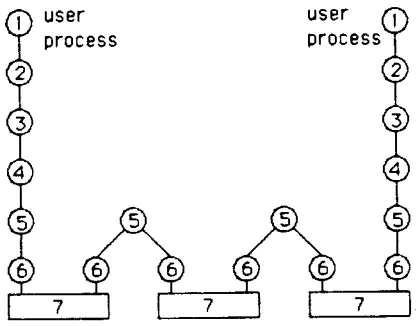


Figure 6. A seven-layer architecture

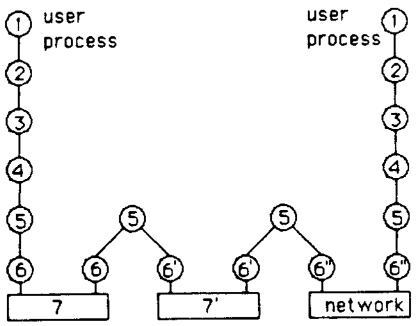


Figure 7. An asymmetric architecture.

As we shall see in Section 3, most internetworking approaches are aimed at making the combined architecture of the interconnected networks into one that belongs to the class of well-structured architectures, so that the internetwork has the above three properties, in particular, the logical connectivity property. To show that a given architecture is well-structured, it is sufficient to demonstrate a sequence of applications of the stepwise refinement rules whereby the architecture can be derived. We shall do so for some internetworking architectures in Section 3.

When is conversion needed in an architecture? Recall that a protocol architecture in this paper is specified by a hierarchy of processes (connected by both intranode and internode physical channels) together with a set of protocols; each protocol is specified for a subset of processes and channels in the hierarchy. The hierarchy provides physical connectivity while the protocols provide logical connectivity. Suppose we are examining the internetworking of two different protocol architectures. And suppose the two hierarchies of processes can be merged into a single hierarchy that can be derived by a sequence of applications of the stepwise refinement rules. (This should not be difficult starting with architectures that are layered.) Then protocol conversion is needed if the combined set of protocols do not provide logical connectivity to user processes. Note that our definition of logical connectivity is recursive. Thus, to determine how and where to perform conversion to achieve interoperability among subsets of processes, it is sufficient to examine the process hierarchy, one level at a time and one protocol at each level at a time. This is highly desirable. Because of this structured approach, the original protocol conversion problem reduces to the problem of achieving interoperability between processes that implement different protocols. We shall address this latter problem in Section 4.

3. Internetworking

Our concept of logical connectivity eliminates the need to make a distinction between architectures for networking and internetworking. The basic solution to providing logical connectivity in an internetworking architecture is self-evident from Figure 7. It is sufficient to have a peer protocol "spanning" the internet providing "host-to-host logical connectivity." Logical connectivity between pairs of processes in the peer protocol can then be provided by different networks. This approach makes use of the recursive definition of logical connectivity. It is the basis of both of the prominent internetworking architectures in use today. We illustrate these architectures in Figures 8-11.

In Figure 8, we show the basic architecture of an X.25/X.75 internetwork [16]. All the X.25 and X.75 processes interoperate to provide host-to-host logical connectivity. (The X.75 processes constitute two halves of a gateway and have some gateway functions as well.) Figure 9 illustrates a refinement of the basic architecture. The logical connectivity of channel A and channel E in Figure 8 are provided by data link and physical layer protocols, e.g., LAP B and X.21 respectively. The logical connectivity of channel C in Figure 8 is also provided by data link and physical layer protocols, e.g., LAP B Multilink and V.35 respectively. The logical connectivity of channel B and channel D in Figure 8 are provided by layered architectures of two different public data networks.

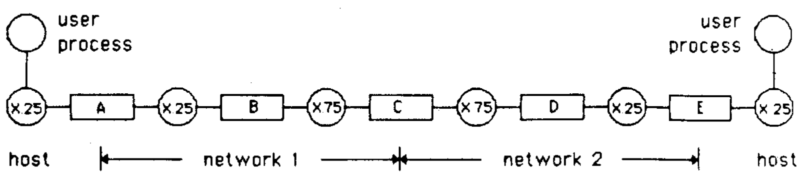


Figure 8. Basic architecture of X.25/X.75 internetworks.

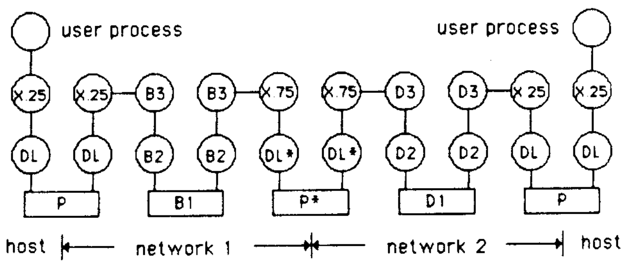


Figure 9. A refinement of the basic X.25/X.75 architecture.

In Figure 10, we show the basic architecture of a TCP/IP internetwork [5]. The IP processes interoperate to provide host-to-host logical connectivity. The logical connectivity between different IP process pairs can then be provided by different networks with different architectures. A refinement of the basic architecture is shown in Figure 11. For example, CSNET is a logical network that spans several physical networks and it uses the TCP/IP protocol for internetworking. As described in [4], network A in Figure 11 is Arpanet, network B is Telenet, host 1 is an Arpanet host, host 3 is a CSNET user, host 2 is both an Arpanet host and a CSNET user. (Please note that the example in Figure 11 represents the architecture of a single logical path in CSNET only and should not be mistaken as the entire CSNET architecture.)

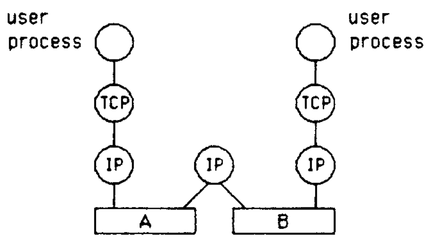


Figure 10. Basic architecture of TCP/IP networks

In both internetworking architectures, many additional protocols are needed between intranode processes to achieve logical connectivity between the user processes. In Figure 9, for example, protocols are needed for the interaction of the X.25 and B3 processes in network 1, and the interaction of the X.25 and D3 processes in network 2. In Figure 11, for example, protocols are needed for the interaction of the IP and B3 processes, and the interaction of the IP and A3 processes. These protocols are between processes residing in the same node and can be implemented with IPC operations. *We do not consider them to be protocol conversion problems.*

Referring back to the CSNET example and Figure 11, process B3 in host 3 is one that implements the X.25 packet-level protocol. The implementation of a protocol between this process and the IP process as a set of procedures in the operating system of the CSNET user is described in [4].

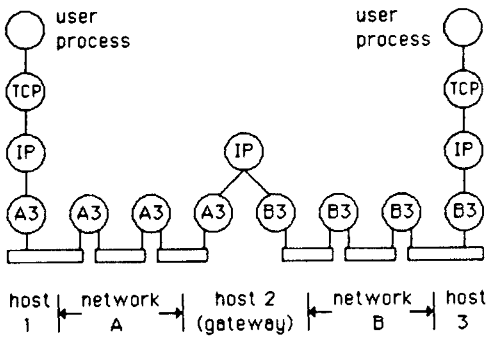
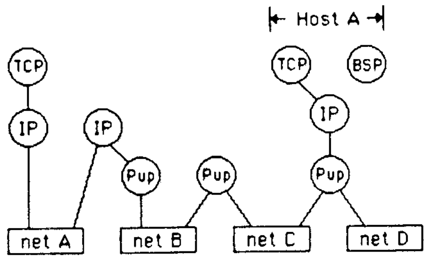


Figure 11. A refinement of the basic TCP/IP architecture.

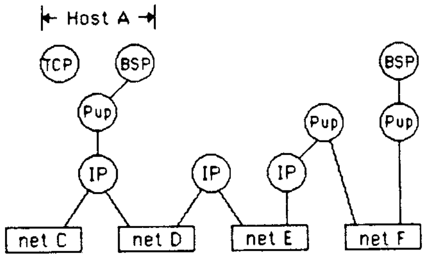
In both the above internetworking approaches, a single peer protocol (X.25/X.75 or IP) is used throughout the internetwork. In [15], the coexistence of two internetworking protocols, the IP protocol of Darpa and the Pup protocol of Xerox, is considered. Each protocol serves a subset of the community of users while some users implement both protocols. In such an environment, three solutions are possible. First, one can try to perform conversion between the two peer protocols to achieve some degree of interoperability between processes that implement these protocols. We shall return to this problem in Section 4. Second, one can apply the abstraction rule one more time to the overall architecture to insert a peer protocol above IP and Pup. This new protocol spans the entire internetwork while using the IP and Pup protocols for logical connectivity of its processes.

Shoch et al. proposed a third solution [15], called mutual encapsulation. Their solution illustrates that we do not have to use the same protocol architecture for all users of the interconnected networks. For some user pairs, the IP protocol is used as the peer protocol for internetworking with the Pup protocol providing logical connectivity to IP processes. In this case, we have $PDU_{Pup} = H_{Pup}(PDU_{IP})$. For some other user pairs, the Pup protocol is used as the peer protocol for internetworking with the IP protocol providing logical connectivity to Pup processes. In this case, we have $PDU_{IP} = H_{IP}(PDU_{Pup})$. Since both types of PDUs coexist in the internetwork, the method is called mutual encapsulation. This particular solution is consistent with our model of protocol architectures and our concept of logical connectivity between user process pairs.

An illustration of mutual encapsulation is shown in Figure 12. For communication between the TCP processes in Figure 12(a), the IP processes provide the host-to-host logical connectivity. For communication between the BSP processes in Figure 12(b), the Pup processes provide the host-to-host logical connectivity. Consider host A which has both IP and Pup processes. These processes exchange roles in the two different logical connections.



(a) Logical connectivity between two TCP processes.



(b) Logical connectivity between two BSP processes

Figure 12. An illustration of mutual encapsulation

4. Protocol Conversion

In a hierarchy of processes with a set of peer protocols specified for subsets of processes at each level of the hierarchy, protocol conversion is necessary when the user processes do not have logical connectivity. Given the recursive definition of logical connectivity for a peer protocol, we need only examine peer protocols at each level of the hierarchy of processes, one at a time, to determine if logical connectivity exists.

If the architecture is that of an interconnection of networks with different architectures, then it is likely that different protocols have been specified for some processes which must interoperate to provide logical connectivity between some other processes. We consider two cases: protocols for two processes, and protocols for more than two proceses.

First, if the number of processes is two and the processes implement different protocols, then conversion is *necessary*. The configuration in Figure 13(a) may occur anywhere in the architecture's process hierarchy, and will be replaced by the peer protocol in Figure 13(b) with the addition of a converter process. The objective of the converter process is to allow P_1 and Q_2 to interoperate. (The meaning of this conversion will be explained more formally below.)

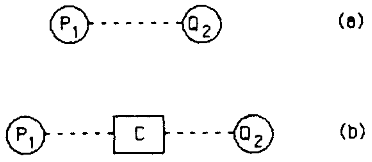


Figure 13. Conversion to achieve interoperability between two processes.

Second, if the number of processes is more than two, then conversion may be performed such as illustrated in Figure 14. (The logical problem is similar to that of conversion between two processes. But the practical problem is a lot harder.) Now, suppose the set of processes does partition into subsets of two or more processes as shown in Figure 14(a) and the processes within each subset interoperate. Then there is an easier solution that we have seen several times already: Exploit the recursive definition of the logical connectivity of a peer protocol, i.e., apply the abstraction rule to create a new peer protocol as shown in Figure 15. The new peer protocol provides the logical connectivity desired of the set of processes in Figure 14(a). The existing protocols provide the logical connectivity between processes in the new protocol.

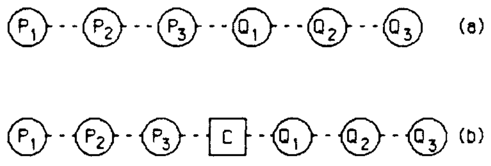


Figure 14. Conversion to achieve interoperability between two sets of processes.

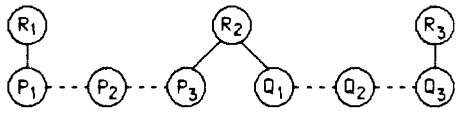


Figure 15. Avoiding conversion by applying the abstraction rule.

In what follows, we shall introduce a formal model that can be used for synthesizing conversions and for reasoning about the correctness of such conversions. For ease of exposition, we shall consider two-process protocols only. (Conceptually, our model and definitions can be extended to N-process conversion problems, but at the expense of much more complex notation and definitions.)

Let us consider peer protocols in which processes interact by exchanging messages. Protocol mismatches in this context refer to differences in the syntax and semantics of messages that can be sent and received in different protocols. The conversion problem can be stated as follows. Consider two protocols P and Q (see Figure 16). In the first protocol, the sets of messages that can be sent by entities P_1 and P_2 are M_1 and M_2 respectively. In the second protocol, the sets of messages that can be sent by entities Q_1 and Q_2 are N_1 and N_2 respectively. Now suppose we want P_1 to interoperate with Q_2 with the help of a protocol converter C_1 as shown in Figure 17(a). (The converter may be a process or a protocol layer in the path between P_1 and Q_2.) Obviously, one task of the converter is to perform syntax transformations of messages that P_1 and Q_2 can send to each other. The next question is: How does the converter map messages in M_1 and N_2 into N_1 and M_2 respectively? Messages that are related by the mapping have to be semantically equivalent in the two protocols. What does semantical equivalence mean? And how does one check it? Obviously, the level of functionality that can be achieved by the protocol conversion is determined by the subsets of semantically equivalent messages.

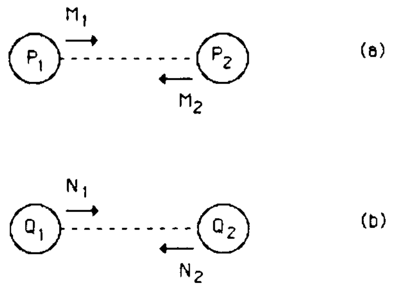


Figure 16. Protocols P and Q

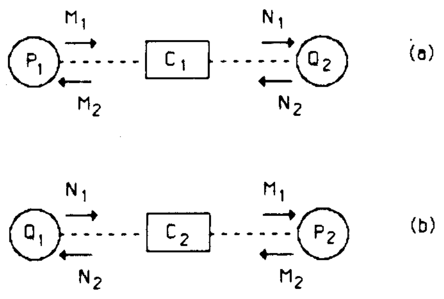


Figure 17 Protocol conversions

The semantics of messages in a message-passing protocol such as P or Q can be found in the reachability graph of the protocol. To avoid the use of reachability graphs (which may be infinite for many protocols) in our reasoning, we propose the use of projections and image protocols, previously developed by Lam and Shankar for protocol verification, for specifying conversions and for reasoning about semantical equivalence. Before proceeding further with the conversion problem, we shall digress and give a brief overview of image protocols and their properties. The reader is referred to [10] for details.

Our discourse shall be based on the abstract state machines model in [10] for protocols. Consider Figure 16(a). Let S_1 (S_2) denote the set of states of process P_1 (P_2). S_1 and S_2 may be finite or infinite. (Thus our model is applicable to protocols specified by state variables and a programming language notation, as in [12].) Each process is event-driven. Events of P_1 are specified for sending messages in M_1 and for receiving messages in M_2. Additionally, some events not associated with the sending and receiving of messages can be specified. These are called internal events and they are used to model timeouts and the peer protocol's interaction with its user processes. If the channels can have errors, then they are modeled as processes; errors are modeled by specifying internal events for these channel processes.

The state of the protocol is described by the four-tuple $(s_1,s_2;\mathbf{m}_1,\mathbf{m}_2)$ where s_1 is the state of P_1 and $\mathbf{m}_1$ is the sequence of messages in the channel from P_1 to P_2; s_2 and $\mathbf{m}_2$ are similarly defined. Let G denote the global state space of the protocol. At any state g in G, a set of events are enabled, the occurrence of one of these enabled events, chosen nondeterministically, will take the protocol to some state h in G. Given an initial state g_0, let R denote the protocol's reachability graph.

So far in this paper, the meaning of "specifying" a protocol for a set of processes is very weak. The existence of such a protocol in the architecture implies that the processes interoperate or "understand the meanings of each other's messages." Specifying the message sets and event sets of P_1 and P_2 will now define operationally the protocol's behavior. It is generally useful to specify a protocol functionally rather than defining its behavior operationally. The functional specification of the protocol can be in the form of safety and liveness assertions about the behavior of the protocol. Abstractly, a safety assertion is a predicate on G. The safety assertion is an invariant property of the protocol if the safety predicate is true at g_0 and at each state of the protocol reachable from g_0. A liveness assertion is a predicate on the set of paths in G. A liveness assertion is a property of the protocol if the liveness predicate is true on all paths in R. Verification of these properties may be carried out by state exploration (in the case of a small finite R), or by proof techniques for parallel programs [13].

We are now in a position to introduce the concept of the *resolution* of a protocol that is central to the theory of projections. Consider again the protocol in Figure 16(a). Since the protocol state is the tuple $(s_1,s_2;\mathbf{m}_1,\mathbf{m}_2)$, the resolution of the protocol is, roughly speaking, given by the number of states in S_1 and S_2 and the number of messages in M_1 and M_2. Suppose the protocol performs many functions, but we are only interested in verifying an assertion about the protocol's performance of one or a subset of its functions. Then, in the verification, the observable resolution of the protocol can be much smaller than the protocol's actual resolution. This gives rise to the idea of constructing an image protocol with a resolution lower than that of the original protocol for verifying the assertion.

Consider the image protocol in Figure 18. Let S'_1 and S'_2 be the sets of states of P'_1 and P'_2. They are obtained as follows. Partition S_1 and S_2. Each partition subset of states in S_i defines a single state in S'_i, for i = 1 and 2. (We shall sometimes refer to this operation as aggregation.) How to do the partitioning requires ingenuity and insights into the meanings of the process states. If the state of P_i is given by the values of a set of state variables, then one way to realize a partitioning of S_i is by retaining in the image protocol a subset of the state variables in the original protocol. Generally, the meanings of state variables in protocols specified by a programming language notation are more self-evident than the meanings of states in finite state machines. (See [10, 12] for illustrations.)

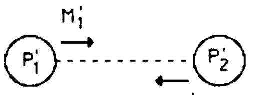


Figure 18. Image protocol

Aggregating states in S_i to define states in S'_i gives rise to an equivalence relation on the message sets M_1 and M_2. Specifically, two messages in M_1 are equivalent if their receptions cause identical state changes in the image state space S'_2; a similar definition applies to messages in M_2. Furthermore, messages in M_i whose receptions do not cause any state change in the image state space of the receiving process are said to have a null image. There are more definitions and details necessary for the construction of image protocols. We shall refer the reader to our previous work for a complete treatment [10].

By its very definition, an image protocol has a resolution lower than that of the original protocol. Given an image protocol, suppose that a second image protocol is obtained by partitioning S'_1 and S'_2 of the first image protocol. Then, the second image protocol has a lower resolution than the first. Thus, we can talk about a sequence of image protocols with decreasing (or increasing) resolution.

It is important to note that an image protocol is specified like any other protocol, i.e., it can be implemented. We shall next state two very useful properties of image protocols from [10]:

Image Protocol Property 1. If a safety assertion holds invariantly for an image protocol, it also holds invariantly for the original protocol.

Image Protocol Property 2. For an image protocol with well-formed events, a liveness assertion about the image protocol holds for the image protocol if, and only if, the same assertion holds for the original protocol.

Again, we shall refer the reader to [10] for the definition of well-formed image events and for proofs of these properties. The proof of Image Protocol Property 2 assumes fair scheduling of events in the protocol processes and that each message sent into a channel will eventually be received or deleted (also a fairness assumption).

Consider a protocol that is an image protocol of protocol P and also protocol Q. Such a protocol is called a *common image protocol* of P and Q. We can now state our basic approach to solving the protocol conversion problem formulated earlier: Find a common image protocol of P and Q having the highest resolution. (By contrast, in protocol verification, we desire an image protocol with the lowest resolution adequate for proving an assertion.) Suppose such an image protocol common to both protocols P and Q is found. Let it be as specified in Figure 18. Let us consider the protocol conversion in Figure 17(a). What C_1 provides is a mapping function. A message sent by P_1 with a nonnull image, say m' in M'_1, is transformed by C_1 into a message in N_1 with image m' for delivery to Q_2, similarly, messages in N_2 are mapped into M_2. What this conversion accomplishes is an implementation of the image protocol. If this image protocol is one common to both P and Q with the highest resolution, then it implements the most functionality that is common to both P and Q. By Properties 1 and 2 of image protocols, the logical properties of the peer protocol obtained by conversion are also logical properties of each of P and Q. Thus the correctness of the conversion is guaranteed. It is also a very rigorous definition of correctness. The advantage of this approach is that it is possible to establish semantical equivalence without having to reason with or about any of the reachability graphs.

This approach requires a heuristic search for an image protocol with useful properties. Note that an image protocol common to both can always be found. Specifically, if we aggregate the set of states in each process in Figure 16 to a single state, we have an image protocol common to P and Q. But it is an image protocol with no useful property. Any difficuty in the heuristic search may not necessarily be the fault of the method however; it could be due to the fact that protocols P and Q have very little in common to begin with. Obviously, the job of synthesizing a conversion will be easier if protocols P and Q are quite similar to each other, such as, they are variants of the same protocol. The example that we have tried using the above method is that of a conversion between a version of IBM's BSC protocol and an alternating-bit (AB) protocol. BSC has the basic structure as AB but differs from it in a number of details.

5. Discussions

Given a conversion so that processes P_1 and Q_2 interoperate, the peer protocol including the converter is considered by us to provide logical connectivity. The common image protocol, however, may not have enough functionality for a particular application. One way to add functionality to the peer protocol in Figure 17(a) is to add a state machine in C_1. For example, consider the BSC protocol. Although it has the same structure as the AB protocol, BSC data messages do not carry a sequence number (0 or 1). But in a AB receiver, a sequence number is expected in each data message received. The common image protocol will not have a sequence number in its data messages. As a result the common image protocol does not have the desired logical property of the AB protocol. This shortcoming can be remedied by a two-state finite state machine in the converter that inserts a sequence number into each message that it sends to the AB receiver. In this case, the set of messages sent by the converter has a higher resolution than the set of messages it receives. To preserve the correctness of the conversion, as defined by us, we will insist that the image protocol that is common to protocols P and Q is also an image protocol of the peer protocol including the converter (Figure 17(a)). This line of reasoning can be extended so that even more functionality is added to the common image protocol by replacing the converter process in Figure 17(a) by a peer-protocol converter as shown in Figure 19. We are now faced with a protocol construction problem very similar to those studied in [3, 14], i.e., given a protocol, how do we construct from it a protocol with more functionality such that the given protocol is an image protocol of the constructed protocol. We sometimes refer to this problem as *inverse projection*.

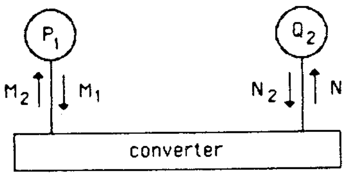


Figure 19. Replacing the converter by a peer protocol

Should the conversion be transparent to the processes P_1 and Q_2 in Figure 17(a)? Recall that messages in M_1 with a nonnull image are translated by C_1 into messages in N_1. What about those messages in M_1, if any, that have a null image in the common image protocol? In Section 4, we have implicitly assumed that P_1 will not send these messages. Hence the conversion is, in general, not transparent to P_1. P_1 is aware that it is interacting with a partner implementing a different protocol and that certain messages should not be sent. Such a requirement may be an acceptable price to pay to achieve logical connectivity. It is possible for the conversion to be transparent to P_1 if the common image protocol satisfies the following condition: The sending of each null-image message does not cause any state change in S'_1. In this case, any null-image message sent by P_1 can simply be discarded by the converter. If this condition is not satisfied, then transparency will have to be achieved by adding functions to the converter as discussed earlier.

In conclusion, we have shown that the class of architectures generated by our stepwise refinement rules is very general and includes many well-known networking and internetworking architectures. We show that these architectures have several correctness properties and they are said to be well-structured. Our concept of logical connectivity eliminates the need to distinguish between architectures of networks and internetworks. The recursive definition of logical connectivity reduces the protocol conversion problem, as posed by Green [6], to the problem of achieving interoperability between processes that implement different protocols. The theoretical framework of projections and image protocols is found to be very suitable for reasoning about semantical equivalence of messages in different protocols and about the correctness of a conversion. However, many design issues remain to be addressed.

Acknowledgements. This paper has benefited from comments by Ken Calvert and Ambuj Singh. The use of a two-state finite state machine in a converter between the BSC and AB protocols was suggested by Ken Calvert.

6. References

[1] ANSI/IEEE Standards for Local Area Networks, 802.2, 802.3, 802.4, and 802.5, IEEE, 1984.

[2] CCITT, Draft Revised CCITT Recommendation X.25, *Computer Communication Review*, January/April 1980.

[3] C. H. Chow, M. G. Gouda, and S. S. Lam, "A Discipline for Constructing Multiphase Communication Protocols," *ACM Transactions on Computer Systems*, November 1985.

[4] D. Comer and J. T. Korb, "CSNET Protocol Software: The IP-to-X.25 Interface," *Proc. ACM SIGCOMM '83 Symp.*, University of Texas at Austin, March 1983.

[5] DoD Standard Internet Protocol and DoD Standard Transmission Control Protocol, *Computer Communication Review*, October 1980.

[6] P. Green, "Protocol Conversion," *IEEE Trans. on Communications*, March 1986.

[7] I. Groenback, "Conversion Between TCP and ISO Transport Protocols as a Method of Achieving Interoperability Between Data Communications Systems," *IEEE J. on Sel. Areas Commun.*, March 1986.

[8] S. S. Lam, "Data Link Control Procedures," in *Computer Communications*, Vol. 1, ed. W. Chou, Prentice Hall, Englewood Cliffs, 1983, pp. 81-113.

[9] S. S. Lam, *Principles of Communication and Networking Protocols*, IEEE Computer Society Press, 1984, Chapter 1.

[10] S. S. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Trans. on Software Engineering*, July 1984.

[11] J. E. McNamara, *Technical Aspects of Data Communications*, Digital, 1977.

[12] A. U. Shankar and S. S. Lam, "An HDLC protocol specification and its verification using image protocols," *ACM Transactions on Computer Systems*, Vol. 1, No. 4, pp. 331-368, November 1983.

[13] A. U. Shankar and S. S. Lam, "Time-Dependent Communication Protocols," *Principles of Communication and Networking Protocols*, (ed. S. S. Lam), IEEE Computer Society Press, 1984.

[14] A. U. Shankar and S. S. Lam, "Construction of Sliding Window Protocols," Technical Report TR-86-09, Dept. of Computer Sciences, University of Texas at Austin, March 1986.

[15] J. F. Shoch, D. Cohen, and E. A. Taft, "Mutual Encapsulation of Internetwork Protocols," *Computer Networks*, 1981, pp. 287-300.

[16] M. S. Unsoy and T. A. Shanahan, "X.75 Internetworking of Datapac and Telenet," *Proc. 7th Data Comm. Symp.*, 1981, pp. 232-239.

[17] H. Zimmerman, "OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980.