Possibilities of Using Protocol Converters for NIR System Construction

Sven Ubik
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Prague, Czech Republic
E-mail: ubik@fsid.cvut.cz

November 1996

Abstract

Volumes of information available from network information services have been increasing considerably in recent years. Users' satisfaction with an information service depends very much on the quality of the network information retrieval (NIR) system used to retrieve information. The construction of such a system involves two major development areas: user interface design and the implementation of NIR protocols.

In this paper we describe and discuss the possibilities of using formal methods of protocol converter design to construct the part of an NIR system client that deals with network communication. If this approach is practicable it can make implementation of NIR protocols more reliable and amenable to automation than traditional designs using general purpose programming languages. This will enable easy implementation of new NIR protocols custom-tailored to specialized NIR services, while the user interface remains the same for all these services.

Based on a simple example of implementing the Gopher protocol client we conclude that the known formal methods of protocol converter design are generally not directly applicable for our approach. However, they could be used under certain circumstances when supplemented with other techniques which we propose in the discussion.

Keywords: network information services, network information retrieval, protocol conversion

1 Introduction

Important terms used in this paper are illustrated in Fig. 1. An *information provider* makes some information available to a user through a network by means of a *network information retrieval (NIR) system*, thereby providing an *information service*. An NIR system consists of the provider's part (the *server*) and the user's part (the *client*) communicating via an *(NIR) protocol*.

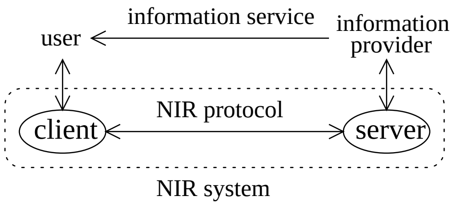

Figure 1: Network information retrieval (NIR) system.

We use the term *information retrieval* to denote passing information, (a document or other form of information), from a repository to a user or from a user to a repository in general. We concentrate, however, on passing documents from a repository to a user. Some authors use the term information retrieval to denote a form of getting information from a repository where searching or some more intelligent query processing is involved [9, 15] or distinguish between document retrieval and information retrieval (retrieval of documents and information in other forms) [14].

Usually, a user wants to use a lot of information services available in order to get as much information relevant to his or her needs as possible. At the same time, a user wants to use all information services in a similar way. Good consistency of user interfaces for accessing various information services and for presenting documents of various types will decrease the *cognition overhead* — the additional mental effort that a user has to make concerning manipulation with a user interface, navigation, and orientation [25]. Consistency can be improved by applying standards for "look and feel" of user interface elements, by using gateways between information services, and, most importantly, by using integrated NIR tools.

An *integrated NIR tool* is a multiprotocol client designed to communicate directly with servers of various information services. This approach has become very popular recently, since it provides a more consistent environment than a set of individual clients and does not have the disadvantages of gateways in the matter of network communication overhead and limited availability.

Nevertheless, current integrated NIR tools often suffer from important drawbacks:

- They support a limited number of NIR protocols and adding a new protocol is not straightforward.
- Implementation of NIR protocols is not performed systematically, which means that it is not possible to make any formal verification of such a system.
- Their portability is limited, and porting to another type of a window system is laborious.

Constructing such tools involves two major development areas: user interface design and implementation of NIR protocols. In the next two chapters we will briefly describe some of the techniques and formalisms used in these development areas. Then we will mention the role of protocol conversion in network communication and shortly describe three formal methods of protocol converter design. In the rest of the paper, we will demonstrate the application of the protocol conversion techniques to a simple protocol, the Gopher protocol. Despite the acceptance of these techniques, our analysis demonstrates the negative conclusion that these techniques alone are inadequate for a protocol even as simple as the Gopher protocol. We conclude with some suggestions for enhancements.

2 User Interface Design and Layered Models

When designing a user interface various models and formal description techniques are used. An important class of models is formed by layered models in which communication between a user and an application is divided into several layers. Messages exchanged at higher layers are conveyed by means of messages exchanged at lower layers. An example of this kind of model is the seven-layer model described by Nielsen in [16] shown in Fig. 2 along with an indication of what kind of information is exchanged at each layer.

Various formalisms can be used to describe the behaviour of the system at individual layers. Traditionally, most attention has been paid to the syntactic layer. Among the formal description techniques used at this layer the most popular are context-free grammars, finite-state automata, and event-response languages. Taylor [23, 24] proposed a layered model where different layers are conceived as different levels of abstraction in com-

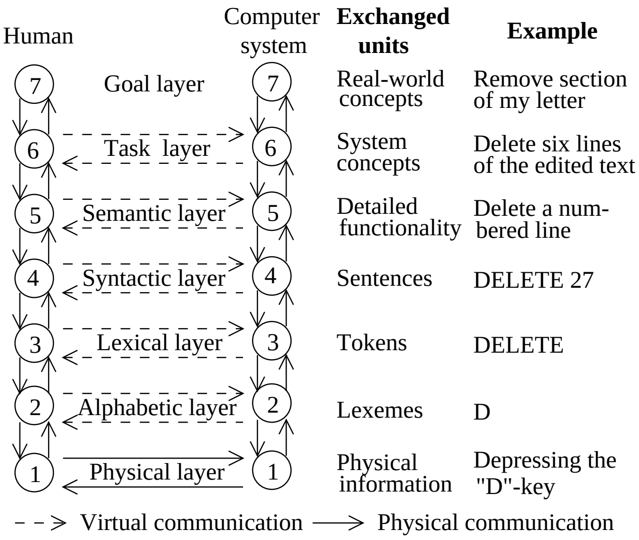

Figure 2: Seven-layer model for human-computer interaction.

munication between the user and the application. Lexical, syntactic, and semantic characteristics are spread within all levels. It appears that the layered structure is natural to human-computer interaction and implementations of user interfaces should take this observation into consideration.

3 Network architectures and layered models

OSI model The ISO Open System Interconnection (OSI) model [6] divides a network architecture into seven layers (Fig. 3). Each layer provides one or more *services* to the layer above it. Within a layer, services are implemented by means of *protocols*.

Many current network architectures are based on OSI model layering. Although not all layers are always implemented and functions of multiple layers can be grouped into one layer, dividing communication functionality into layers remains the basic principle.

Protocol specification Formal techniques for protocol specification can be classified into three main categories: transition models, programming languages, and combinations of the first two.

Transition models are motivated by the observation that protocols consist largely of processing commands from the user (from the higher level), message arrivals (from the lower layer), and internal timeouts.

In this paper, we will use finite state machines (graphically depicted as state-transition diagrams) which seem

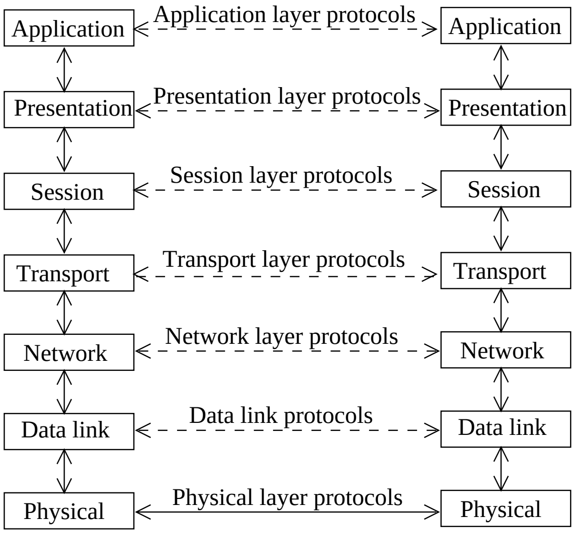

Figure 3: OSI model.

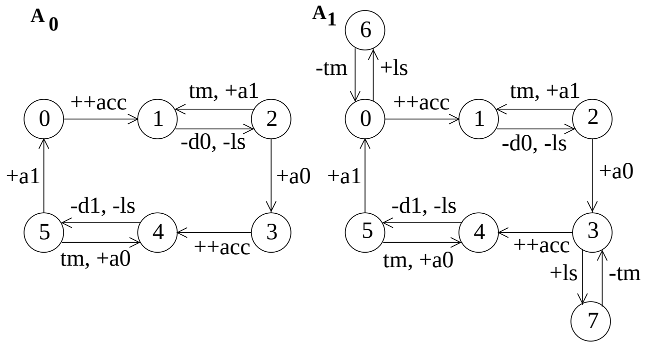

Figure 4: The alternating-bit (AB) protocol.

to be the most convenient specification technique for our application domain.

An example of the state-transition diagram depicting the alternating-bit (AB) protocol [1] is shown in Fig. 4. Data and positive acknowledgment messages exchanged between two entities are stamped with modulo 2 sequence numbers. $d0$ and $d1$ denote data messages, $a0$ and $a1$ denote acknowledgment messages, a plus sign denotes receiving a message from the other party, a minus sign denotes sending a message to the other party, a double plus sign (++) denotes getting a message from the user (acceptance), a double minus sign ($--$) denotes putting a message to the user (delivery), *ls* and *tm* denote loss of a data message and timeout, respectively.

4 Protocol Converters

In network communication we can encounter the following problem. Consider that we have two entities P_0 and

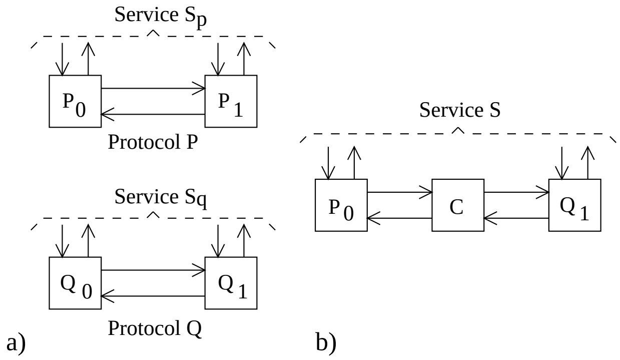

Figure 5: a) Communicating entities, b) Using a protocol converter to allow different entities to communicate.

P_1 communicating by means of a protocol P (thus providing a service S_p) and other two entities Q_0 and Q_1 communicating by means of a protocol Q (thus providing a service S_q), see Fig. 5 a. Then we might want to make P_0 communicate with Q_1, thus providing a service S similar to services S_p and S_q. When protocols P and Q are compatible enough, this can be achieved by a protocol converter C, which translates messages sent by P_0 into messages of the protocol Q, forwards them to Q_1, and performs a similar translation in the other direction, see Fig. 5 b.

To solve the problem of constructing a protocol converter C, given specifications of P_0, Q_1, and S, several more or less formal methods have been developed. Most of them accept specifications of protocol entities in the form of communicating finite-state machines. We give a brief description of the principles used by three important methods.

Conversion via projection An image protocol may be derived from a given protocol by partitioning the state set of each communicating entity; states in the same block of the partition are considered to be indistinguishable in the image of that entity. Suppose protocols P and Q can be projected onto the same image protocol, say R. Then R embodies some of the functionality that is common to both P and Q. The specification of a converter can be derived considering that the projection mapping defines an equivalence between messages of P and Q, just as it does for states. Finding a common image protocol with useful properties requires a heuristic search using intuitive understanding of the protocols. For more details, refer to [13, 5].

The conversion seed approach This approach was first presented in [17]. From the service specification S, a conversion seed X is (heuristically) constructed. The seed X is a finite-state machine whose message set is a subset of the union of the message sets of P_1 and Q_0; it is

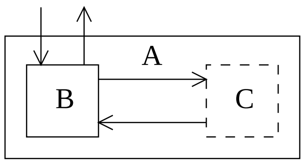

Figure 6: The quotient problem.

a partial specification of the converter's behaviour in the form of constraints on the order in which messages may be sent and received by the converter. Then a three-step algorithm [5] is run on P_1, Q_0, and X. If a converter C is produced (the algorithm generates a non-empty output), the system comprising P_0, C, and Q_1 should be analyzed. If this system satisfies the specification S, then C is the desired converter. Otherwise, a different iteration with a different seed could be performed. Unfortunately, if the algorithm fails to produce a converter, it is hard to decide whether the problem was in the seed X used or if there was a hard mismatch between the P and Q protocols.

The quotient approach Consider the problem depicted in Fig. 6. Let A be a service specification and let B specify one component of its implementation. The goal is to specify another component C, which interacts with B via its internal interface, so that the behaviour observed at B's external interface implements the service defined by A. This is called the quotient problem. It is clear that Fig. 5 b depicts a form of the quotient problem: P_0 and Q_1 correspond to B, S corresponds to A, and the converter to be found is C. An algorithmic solution of the quotient problem for a class of input specifications was presented in [4, 5]. A similar problem has been discussed in [12].

5 NIR System Design Proposal

5.1 Information Retrieval Cycle

A user wants to receive required information easily, quickly, and in satisfactory quality. Although there are many differences in details, the process of obtaining information when using most NIR services can be outlined by the *information retrieval cycle* shown in Fig. 7.

At each step there is an indication whose turn it is, either user's (U) or system's (S).

First, the user has an *idea about required information*, such as "I would like to get some article on X written by Y". Next, the user has to choose an *information service* and a *source of information* (a particular site which offers the chosen information service) to exploit.

Then, the user has to *formulate a query* specifying the required information in a form that the chosen information service understands and that is sufficient for it to find the information.

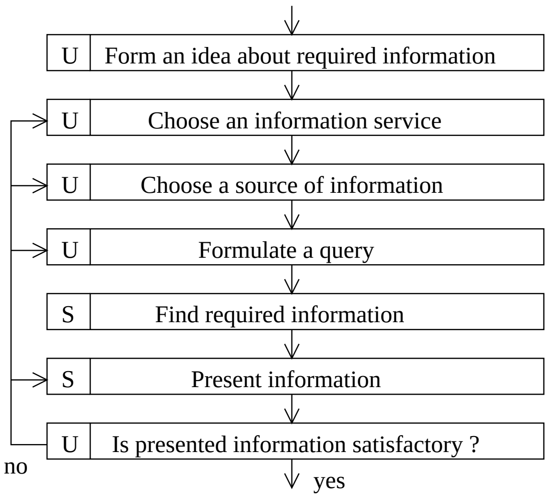

Figure 7: Information retrieval cycle.

Now, it is the computer system's task to *find* the information and *present* it to the user. There are four possible results:

- information was found and presented in satisfactory quality
- information was found and presented, but the user is not satisfied with it
- no information was found using the given description
- the given description was not in a form understandable by the given information service

When some information was found and presented to the user, but it is not to the user's satisfaction and he or she believes that better information could be obtained, the next iteration in the information retrieval cycle can be exercised. According to the measure of the user's dissatisfaction with the presented information, there are several possible points to return to. In some cases, different presentation of the same information would be satisfactory. In other cases, the user has to reformulate the query or choose another source of information or even another information service.

Looking at Fig. 7, we can see that the user's role is much easier when a NIR system logically integrates access to various information services. This means that differences between individual information services should be diminished in all steps as much as possible.

Figure 8: Client as a protocol converter.

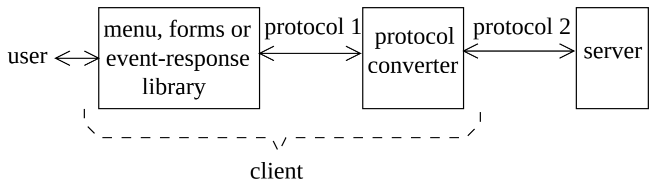

Figure 9: Client as a library and a protocol converter.

5.2 Possibilities of Employing Protocol Converters

A client part of a NIR system may be seen as communicating via two sets of protocols. It uses one set of protocols to communicate with the user (see Fig. 2) and another set of protocols to communicate with the server (see Fig. 3). This may suggest an idea to us: can the client be constructed as a protocol converter (or a set of protocol converters in the case of a multiprotocol client) using some of the formal methods of protocol converter design? This idea is illustrated in Fig. 8.

The first question that arises is: what are the protocols which the converter should transform? In a typical environment, all lower layer protocols up to the transport layer are the same for all NIR services supported by the client. It is the upper layer NIR protocols that differ, usually based on the exchange of textual messages over a transport network connection (e.g., FTP, Gopher protocol, SMTP, NNTP, HTTP). These protocols seem to be good candidates for the protocol on a server's side of the converter.

In the case of a command language user interface, communication with the user can also be regarded as a protocol based on the exchange of textual messages. This could be a protocol on the user's side of the converter.

If the user interface uses interaction techniques like menu hierarchy, form filling or currently the most popular direct manipulation, it may be implemented as a library that offers an interface in the form of a protocol similar to that of a command language. Again, a protocol converter could be employed as shown in Fig. 9.

While the protocols on the server's side of the converter are given by the information services we want to support, the protocol on the other side is up to us to specify (if it is not directly the user — client protocol as in the configuration shown in Fig. 8). An important decision is the choice of the proper level of communication. Low level communication consisting of requests to display user interface elements and responses about user interaction can ease the development of the user interface but it would make the protocol converter construction more difficult because of a great semantic gap between the two protocols to be converted. We will try to find a level of communication which makes it possible to use one of the formal methods of protocol converter design that we mentioned before.

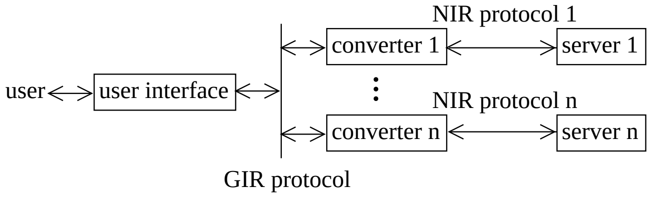

Figure 10: NIR system based on the GIR protocol.

5.3 General Information Retrieval Protocol

Many protocols for client-server communication in current NIR services are similar to some extent. There are common functions that can be identified in most of them such as a request for an information object, sending the requested information object, sending an error message indicating that the requested information object cannot be retrieved, a request to modify an existing or to create a new information object, a request to search the contents of an information object, etc. It seems feasible that a high-level *general information retrieval (GIR) protocol* providing a high-level *(GIR) service* can be designed. Such a protocol has to support all major functions of individual NIR services. It would work with a *global abstract information space* formed by the union of information spaces of individual NIR services. This protocol operating on information objects from the global abstract information space would be converted by a set of protocol converters to particular NIR protocols operating on information objects from information spaces of concrete NIR services. A structure of an NIR system built around the GIR protocol is shown in Fig. 10.

Considering the structure of protocols used in current NIR services and respecting that the behaviour of the entity on the left side of the GIR protocol, the user interface, corresponds to the information retrieval cycle depicted in Fig. 7, we can propose a very simple GIR protocol depicted by the state transition diagrams U_0 (the client) and U_1 (the server) shown in Fig. 11. This version is certainly far from being complete and needs to be improved on the basis of later experience, see Chapter 7.

The notation used corresponds to that described in Chapter 3. A dashed transition leading to state 1 matches the first step in the information retrieval cycle (Fig. 7). It represents a solely mental process with no interaction over the network and will not be considered in further discussion. A letter u in front of a message name means that it is a GIR protocol message (u stands for universal). Later we will use a letter g for Gopher protocol messages in order to distinguish them.

The user chooses both an information service and a source of information in one step. It may be divided into two steps but one step better corresponds to picking up a bookmark or entering a document URL. The choice of information service would select a matched protocol converter. Sending all information to the server is delayed in the client until the user enters it completely. This allows backtracking in user input without having to inform the server about it.

5.4 General Window System Interface

We may consider another possible usage of protocol converters in an NIR system. There is often a need to port such a system to several platforms — window systems. Some developers try to make their applications more portable by performing the following steps:

1. Identify common functions of all considered window systems.

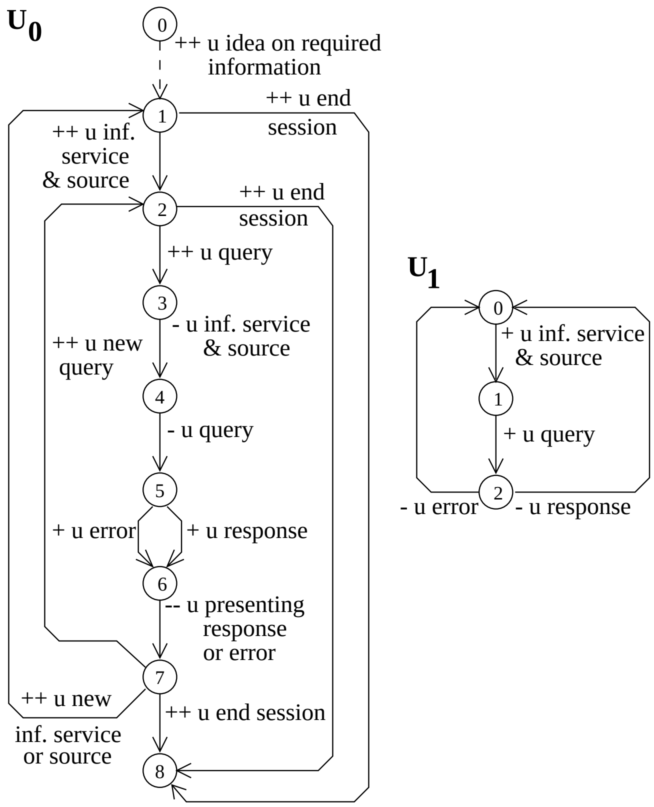

Figure 11: General information retrieval protocol.

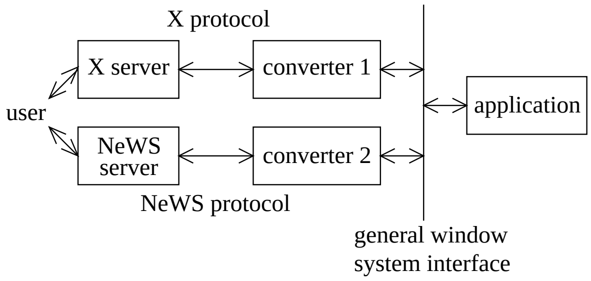

Figure 12: Using protocol converters to construct the general window system interface.

2. Define interface to a general window system that implements functions identified in step 1.

3. Implement the general window system on the top of all considered window systems.

4. Implement the application using the general window system.

Some window systems are based on the client-server model (e.g., the X Window System or NeWS). We may try to realize step 3 above with a set of protocol converters converting client-server protocols of considered window systems to the protocol used by the application to communicate with the general window system defined in step 2. This idea is depicted in Fig. 12.

Unfortunately, this idea is hardly feasible. The client-server protocols of today's window systems are usually complex and differ significantly from each other (and from a possible protocol of the general window system). Currently known methods of protocol converter design are suitable for protocols that are sufficiently compatible. Therefore, the general window system can be more easily implemented as a set of shim libraries built on the top of existing window systems (see Fig. 13). Examples of systems that use this approach are stdwin [20] and SUIT [18].

5.5 Structure of the Proposed System

The possible structure of a multi-service NIR system that uses shim libraries to adapt to various window systems and a set of protocol converters between the GIR protocol and individual NIR protocols is depicted in Fig. 14.

The gap between the general window system interface and the GIR protocol is bridged by a module which implements the user interface. It can be designed either as a protocol converter, or in the case of difficulties when applying the formal methods described, as an event-response module written in a general programming language.

Integration of information services that are not based on a distributed client-server architecture (with a local

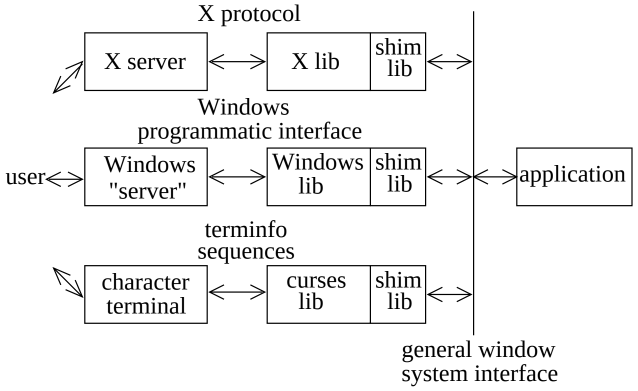

Figure 13: Using shim libraries to construct the general window system interface.

client and a remote server) with a well-defined NIR protocol, that have their own remote user interfaces accessible by a remote login, such as library catalogs and databases, can be achieved by using a new front end to the old user interface. This can be a protocol converter that converts the GIR protocol to sequences of characters that would be typed as input by a user when communicating directly with the old user interface, and that performs a similar conversion in the other direction. Quotation marks around the "server" for this type of information service in Fig. 14 indicate that such an information service may or may not be based on the client-server model. Although this front end seems to be subtle and vulnerable to changes of the old user interface, this approach has already been successfully used (but the front end was not implemented as a protocol converter). Examples are SALBIN [8] and IDLE [19].

5.6 OSI Model Correspondence

Fig. 15 illustrates how protocol layering in the proposed system corresponds to the OSI model. Internet information services are based on the TCP/IP protocol family, which provides no explicit support for functions of the session and presentation OSI layers. NIR protocols (FTP, Gopher protocol, etc.) are, with respect to the OSI model, application layer protocols.

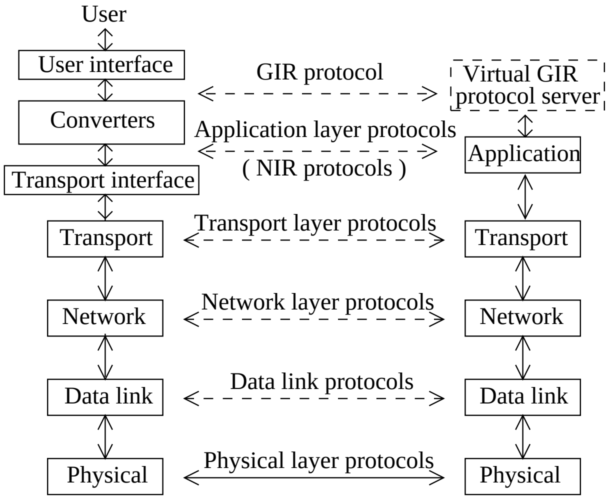

Figure 15: OSI model correspondence.

Protocol converters in our system use application layer protocols (NIR protocols) on one side and the GIR protocol on the other side. They act as a client side implementation of the application layer and from the client's point of view they create another higher layer based on the GIR protocol. The server side of this higher layer is not actually implemented, it exists as a client side abstraction only. Being application layer implementation, the converters communicate with the transport layer, possibly over a transport interface module which converts an actual transport layer interface to a form suitable for the converter's input and output. On the upper side of the converters, there is the user interface which presents information services to the user.

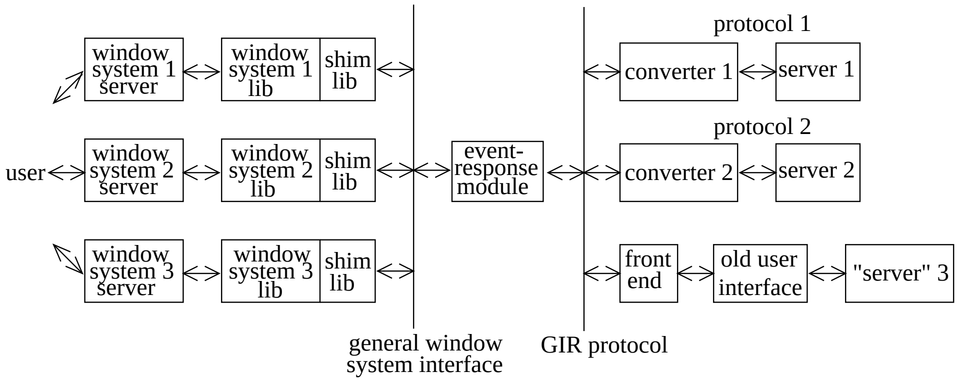

Figure 14: A possible structure of a multi-service information retrieval system.

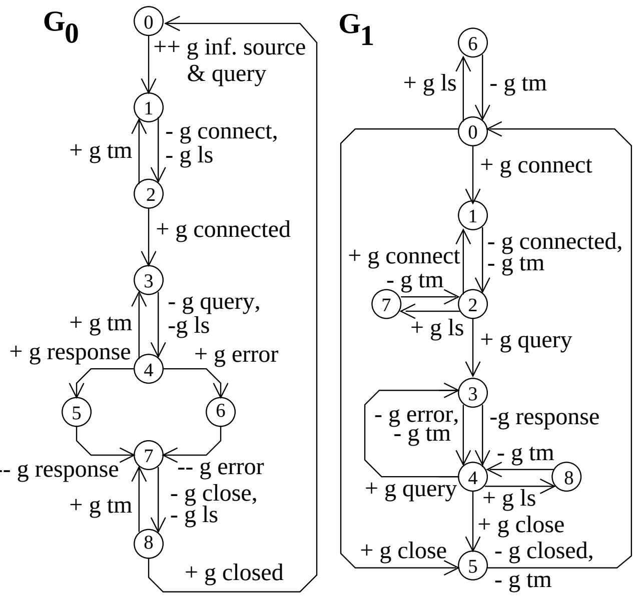

Figure 16: Gopher protocol client (left) and server (right).

6 Example

As an example of applying the approach described in Chapter 5, we will try to construct a protocol converter implementing the network communication part of the Gopher protocol client. We will try to use all three protocol converter design methods mentioned in Chapter 4 in order to decide which one is the most suitable and whether our approach is feasible at all.

The Gopher protocol [2, 3] can be described by two finite-state machines, one for the client side (G_0), and the other for the server side (G_1), communicating via message passing. Corresponding state-transition diagrams are shown in Fig. 16.

The notation used corresponds to that described in Chapter 3. Opening a connection and closing a connection between the client and the server are modeled by the exchange of virtual messages *g connect*, *g connected*, *g close*, and *g closed*. The loss of a message sent by the client is represented by sending a virtual *g ls* message instead of a "normal" message to the server which then sends back a virtual *g tm* (timeout) message as a response. The loss of a message sent by the server is represented by sending a virtual *g tm* message only. If more than one message may be sent in a certain state, one is chosen non-deterministically (of course, with the exception of *g response* and *g error* messages, one of them is sent by the Gopher protocol server based on the result of a query processed).

Our task is to construct a protocol converter which allows U_0 (the GIR protocol client) and G_1 (the Gopher protocol server) to communicate.

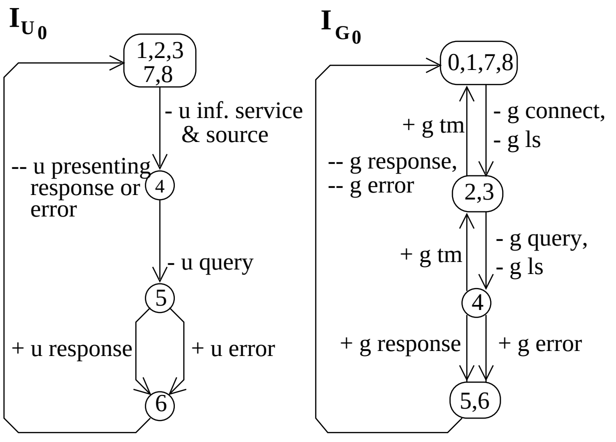

Figure 17: Images of client finite-state machines.

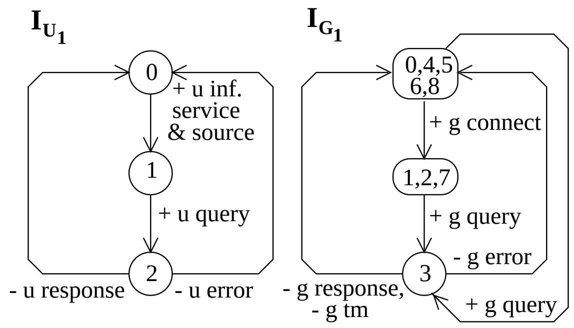

Figure 18: Images of server finite-state machines.

6.1 Conversion via Projection

When we try to project the finite-state machines U_0 and G_0 onto the common image, some of the best results we can get in terms of achieved similarity and retained functionality are the I_{U_0} and I_{G_0} images shown in Fig. 17.

They are close to each other, but they are not exactly the same. To obtain one common image, the functionality would have to be further reduced, which is not acceptable. On the server side, there is a similar problem. We can get images I_{U_1} and I_{G_1} of U_1 and G_1 shown in Fig. 18. These are not exactly the same and, moreover, the common functionality is not sufficient. We would not be able to translate the *g connected* and *g closed* messages sent by the Gopher protocol server which the converter needs to receive.

The conclusion is that conversion via projection is only suitable for protocols that are close enough, which is not our case.

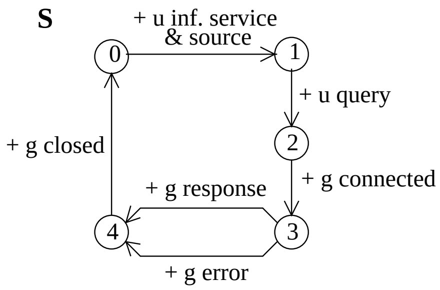

Figure 19: Conversion seed S for U_0 — G_1 converter.

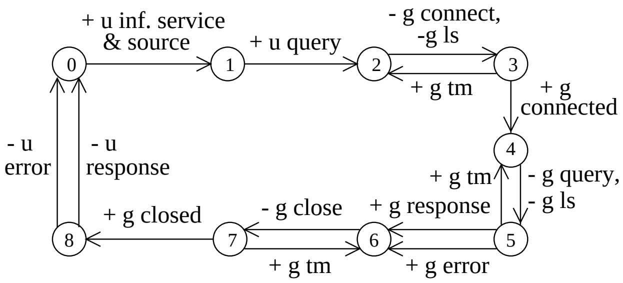

Figure 20: U_0 — G_1 converter produced from the conversion seed S.

6.2 Conversion Seed Approach

For the conversion seed approach, G_0 has to be modified so that it contains only transitions that correspond to interaction with the peer entity G_1 (interaction with the user is not included). That is, the transitions from state 4 lead directly to state 7 and the transition from state 8 leads to state 1, which is the starting state.

A simple conversion seed S is shown in Fig. 19. It defines constraints on the order in which messages may be received by the converter. Ordering relations between messages being sent and messages being received will be implemented in the converter by the algorithm which constructs the converter as a reduced finite-state machine of U_1 when communicating with U_0 and a reduced finite-state machine of G_0 when communicating with G_1 [5]. The output of the algorithm for U_1, G_0, and S is shown in Fig. 20. In state 8, the converter has to decide whether to send the *g response* or the *g error* message to U_0 based on the receiving transition that was used to move from state 5 to state 6. This requires some internal memory and associated decision mechanism in the converter.

We can conclude that the conversion seed approach is applicable to our example, but we have to construct a conversion seed heuristically using our knowledge of the converter operation.

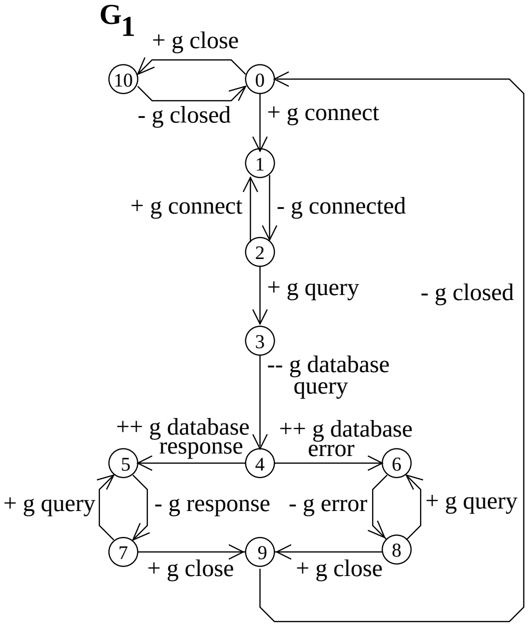

Figure 21: Gopher protocol server for the quotient approach.

6.3 The quotient approach

The algorithm based on the solution of the quotient problem described in [4, 5] uses a rendezvous model (as opposed to the message-passing model used in the previous two approaches), in which interaction between two components occurs synchronously via actions. An action can take place when both parties are ready for it. State changes happen simultaneously in both components.

Transmission channels between the converter and other communicating entities are modeled explicitly as finite-state machines with internal transitions, which may or may not happen, representing loss of messages. After such a loss, a timeout event occurs at the sender end of the channel.

Because of different modeling of message losses in the quotient approach, the state-transition diagram for the Gopher protocol server has to be slightly modified, as shown in Fig. 21. Virtual *g ls* and *g tm* messages are removed and new receive transitions are added to cope with duplicate messages sent by the converter.

In our case, the converter is collocated with U_0, meaning there are no losses in U_0 — C communication. We only have to model the C — G_1 channel (shown in Fig. 23), thereby obtaining the configuration shown in Fig. 22. The composition of U_0, CG_1chan, and G_1 forms the B part in the quotient problem (Fig. 6). The service specification A is shown in Fig. 24.

After applying the algorithm on these inputs we get a converter which has 194 states and 590 transitions, too many to be presented here. Some of the states and their

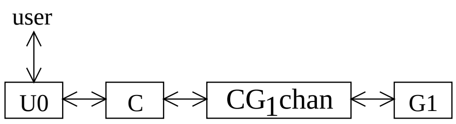

Figure 22: Quotient approach configuration.

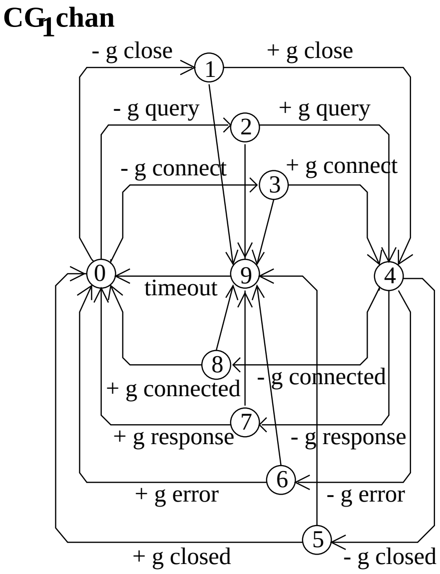

Figure 23: Converter — G_1 channel.

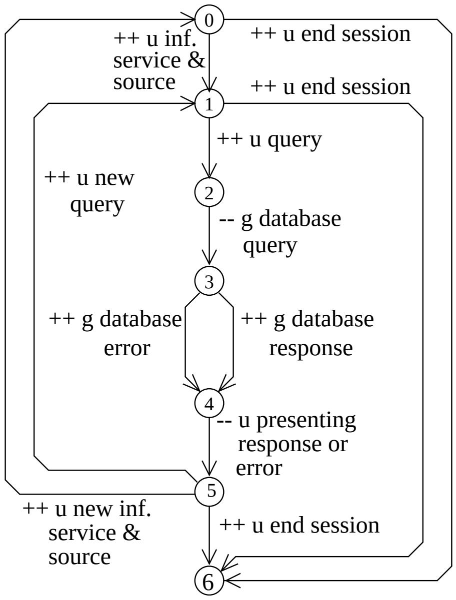

Figure 24: Service specification.

associated transitions represent alternative sequences in which messages may be sent by the converter. For example, *g close* message (request for the Gopher protocol server to close the connection) may be sent by the converter before sending *u response* message (response for the GIR protocol client) or after it. These alternatives are redundant with respect to the function of the converter. Unfortunately, it seems to be difficult to remove them in an automatic manner.

A more important problem is that some other states and transitions represent sequences which are not acceptable because the converter has not enough knowledge to form the content of a message to be sent. For example, *g connect* message (request for the Gopher protocol server to connect to it) can be sent only when the converter knows where to try to connect, that is after *u inf. service and source* message has been received, not before it. Transitions leading to such unacceptable sequences have to be eliminated by defining semantic relations between messages and enforcing them in run-time. In our example, two semantic relations are sufficient:

+ u inf. service and source $\Rightarrow$ *- g connect*
+ u query $\Rightarrow$ *- g query*

The symbol $\Rightarrow$ means that a message on the right side can be sent or received after a message on the left side has been sent or received, as indicated by the plus and minus signs.

Another problem with the quotient seeking algorithm is its computing complexity. Let S_X be the set of states of an entity X and let $|S_X|$ be the number of states in this set. Then the state set of the quotient can grow exponentially with the upper bound of $2^{|S_A|*|S_B|}$ states. In our configuration, the maximum number of states of B is equal to the product of the numbers of states of U_0, CG_1chan, and G_1. Thus:

$$2^{|S_A|*|S_B|}=2^{|S_A|*|S_{U0}|*|S_{CG_1chan}|*|S_{G1}|}=2^{7*8*10*11}=2^{6160}.$$

In practice, however, the algorithm does not always use exponential time and space. In our example, the maximum number of states of C during computation was 330.

We can conclude that the quotient approach, when supplemented with the definition of semantic relations between messages, is applicable to our example. It is more compute-intensive than the conversion seed approach but it is more systematic, and no heuristic constructions are required.

7 Discussion

In our example, we have shown that even the quotient approach, which is the most systematic of the currently known formal methods of protocol converter design, is not sufficient for our approach on its own. It has to be complemented with the definition of semantic relations between messages.

In addition to this, there are several problems with using protocol converters in the proposed way which we have not yet consider: GIR protocol universality, message contents transformation, and covering details of network protocols.

GIR protocol universality The GIR protocol suggested in Fig. 11 is too simple. It contains only two methods: set source and get object. A GIR protocol for use in real designs has to incorporate at least the following methods: get object metainformation, modify object, create new object, remove object, and search object.

Message content transformation When a converter should send message X in response to receiving message Y, it may have to build the content of message X based on the content of message Y. We call this process the message content transformation. In our example we need to provide for the transformation of the content of the following messages:

u inf. service and source	$\Rightarrow$	*g connect*
u query	$\Rightarrow$	*g query*
g response	$\Rightarrow$	*u response*
g error	$\Rightarrow$	*u error*

There are several ways to formalize message content transformation. We briefly outline four possible approaches.

translation grammars We can conceive the set of all possible contents of messages on each side of the converter as a language, and individual message contents as sentences of this language. We can then formally describe the languages with two grammars (one for each side of the converter) and the transformation of sentences with two translation grammars (one for each direction).

sequential rewriting system We can define a set of rewriting rules that specify how to get the content of an outgoing message beginning with the content of the corresponding incoming message. These rules would be applied sequentially on the message content using string matching in a similar manner as rules for mail header and envelope processing work in the sendmail program [21].

SGML link process The set of all possible contents of messages can be modeled using two SGML-based markup languages, one for each side of the converter. The transformation between them can then be performed as a pair of SGML link processes [10] (one for each direction) or as a pair of SGML tree transformation processes (STTP) defined within DSSSL [7] specifications for both languages.

predicate-based rewriting system We can define a set of facts and rules in Prolog or in a similar logical language that specify relations between pieces of information in an incoming message and the corresponding outgoing message. We then begin with a framework of the outgoing message in the form of a term composed of unbound variables. When we apply the defined set of predicates using a rewriting system to this term, its variables become step by step bound to the values from the incoming message content.

Further research will be required to find whether and under what conditions the proposed techniques could be used for message content transformation in our approach.

Covering details of network protocols Another problem concerns covering details of network communication protocols such as port numbers, parallel connections, and various options and parameters. It seems to be practicable to incorporate them into the protocol on the bottom side of the protocol converter (see Fig. 15) as variables in the content of exchanged messages or even as additional virtual messages recognized by the transport interface used.

8 Conclusion

We have shown that formal methods of protocol converter design could under certain circumstances be used to construct the part of an NIR system client that deals with network communication. However, these methods are not sufficient on their own and have to be supplemented with other techniques for our approach to become practicable. In the discussion we have mentioned the most important problems and proposed possible solutions or directions for further research work.

If these problems are resolved it will be possible to design many new specialized NIR services with their own custom-tailored protocols since implementation of these protocols can be done easily and reliably. New protocols could also be supported in a different way. Some new NIR services (e.g., HotJava [11]) intend to support new protocols by retrieving the code to implement them by a client from a server. Instead of retrieving the code, only the formal specification could be retrieved and the protocol would be implemented by the converter from the specification.

References

[1] K. A. Bartlet, R. A. Scantlebury, P.T. Wilkinson. "A note on reliable full-duplex transmission over half-duplex lines," *Communications of the ACM*, Vol. 12, No. 5, May 1969.

[2] B. Alberti, F. Anklesaria, P. Lindner, M. McCahill, D. Torrey. *The Internet Gopher protocol*, University of Minnesota, Microcomputer and Workstation Networks Center, 1992.

[3] B. Alberti, F. Anklesaria, P. Lindner, M. McCahill, D. Torrey, D. Johnson. *Gopher++ — Upward compatible enhancement to the Internet Gopher protocol*, University of Minnesota, Microcomputer and Workstation Networks Center, July 1993.

[4] Kenneth L. Calvert, Simon S. Lam. "Deriving a Protocol Converter: a Top-Down Method," *Proceedings of ACM SIGCOMM'89*, Austin, TX, 1989.

[5] Kenneth L. Calvert, Simon S. Lam. "Formal methods for protocol conversion," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 1, January 1990, pp. 127-142.

[6] J. D. Day, H. Zimmermann. "The OSI reference model," in *Proc. of the IEEE*, Vol. 71, December 1983, pp. 1334-1340.

[7] *Information technology — Text and office systems — Document Style Semantics and Specification Language (DSSSL)*, Draft International Standard ISO/IEC DIS 10179.2, 1984.

[8] R. Gartner. "Some network tools for Internet gateway service," *Journal of Information Networking*, Vol. 1, No. 2, 1994, pp. 168-173.

[9] Susan Gauch. "Intelligent information retrieval: An introduction," *Journal of the American Society for Information Science*, 1992, Vol. 43, No. 2, pp. 175-182.

[10] Charles F. Goldfarb. *The SGML Handbook*, Oxford University Press, 1990.

[11] *The HotJava Browser: A White Paper*, Sun Microsystems. 1995.

[12] Tilman Kolks, Gjalt de Jong, Bill Lin. "Modeling and optimization of hierarchical synchronous circuits," *Proceedings of ED&TC*, Paris, 1995.

[13] Simon S. Lam. "Protocol conversion," *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, March 1988, pp. 353-362.

[14] David D. Lewis, Karen Sparck Jones. "Natural Language Processing for Information Retrieval," *Communications of the ACM*, January 1996, Vol. 39, No. 1, pp. 92-101.

[15] X. Lin. *Self-Organizing Semantic Maps for Information Retrieval*, Doctoral dissertation, College of Library and Information Services, University of Maryland.

[16] Jakob Nielsen. "A virtual protocol model for computer-human interaction," *Int. J. Man-Machine Studies*, 1986, Vol. 24, pp. 301-312.

[17] K. Okumura. "A formal protocol conversion method," in *Proc. ACM SIGCOMM'86*, Stowe, VT, 1986.

[18] R. Pausch, M. Conway, R. Deline. "Lessons learned from SUIT, the Simple User Interface Toolkit," *ACM Transactions on Information Systems*, Vol. 10, No. 4, October 1992, pp. 320-344.

[19] Louis Perrochon, Roman Fischer. "IDLE: Unified W3-access to interactive information servers," *Computer Networks and ISDN System*, Vol. 27 (1995), pp. 927-938.

[20] Guido van Rossum. *STDWIN — A Standard Window System Interface*, Report CS-R8817, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

[21] Bryan Costales, Eric Allman, Neil Rickert. *Sendmail*, O'Reilly & Associates, Inc., 1993.

[22] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer interaction*, Addison-Wesley, Reading, MA, 1987.

[23] M. M. Taylor. "Layered protocols for computer-human dialogue. I: Principles," *Int. J. Man-Machine Studies*, 1988, Vol. 28, pp. 175-217.

[24] M. M. Taylor. "Layered protocols for computer-human dialogue. II: Some practical issues," *Int. J. Man-Machine Studies*, 1988, Vol. 28, pp. 219-257.

[25] Manfred Thüring, Jörg Hanneman, Jörg M. Hoake. "Hypermedia and Cognition: Designing for Comprehension," *Communications of the ACM*, August 1995, Vol. 38, No. 8, pp. 57-66.