Adaptors for Protocol Conversion*

Kenneth L. Calvert and Simon S. Lam
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

We propose the use of ***adaptors*** for protocol conversion in heterogeneous data networks with layered architectures. An adaptor is a form of protocol converter enabling a peer component of one protocol to simulate a peer of a different protocol. Adaptors have certain architectural advantages over other conversion approaches; in addition, the correctness definition for an adaptor is simpler than for other converters. We illustrate the approach with an example involving three different connection management protocols.

1 Introduction

Enabling users of different computer networks to communicate in spite of differences in network architectures and protocols is an important and difficult task. In this paper we consider the problem of enabling peer entities of different protocols to cooperate and to provide a useful service. The basic problem is shown in Fig. 1. P_0 is a peer of protocol P, while Q_1 is a peer of a different protocol Q, the counterpart of P in another network. The two networks are interconnected at a lower level, giving "logical connectivity" [13] between P_0 and Q_1, i.e. messages can travel between between them. We want to implement a service between the two users by making it possible for the two peers to cooperate with each other. However, in general the syntax and semantics of the messages of the two protocols differ. There are two choices: modify one or both of the peers so that they use the same protocol, or perform some ***translation*** on the messages they send and receive. A module that performs such a translation is called a ***protocol converter***.

We propose an architecture for protocol conversion based on the notion of *adaptors*. An adaptor is a protocol converter enabling a peer of one protocol to simulate a peer of a different protocol. In this paper we point out several advantages of adaptors over other architectures for conversion. One of these is that the problem of deriving an adaptor algorithmically, as a ***quotient*** of known specifications, is simpler than the corresponding problem for more general converters.

*work supported by the National Science Foundation under grant number NCR-8613338

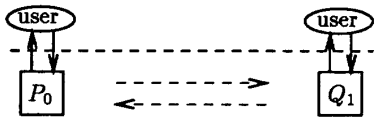

Figure 1: Protocol peer mismatch

The paper is organized as follows. In the next section, we define the problem, describe architectural advantages of the adaptor approach, and discuss some necessary conditions for its application. In section 3, we consider the problem of deriving and/or verifying the correctness of different types of converter. Finally, we illustrate the use of adaptors with an example involving connection management protocols.

2 Conversion Architectures

Consider a generic internetworking problem involving networks with different layered architectures (Fig. 2). P_0 and P_1 are peers of protocol P; using a lower-level service P_N, they provide a service to users of the left network. In another network, peers Q_0 and Q_1 use Q_N to provide a similar service. We would like to provide a service to users of the networks without extensive modification of the existing components. In particular, the ***upper interfaces*** of protocols P and Q should remain unchanged, so that the users (which may be peers of a higher-level protocol) need not be modified. While the following discussion applies to any level in the protocol hierarchy, the conversion problems of primary interest today arise at the Network and Transport layers [8, 9].

In some cases the required service can be provided simply by "bridging" the upper interfaces of P_1 and Q_0, so as to concatenate the services of P and Q. This solution, based on the mapping of *service primitives* between interfaces [1, 2, 6], is nice because the protocol peers P_0 and Q_1 themselves need not enter the picture. However, this approach may be inadequate if the particular service to be provided is ***end-to-end*** in nature, i.e., involves synchronization of the two end-user interfaces. For example, suppose protocol P provides confirmation to the user that a message has been queued for delivery to the remote user. The concatenation approach changes the end-to-end nature of this confirmation, which is given when the message has been queued at P_1, not Q_1.

Efficiency is also a concern with this method, as each message must be handled by two protocol peers and a bridge each time it crosses a boundary between architectures. (Fig. 2 depicts the two networks as adjacent; in a more general case, they may be separated by several intervening networks.)

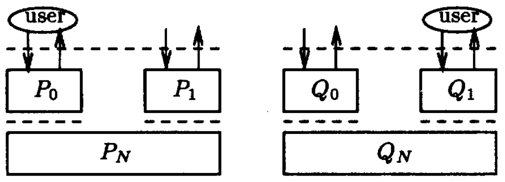

Figure 2: Layered networks to be connected

In order to provide a true end-to-end service, P_0 and Q_1 must communicate with each other; in that case some *translation* of protocol messages is required. In the conventional approach [9, 20], peers P_1 and Q_0 are replaced by a *gateway-converter* (Fig. 3), which translates messages received from P_0 via P_N and forwards them to Q_1 via Q_N, while also performing the reverse translation in the other direction. Such a converter is typically envisioned as a shared resource, supporting simultaneous conversions between many pairs of users; as such, it must deal with buffering and resource allocation issues on both sides.

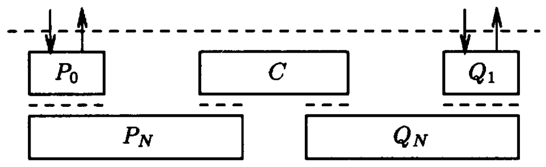

Figure 3: Gateway-type converter

The gateway-converter is a potential bottleneck for trans-network communications. In general, translation of messages between P_0 and Q_1 will require maintenance of some state information in the converter; this makes the gateway node an additional point of failure for the communication between P_0 and Q_1. As a practical matter, such state information should be located in a single node, so adding more gateways at a given network boundary cannot increase reliability for any particular communication.

The above problems are multiplied when the P and Q networks are not adjacent. If conversion is performed at network boundaries, a message traveling from P_0 to Q_1 must be translated at each intervening network. (With a "half-gateway" approach, in which each protocol is translated first to a common intermediate form and then to the next network's protocol [7], each intervening network adds *two* translations.) Each additional converter on the path also adds another point of failure to the conversion system. These and other issues related to gateway-converters are discussed in [16].

Fig. 4 shows a different approach, in which the translation of protocol messages of P_0 and Q_1 does not take place at the network boundary. The lower-level problem—provision of logical connectivity between P_0 and Q_1—is solved separately, resulting in an internetwork service through which P_0 and Q_1 can exchange messages directly. This lower-level problem may be solvable without requiring translation of messages if the services P_N and Q_N are not end-to-end in nature [2, 3]. The most common approach is encapsulation, i.e. adding a sublayer that concatenates the existing services; the canonical example is the DARPA Internet Protocol [17]. (Note that at *some* lower level, some form of gateway between networks will be required. The point is that it should not contain any end-to-end functionality.)

The translation of P and Q messages is handled in Fig. 4 by an *adaptor*, D, which is colocated with Q_1. The defining characteristic of an adaptor is that it enables a peer of one protocol (in this case Q_1) to simulate a peer of a different protocol (in this case P_1). That is, the behavior of Q_1 and D, as observed by the user and the lower-level service, is indistinguishable from the behavior of P_1. If the service provided by N is adequate for protocol P to function correctly, it follows that the system provides the same service as P.

Adaptors avoid many of the architectural shortcomings of gateway-converters noted above. Because an adaptor resides in the end node, it is not a shared resource and need not deal extensively with resource-management issues; instead of serving users throughout the network, one adaptor serves one peer. State information for the conversion is confined to the end nodes, so no additional points of failure are introduced, and reliability-enhancing features of the lower-level service (e.g. alternate routing) can be taken advantage of.

Moreover, the number of adaptors (and translations of each message) is fixed and independent of the number of intervening networks between P_0 and Q_1.

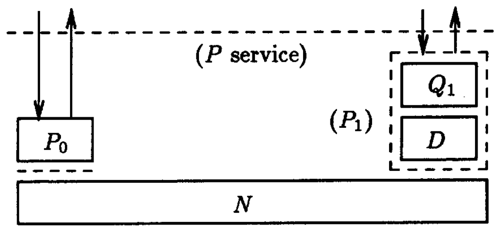

Figure 4: Adaptor for peer Q_1

Another advantage of the adaptor approach is its flexibility. Figure 4 is actually a special case of the architecture shown in Fig. 5; in this more general case, two adaptors are used, and both peers are made to simulate peers of a third protocol. Thus, conversion among three or more protocols can be achieved by adapting each to some "target" protocol. The cost to include an additional protocol is the cost of an adaptor for that protocol. With gateway-converters, conversion among multiple protocols involves multiple translations and/or gateways.

Suppose a basic connection management service, Z, is implemented by peers Z_0 and Z_1 using the service N. By defining adaptors for a variety of other connection management protocols, we can provide the service of Z with peers of any two of those protocols. If P and Q are other connection management protocols, and we have adaptors D_0 and D_1, such that D_0 and P_0 together satisfy the specification of Z_0, and Q_1 and D_1 together satisfy Z_1, then we can use P_0 and Q_1 to implement the basic service provided by Z (Figure 5).

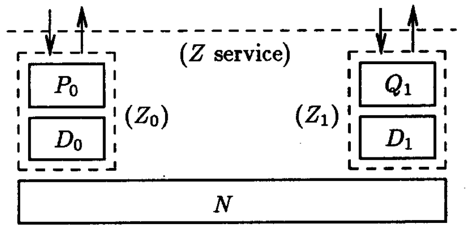

Figure 5: Conversion between P and Q via Z

Various possibilities exist for the form of P and Q, and the corresponding adaptors. Both P and Q might be very similar to Z, in which case the primary function of the adaptors is to perform syntactic translation of messages. Such an adaptor requires little internal state information of its own; this is the most desirable case. If either P or Q is semantically different from Z, the corresponding adaptor may have to do quite a bit of work; in particular, the adaptor may have to maintain some state information. Finally, if P or Q is semantically very different from Z, no adaptor may be feasible.

It seems that a conversion architecture based on adaptors is an attractive alternative to gateway-converters. However, some conditions must be satisfied in order for the use of adaptors to be feasible. First, the upper (user) interface of the existing protocol entity must be similar to that of the entity to be simulated. In terms of Fig. 4, Q_1's upper interface must be adequate to simulate P_1's. If the interface is defined in terms of abstract ***service primitives*** (e.g., as in [12]), then Q_1 must implement those service primitives implemented by P_1. This is not an unreasonable condition if both protocols are intended to implement the same type of service.

The second condition is that an adequate lower-level service must be provided between the endpoints. In Fig. 4, the service N must be sufficient for protocol P to function correctly; otherwise we cannot claim that the conversion service implements the service of P. This condition suggests that in general, a protocol requiring less of its underlying service is the better choice as "target" protocol. If Q works with an underlying service that can lose and reorder messages, while P is correct only if messages are received in the order sent, then the P-peer should be adapted to Q.

The third condition for use of adaptors concerns the interface between the existing peer and the underlying service. This interface must be explicitly implemented, and must be sufficiently accessible to allow an adaptor implementation to be interposed between the peer and the service. We can think of the service primitives exchanged between peer and lower-level service as being "filtered" through the adaptor module. A facility for the insertion of such "filters" into a communications system is a feature of some operating systems.

3 Formal Models

Our work has dealt with the correctness of conversion systems and methods for reasoning about whether a conversion between two given protocols is possible [4, 5, 13]. It turns out that, in addition to their architectural benefits, adaptors also have a significant advantage in this area.

Consider a class of ***state machines***, which can be interpreted as protocol specifications. The binary composition operator $\|$ maps machines B and C to $B \| C$, which models the system formed when B and C interact. The relation **sat** on machines corresponds to the relationship between a correct implementation and a specification: B **sat** A means that B is an adequate replacement for A as a component of any system. Composition is monotonic with respect to **sat**: B **sat** A implies $(B \parallel C)$ **sat** $(A \parallel C)$ for any A, B, and C.

Such a model allows us to reason *hierarchically* about protocol architectures. Verifying the correctness of a protocol comprising peers P_0 and P_1 and lower-level service P_N consists in establishing the relationship $(P_0 \parallel P_N \parallel P_1)$ **sat** P_S, where P_S is the required service. Correctness of protocol P is independent of the particular protocol used to implement the service P_N.

In the context of such a formalism, the *quotient problem* is the following. Given A and B, find C such that $(B \parallel C)$ **sat** A. Merlin and Bochmann [14] considered the problem of defining "submodule specifications" (essentially the quotient problem) and gave a solution, using a model dealing with finite sequences of events (*safety* properties). That model, however, is not adequate for *progress* properties such as absence of deadlock. In [4], we showed how protocol converters can be defined as quotients and gave an algorithm for solving a class of quotient problems, using a model capturing both safety and progress properties. As discussed below, the definition of an adaptor as a quotient is simpler than that of the corresponding gateway converter; this makes it easier to derive an adaptor algorithmically or verify the correctness of one obtained heuristically.

Referring again to Fig. 4, D can be defined as the quotient of P_1 and Q_1; that is, for a correct D we have $(D \parallel Q_1)$ **sat** P_1. From the correctness of protocol P, we have $(P_0 \parallel P_N \parallel P_1)$ **sat** P_S. If we also know that N **sat** P_N, then it follows from the monotonicity of composition that

$$(P_0 \parallel N \parallel (D \parallel Q_1)) \textbf{ sat } P_S.$$

On the other hand, the gateway-converter C of Fig. 3 is defined by the relation

$$(P_0 \parallel P_N \parallel C \parallel Q_N \parallel Q_1) \textbf{ sat } S_c$$

where S_c is an explicit specification of the minimum service acceptable to the users of the conversion. Thus, solving the quotient problem for D involves only Q_1 and P_1, while the same problem for C involves five different specifications, one of which (S_c) may not occur in any of the original systems.

Before presenting an example, we briefly explain our model, in which services and protocols are specified as finite-state machines (FSMs). The behavior of a protocol component (i.e., a peer or service) is defined in terms of *actions*. Actions represent a handshake or exchange of information across the interface, and correspond to the abstract *service primitives* used to define communications services. The "rendezvous" model is used; an action is not under exclusive control of either side of the interface, but can occur only when *both* sides of the interface are ready for it to occur. Occurrence of an action may result in a change of internal state, although such changes cannot be observed by the environment.

The behavior of a component B is specified as a finite state machine comprising a finite set Σ_B of actions, a finite set S_B of states, an initial state $b_0 \in S_B$, an external transition relation, and an internal transition relation. The set Σ_B defines B's interface. External transitions define how the state of the component is affected by interaction with its environment; the presence of an external transition $b \xrightarrow{e} b'$ means that whenever the state of the machine is b, the action e is enabled; if e occurs, the state changes to b'. Internal transitions $b \rightarrow b'$ define state changes that may occur under local control; they cannot be observed or prevented by the environment. In this paper, internal transitions are present only as a result of composition of FSMs, and represent hidden interactions between sub-components of a composite machine.

The machine formed by composing machines B and C is defined as follows. The set Σ of actions of $(B \parallel C)$ is the set of all actions in Σ_B or Σ_C but not in both. The state set of $(B \parallel C)$ is formed from the Cartesian product of S_B and S_C; states are denoted as pairs $\langle b, c\rangle$. More precisely, the state set S is the subset of $S_B \times S_C$ containing all states reachable from the initial state $\langle b_0, c_0\rangle$ via a sequence of zero or more external or internal transitions of $(B \parallel C)$. The composite $(B \parallel C)$ has an external transition $\langle b, c\rangle \xrightarrow{e} \langle b', c'\rangle$ iff $e \in \Sigma$ and either ($b = b'$ and $c \xrightarrow{e} c'$ in C) or ($b \xrightarrow{e} b'$ in B and $c = c'$). The internal transition $\langle b, c\rangle \rightarrow \langle b', c'\rangle$ is present in $(B \parallel C)$ iff ($b = b'$ and $c \rightarrow c'$ in C) or ($b \rightarrow b'$ in B and $c = c'$) or there exists $e \in (\Sigma_B \cap \Sigma_C)$ such that $b \xrightarrow{e} b'$ in B and $c \xrightarrow{e} c'$ in C. Thus, composition synchronizes and "hides" matching external transtions.

A *path* of a FSM B is a finite sequence of alternating states and transitions of B, starting with (and containing at least) the initial state. A *trace* is a (possibly empty) sequence of actions. Each path of B defines a trace, namely the sequence of actions associated with its state transitions. A FSM defines a set of paths and a corresponding set of traces; the latter represents all of its possible finite behaviors, as they would be observed by its environment.

For trace t and set $X \subseteq \Sigma_B$ of actions, the pair (t, X) is a *refusal* of B if and only if there is a path in B corresponding to t which terminates in a state in which (i) no internal transition is enabled, and (ii) no action in X is enabled. We denote the set of all refusals of

B by $ref.B$. The operational interpretation of a refusal (t, X) is that B *may* deadlock after performing t in any environment in which only actions in X are enabled after t (see also [10]). Note that $ref.B$ is prefix-closed for traces and subset-closed for sets of actions, i.e. if $Y \subseteq X$ and $(t, X) \in ref.B$ then $(t, Y) \in ref.B$.

For FSMs B and C, B **sat** C if and only if $ref.B \subseteq ref.C$. This definition says that every possible (bad) behavior of B is possible for C (every trace of B is a trace of C), and B is guaranteed to do everything C does (B can refuse X after any t only if C does).[1]

4 An Example

Turning now to our example, let us consider a simple *connection management* service, enabling two users to reach a "synchronized" state. Such a state is a prerequisite for many tasks, including data transfer. We describe two protocols, P and Q, each implementing the same service. Then we introduce a third protocol, Z, as the "target" protocol; using adaptors for the P and Q peers, the service of Z can be provided.

In our simple service, one user (the *caller*) initiates the communication, while the other (the *listener*) may either accept or refuse the request for a connection. Once established, the connection persists until the caller requests that it be taken down. The connection service is defined in terms of service primitives *Connect Request* (CR), *Connect Indication* (ci), *Connect Response* (cs), *Connect Confirm* (CC), *Disconnect Request* (DR/dr), and *Disconnect Indication* (DI/di)[2] Capital letters denote primitives of the caller interface, lower case denotes those of the listener.

The behavior of the service is specified by the FSM shown in Fig. 6. When "CR" occurs at the caller, the listener is notified of the request via "ci." If the user accepts ("cs"), the successful open is confirmed with "CC." A refused connection ("dr") results in a "DI" at the caller. The square state (0) is the *closed* state, in which the connection consumes no resources. Upon reaching the double-circle state, the connection is established, and can be used for data transfer or other purposes. Note that a new connect request will not be accepted from the caller until after *both* parties are informed that the previous connection has ended.

Implementing such a service over unreliable transmission media, where messages are subject to loss, duplication, delay, etc., is not a trivial problem. Because of the possibility of delayed messages, naive protocols are liable to result in "half-open" connections, in which one side reaches the "synchronized" state while the other remains in the closed state [15, 19]. Our protocols must prevent such anomalies *without* requiring either peer to maintain information about the state of the other across connections.

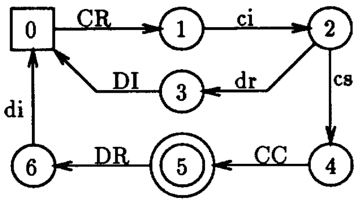

Figure 6: Simple Connection Management Service

In Figures 7–10, actions beginning with '–' or '+' indicate respectively sending or receiving the named message via the lower-level medium, which provides a (lossy) channel in each direction. The channels are not "broken:" a message sent repeatedly will eventually arrive correctly. Timeouts, assumed to be supplied by the medium, are indicated by "TM," "tm," and so forth. Any message that arrives in a state in which no reception is specified for it is simply removed from the channel and ignored. In the following, for convenience we shall refer to the peers themselves as "caller" and "listener."

A simple connection management protocol is shown in Fig. 7. This protocol (P) uses a *two-way handshake* in opening the connection, and another two-way handshake in closing it, or after a connection is refused by the listener. Protocol P is correct (i.e., implements the service of Fig. 6) if the channels can lose or duplicate messages but not reorder them. The opening two-way handshake consists of a C message from caller to listener, which is acknowledged by a message y in the other direction. The caller retransmits C until either y or n is received. When disconnection is requested or the listener refuses a connect attempt, the caller sends D repeatedly until receiving the acknowledgement d. Besides ensuring that the connection is not left "half open," this second handshake "flushes" any duplicate C messages from the channel; once the listener receives D, any subsequent C message indicates a new connection. Thus, the correctness of P depends upon the fact that messages cannot be reordered.

A second protocol (Q) providing the same service is shown in Fig. 8. It is correct if the channels can lose, duplicate, or reorder messages, provided a *bounded life-*

[1] The definitions given here are adequate under the assumption of non-divergence, i.e. no machine has two or more states reachable from each other via only internal transitions. This assumption is valid for all the machines in this paper. If it does not hold, the theory becomes more complex (cf. [4]).

[2] These same primitives are used in defining the connection management part of the ISO Transport Service [12].

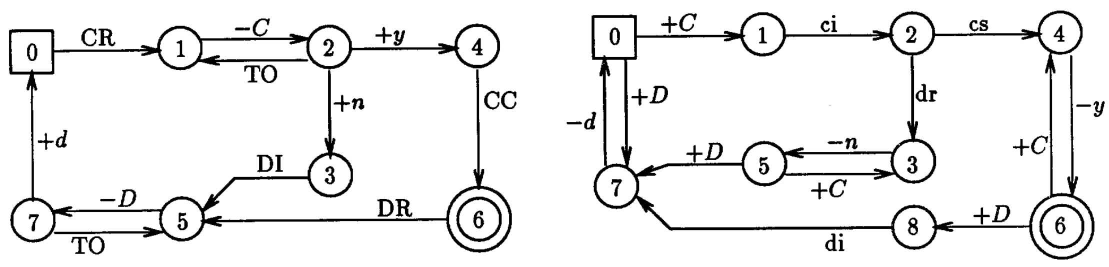

Figure 7: Protocol P for order-preserving channels

time for messages is enforced. That is, a certain time after any message is sent, the sender "knows" that it has been removed from the channel, either by the receiver or by the channel itself. Protocol Q uses the same 2-way handshakes as P does for opening and closing, plus a timeout on closing or refusal. After receiving d or n, the caller enters a wait state, where it remains for a time T1, which exceeds the sum of the maximum message lifetimes for each channel plus the maximum time it takes a listener to respond to a received message. Thus, when the caller re-enters the initial state the channels are again empty. The listener, after sending d or n, waits until time t2 has elapsed since it last received a D message; t2 is sufficient to allow any duplicate messages from the *current* connection to drain from the channel. Together, these delays ensure that any C message received at the listener indicates a new connection, and also that no "old" D message is in the channel when a new C arrives at the listener.

Although Q is correct for channels that can reorder messages, it constrains the users to wait between connections. This wait can be eliminated if each peer maintains a local ***incarnation number*** that changes with each connection. By tagging each message with the incarnation numbers of its sender and receiver, delayed messages can be detected and ignored. This technique is used in the connection management phase of well-known transport protocols such as TCP and TP-4 [11, 18]. For the sake of efficiency, the per-message overhead devoted to incarnation numbers is fixed, and the set of possible values for each incarnation number is finite. Correct functioning of a multi-incarnation protocol requires that no message tagged with a particular incarnation value be in the channel when that value is re-used. If the channels enforce maximum message lifetimes as described above, it is sufficient to enforce a lower bound on the time between uses of any particular incarnation value; this in turn can be ensured by placing an upper bound on the rate at which values are used. (Observe that protocol Q can be viewed as a special case in which each peer has only one possible incarnation value.)

In the closed state, neither peer "knows" the other's incarnation number. In particular, the initial C message is tagged with the caller's incarnation but not the listener's, and upon receiving a message C_x the listener cannot tell whether the message is an old one: x may or may not be the caller's current incarnation number. As a result, the initial exchange becomes a ***3-way handshake*** [19], in which the caller (incarnation u) sends C_u, the listener (incarnation v) responds with y_{uv}, and the caller confirms its current incarnation by sending a third message A_{uv}. Successful completion of a 3-way handshake thus establishes the connection.

Our third protocol, Z, uses multiple incarnation numbers and a 3-way handshake. Each peer has two state variables *loc-inc* and *rem-inc*, which keep track of the local and remote incarnation numbers respectively. The value of *loc-inc* changes only when the peer is in the closed state; *rem-inc* initially has the value "unknown," and is set during the initial handshake. Fig. 9 shows the caller and listener of Z in a parameterized form: each state is marked with a pair denoting the value of *loc-inc* and *rem-inc* at that state (a hyphen denotes the value "unknown" for the remote incarnation). The behaviour shown is for the single caller incarnation u and listener incarnation v. A state (or transition) in which v appears in the figure represents similar states (transitions) for all possible listener incarnations; similarly u is representative of all caller incarnations. For example, if the possible listener incarnation numbers are 0 and 1, the transition labeled $+y_{uv}$ in the caller represents the two actual transitions $+y_{u0}$ and $+y_{u1}$, leading to states $5/u$,0 and $5/u$,1 respectively. The symbols u' and v' represent respectively the values of the caller and listener incarnation numbers to be used next after u and v. The caller state marked $0/u'$,- is analogous to state $0/u$,-; thus the structure of states 0–15 is repeated beginning with state $0/u'$,-, with u' replacing u. Similarly, the listener structure repeats beginning with state $0/v'$,-. If each peer has two possible incarnation values, the full caller has 55 states and the full listener

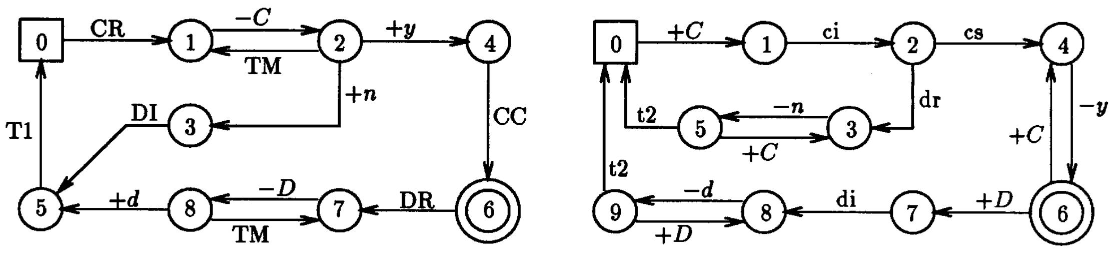
Figure 8: Protocol Q with time-wait between attempts

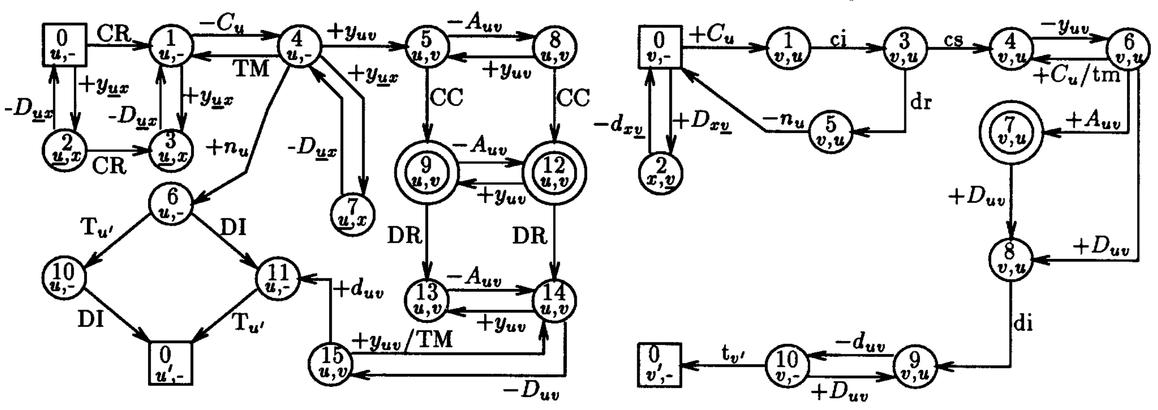
Figure 9: Protocol Z with incarnation numbers

has 42 states.

A normal connection begins as described above, with a 3-way handshake. Upon receiving the message C_u, the listener sets its *rem-inc* variable to u, and (if the user accepts the connection) sends y_{uv} until a response is received. Upon receiving y_{uv}, the caller sets *rem-inc* to v, and sends A_{uv}. When disconnection is requested, the caller sends D_{uv}; upon receiving it, the listener informs the user and then responds with d_{uv}. After this closing 2-way handshake, the caller "waits" until the timeout $T_{u'}$ expires. The purpose of this timeout is to ensure that any messages from the last use of incarnation u' have drained from the channel when the caller enters state $0/u'$,-. (This can be achieved by various local protocols [15, 19].) Note that this timeout event may not involve any actual waiting if the required time has already elapsed when it is encountered, or if the value u' has not been used yet. The listener timeout $t_{v'}$ is exactly analogous, ensuring that no messages with tag v' are in the channel when listener enters state $0/v'$,-.

If the user refuses a requested connection, the listener responds to C_u with n_u and returns to the closed state. There are two points to note about this case. First, the listener sends no message with v in the tag, and thus need not wait before returning to the closed state. Second, if a duplicate C_u message is subsequently received after the n_u has been sent, it will be treated as a new request, and the connect indication will again be given to the user. (Indeed, *any* C message received is treated as a new request by the listener.) In these circumstances protocol Z does not implement the service of Fig. 6, but a somewhat weaker one, in which "ci" can occur when it is not preceded by a "CR."[3] However, even if the user accepts such a spurious attempt, the connection will not persist. In both peers, the notation $\underline{x}$ denotes a number other than x; thus, the caller recognizes any message with tag $\underline{u}v$ as invalid. Suppose the listener receives C_w when the caller's incarnation is actually $u \neq w$. The listener responds with y_{wv}. This message matches the $+y_{\underline{u}x}$ transition from state 0 of the caller, so the caller responds with D_{wv}, and the listener terminates the connection. The transition $+D_{x\underline{v}}$ from state 0 in the listener ensures that the caller receives a d message in response to disconnection, even when the listener has already reached the closed state.

Suppose now that we wish to convert both P and Q to the 3-way handshake protocol Z. Adaptors for this purpose are shown in Fig. 10. The state machine for

[3]Such spurious connect indications are practically impossible to avoid, and are not precluded by the ISO Transport Service and Class 4 protocol definitions [11, 12].

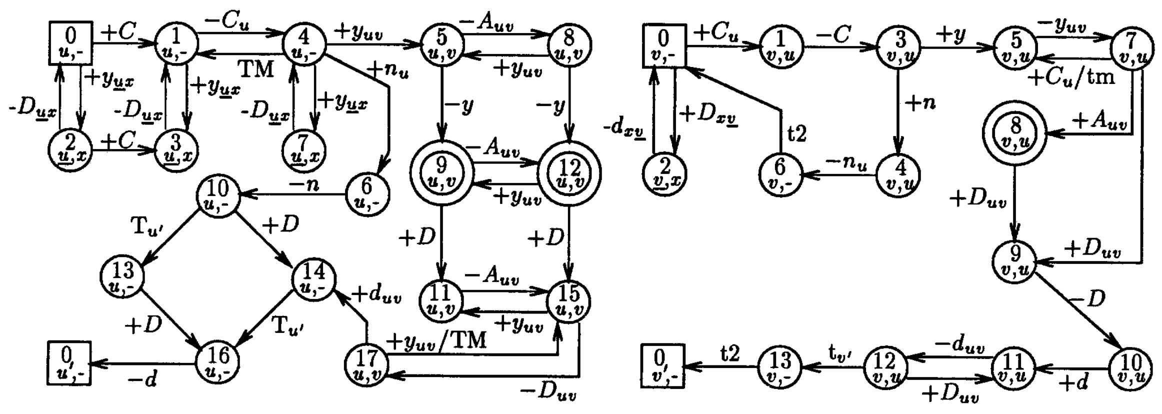
Figure 10: Adaptors D_0 (left) and D_1

each is similar to the corresponding Z peer; they are depicted in the parameterized form described above. The adaptor D_0 is for the P caller, while D_1 is for the Q listener. The adaptors were obtained with the algorithm of [4], with subsequent "trimming" of the machines by hand.[4] In the input to the quotient algorithm, caller and listener of Z were each defined to have two possible incarnation numbers 0 and 1.

Instead of service primitives at the upper interface, the adaptor sends and receives messages to/from the P or Q peer. For example, instead of "CR," adaptor D_0 has "$+C$;" instead of "di," the listener has $-D$, etc. Note that D_1 provides the t2 timeout event to the Q listener, but this event does not involve any actual waiting after $t_{v'}$ occurs—the condition enforced by $t_{v'}$ obviates the t2 requirement. It can be verified that (D_0 || P-caller) **sat** Z-caller and (D_1 || Q-listener) **sat** Z-listener. Adaptors for the P-listener and Q-caller also exist.

5 Conclusions

We propose the use of *adaptors* to enable cooperation between peers of different protocols. Adaptors have several advantages over other conversion architectures, especially gateway-type converters: they avoid bottlenecks at network boundaries, and a message is translated at most twice on its way from one peer to the other. Adaptors are well-suited for conversion among multiple protocols. In addition to these advantages, the definition of an adaptor as the "quotient" of known components is simpler than for other converters; it is therefore simpler to compute an adaptor algorithmically or to verify one derived heuristically.

[4]The algorithm given in [4] solves problems in which the quotient has only one interface, but is easily modified to handle problems in which the quotient has two interfaces, as for an adaptor.

Certain conditions must be satisfied for use of adaptors to be feasible, including: an adequate internetwork-spanning service at the lower level (as provided, for example, in the DARPA Internet by the Internet Protocol [17]); implementation of the target peer's upper interface primitives by the existing peer, and an explicit interface between the existing peer implementation and the lower-level service, allowing insertion of the adaptor. These requirements are somewhat stronger than those for more general converters; that is, nonexistence of an adaptor solution for a given problem does not necessarily imply that no gateway-converter solution exists. The conditions, however, do not seem unreasonably strong.

The adaptor approach to conversion is quite general, and can be applied at any level in an architecture, subject to the above conditions. It seems especially suitable for transport protocols, in conjunction with an internetwork service formed by concatenating individual network services.

References

[1] J. Auerbach. A protocol conversion toolkit. *IEEE Journal on Selected Areas of Communications*, SAC-8(1), January 1990, pp.143–159.

[2] G. v. Bochmann and P. Mondain-Monval. Design principles for communication gateways. *IEEE Journal on Selected Areas of Communications*, SAC-8(1), January 1990, pp.12–21.

[3] F. M. Burg and N. D. Iorio. Networking of networks: Interworking according to OSI. *IEEE Jour-*

nal on Selected Areas of Communications, SAC-7(7), September 1989, pp.1131–1142.

[4] K. L. Calvert and S. S. Lam. Deriving a protocol converter: a top-down method. In *Proceedings of SIGCOMM '89 Symposium, Austin, TX*, 1989.

[5] K. L. Calvert and S. S. Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas of Communications*, SAC-8(1), January 1990, pp.127–142.

[6] R. Cole. Experience and analysis of network interconnection. *IEEE Journal on Selected Areas of Communications*, SAC-8(1), January 1990, pp. 49–56.

[7] D. Einert and G. Glas. The SNATCH gateway: Translation of high-level protocols. *Journal of Telecommunications Networks*, 1983, pp.83–102.

[8] P. E. Green, Jr. Protocol conversion. *IEEE Transactions on Communications*, COM-34(3), March 1986.

[9] I. Groenbak. Conversion between the TCP and ISO transport protocols as a means of achieving interoperability between data communications systems. *IEEE Journal on Selected Areas in Communications*, SAC-4(2), March 1986, pp.288–296.

[10] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1986.

[11] International Organization for Standardization, International standard 8073. *Connection-oriented Transport Protocol Specification*, 1986.

[12] International Organization for Standardization, International Standard 8072. *Transport Service Definition*, 1986.

[13] S. S. Lam. Protocol conversion. *IEEE Transactions on Software Engineering*, SE-14(3), March 1988, pp.353–362.

[14] P. M. Merlin and G. von Bochmann. On the construction of submodule specifications and communications protocols. *ACM Transactions on Programming Languages and Systems*, 5(1), Jan 1983.

[15] S. Murphy and A. U. Shankar. Connection management for the transport layer: service specification and protocol verification. Technical Report, Univ. of Maryland Computer Science Dept., 1989. Preliminary version in *Proceedings of ACM SIGCOMM 88 Symposium*, Stanford, CA, August 1988.

[16] M. A. Padlipsky. Gateways, architectures, and heffalumps. DDN Network Information Center, DARPA Internet Request for Comments No. 875, September 1983.

[17] J. Postel. *Internet Protocol.* DDN Network Information Center, DARPA Internet Request for Comments No. 791, September 1981.

[18] J. Postel. *Transmission Control Protocol.* DDN Network Information Center, DARPA Internet Request for Comments No. 793, September 1981.

[19] C. A. Sunshine and Y. K. Dalal. Connection management in transport protocols. *Computer Networks*, 2, 1978.

[20] K. Sy, O. Shiobara, M. Yamaguchi, Y. Kobayashi, S. Shukuya, and T. Tomatsu. OSI-SNA interconnections. *IBM Systems Journal*, 26(2), 1987.