APPLYING FORMAL METHOD IN THE IMPLEMENTATION OF THE INFORMATION RETRIEVAL PROTOCOL Z39.50

By

Binh Do

B. Sc. (Mathematics) Charles' University, Prague, 1977

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

January 1996

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Computer Science

The University of British Columbia

2366 Main Mall

Vancouver, Canada

V6T 1Z4

Date: January 04, 1996

Abstract

The success of any today large-scale software depends largely on the techniques applied to different phases of its whole life cycle.

Various formal methods and testing methods are developed to be applied to the specification phase and testing phase. Based on the author's own experience as a member of a team implementing the Information Search and Retrieval Protocol, the Z39.50, this thesis explores application of formal methods and a methodology from a practical point of views.

Table of Contents

List of Figures

Acknowledgment

First of all, I would like to express my deep gratitude to Dr. Son T. Vuong, my supervisor, for his guidance, encouragement and commitment during my time at UBC as a visiting scholar and as a MSc student. More than that I also owe him for being one of my greatest friends in my daily life.

Secondly, I would like to express my sincere thanks to Dr. Norm Hutchison for his helpful comments and careful reading of the final draft despite his very busy schedule.

I cannot forget to mention some people from EBSCO Publishing Company who introduced me to the Z39.50 protocol, helped me a lot during its implementation, encouraged me to write on Z39.50 protocol and allowed me to use some of the information related to the implementation. They are Mr. Oliver Pesch, Dr. Tadeusz Orlowski, Mr. Lech Woitojwisz, Mr. Chris Taylor and Mr. Charles Evans. My time being with them was wonderful and unforgettable. I really enjoyed working with them.

I also would like to thank Michael McAllister for the explanation of his coverage tool. I really appreciate his spending time reading the thesis draft and making very good suggestions about it, and for letting me use the machine in our office room most of the time we both were in.

My thanks also belong to Rui Zhang who did not mind answering all my boring and stupid questions about TESTGEN and who let me use her tool to generate the test cases in TTCN form.

Last but not least, I would like to thank my wife Kathy Bui and my son Jackson Do for being my wonderful family and for their endless love during the course of the thesis.

Chapter 1

INTRODUCTION

1.1 Introduction

Most protocols have been implemented and tested in ad hoc manners, especially in industrial environments. To increase the probability that different protocol implementations can inter-operate with each other, the International Standards Organization (ISO) developed the Conformance Testing Methodology and Framework [ISO-9646]. According to the conformance testing methodology, an implementation conforms to the protocol standards if it fulfills the requirements defined in the ISO 9646 standard.

The test method is based on the black-box principle. Conformance testing is carried out by using test suites which in turn consist of a complete set of test cases. The ISO also defines the Tabular and Tree Combined Notation (TTCN) [ISO-TTCN] to specify abstract test suites.

Several protocol test generation methods such as T-method [Nai-81], U-method [Sab-88], D-method [Gon-70], W-method [Cho-78] and TESTGEN [Vuo-94] have been developed to generate test suites for a given protocol. So far these methods are advocated by researchers and are used mainly in academic environments.

As a member of a software team implementing the Search and Information Retrieval Protocol Z39.50, the author would like to explore the application of one of the formal methods, namely TESTGEN to generate a test suite semi-automatically. Furthermore since this method is based on the Protocol Specification in Finite State Machine (FSM)

notation, a practical usage of FSM in coding is also explored in detail.

1.2 Motivation and objective

The success of any of today's software largely depends on the software engineering techniques applied to each phase of its whole life cycle such as specification, design, prototyping, implementation, unit and integrated testing and maintenance. The testing phase is usually very time consuming and very costly.

Using test generation methods can help to generate complete test suites quickly and possibly semiautomatically, thus reducing the cost and efforts.

The objective is to show how a formal method and its test generation tool can be applied to an implementation of a well-known real-life and complex protocol, the Information Search and Retrieval Protocol Z39.50. The advantages and disadvantages will be discussed to allow for improvements in future versions of the tool.

Beside the popularity and complexity, another reason I chose Z39.50 is that I worked on its implementation for a while and therefore had gained some knowledge about it.

1.3 Thesis outline

Chapter 2 gives an overview of the OSI Reference model and TCP/IP protocol, as well as the TESTGEN tool.

Chapter 3 introduces the Z39.50 protocol as it will be used as an example through the next chapters.

Chapter 4 introduces the application of a formal method in the implementation of the Z39.50 protocol. Emphasis is placed on using TESTGEN to generate conformance test cases in the form of subtours and TTCN, and some issues related to test coverage.

Chapter 5 presents a practical technique using Finite State Machine in the actual coding process of the Z39.50 implementation.

Chapter 6 discusses experiences, advantages and issues in the pragmatic methodology as learned from the implementation of the Z39.50 protocol. Specifically, advantages and limitations of TESTGEN tool are exposed here.

The last chapter, Chapter 7, summarizes some important issues and discusses potential relevant future work.

Chapter 2

BACKGROUND

The IR Z39.50 protocol is defined as an application layer protocol within the OSI reference model and the presence of lower-level OSI services are assumed. Many organizations and institutions implementing this protocol (including my previous employer), however, chose to layer Z39.50 directly over TCP/IP rather than implement it in an OSI environment. Therefore I include the basics of OSI as well as TCP/IP in this chapter.

The tool to generate the conformance test cases, developed by our group under the supervision of Dr. Son Vuong, is also introduced.

2.1 The OSI Reference Model

During 1970's the International Standards Organization (ISO) began working on a set of protocol standards for data communication between computers. Their goal was to develop standards which were not proprietary to a particular vendor nor of interest only to a small subset of the international community. The result of this effort is the basic model ISO Open Systems Interconnection (OSI) seven-layer model. It is described in ISO 7498 and formally called *Open Systems Interconnection - Basic Reference Model.* It deals with connecting open systems - that is, systems that are open for communication with other systems.

Figure 2.1 depicts the seven OSI layers.

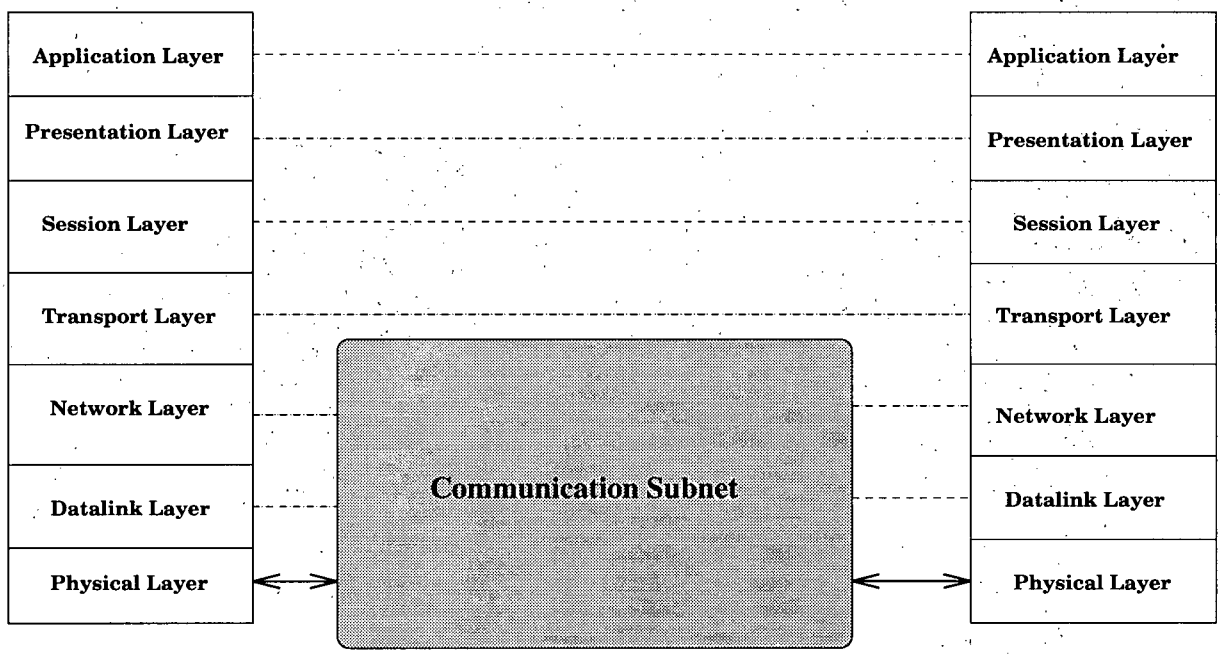

Figure 2.1: The OSI Reference Model

1. The Physical Layer is concerned with transmitting raw bits over a communication channel. It deals mainly with electrical and mechanical characteristics of the media.

2. The Datalink Layer is responsible for taking a raw transmission facility and transforming it into a line that appears free of transmission errors to the network layer. In the case of a Local Area Network (LAN), this layer is divided into two sublayers known as Logical Link Control (LLC) and Medium Access Control (MAC). LLC is responsible for error management so that it can guarantee error-free transmission while MAC detects errors in the transmission of a single frame.

3. The Network Layer is concerned with controlling the operation of the subnet. Basically, it provides addressing and routing services allowing stations, not physically connected to the same network link, to communicate with each other.

4. The Transport Layer is responsible for reliable and transparent data transfer between two end systems. It is also supposed to perform end-to-end flow control, which prevents fast transmitters from overrunning the buffer capacity of slower receivers.
5. The Session Layer provides various connection management services, specially full-duplex or half-duplex connections and graceful closing of connections.
6. The Presentation Layer is concerned with the syntax and semantics of the information transmitted. It allows heterogeneous computers architectures and operating systems to transmit complex data structures between systems and have the content and meaning of the data structures preserved.
7. The Application Layer provides a set of services accessible by an application programmer via high level interfaces. Application layer software is often responsible for the semantic verification of data transferred between systems.

In this model, there is a strict division between each of the seven layers. Each layer in the stack provides a set of services to the layer above and requests services from the layer below. The interface to each layer is a set of primitives which are part of the protocol service specification. The adjacent layers communicate to each other through *service access points* (SAP). Data is sent between systems in a packet called a *Protocol Data Unit* (PDU).

2.2 TCP/IP Protocol Suite

Even though OSI is important and complete, its development is very expensive and besides, over 95 percent of networking products are not OSI [Jam-95]. In industry, the more well-known and widely used networking protocol suite is TCP/IP (stands for

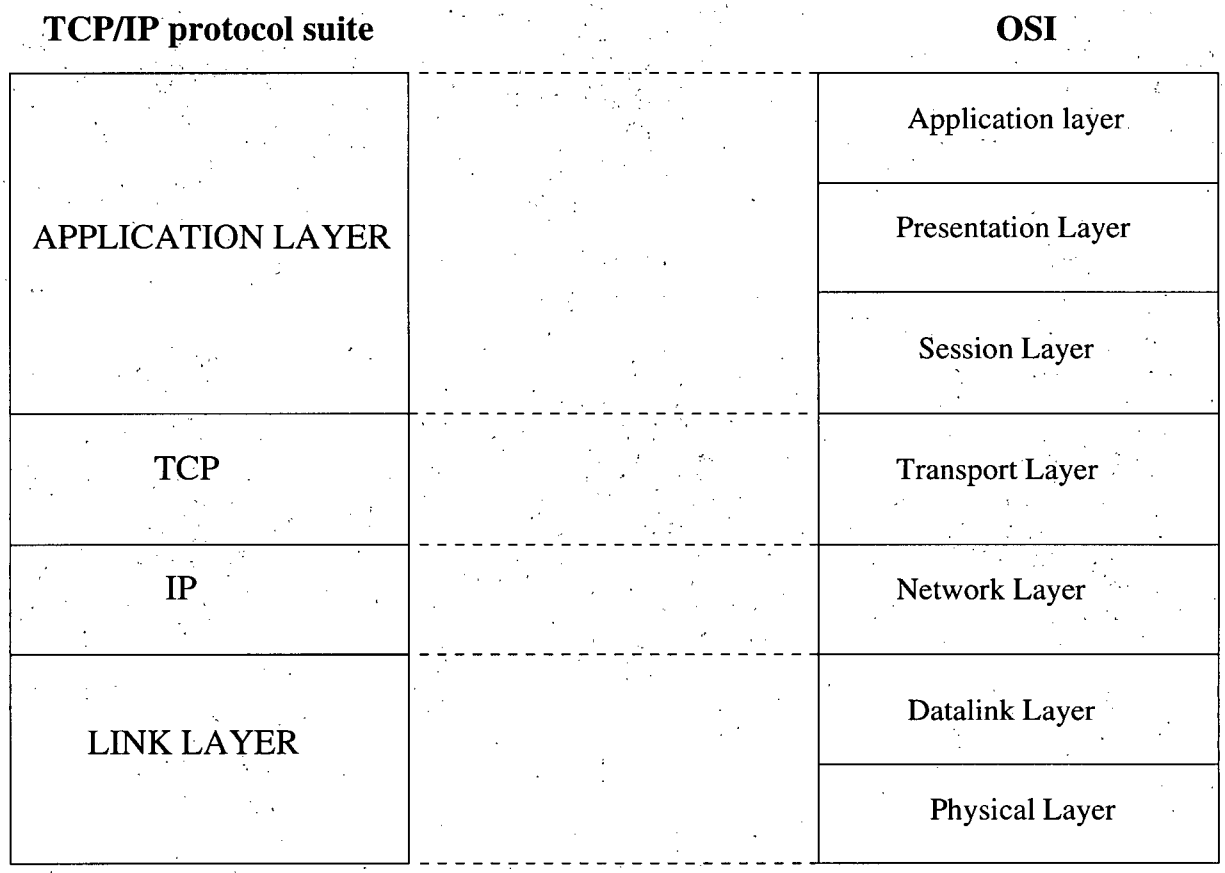

Figure 2.2: TCP/IP stack compared with OSI stack

Transmission Control Protocol/Internet Protocol), developed by the U.S. Department of Defense Advanced Research Project Agency (DARPA) in the 1970's.

The TCP/IP protocol suite also utilizes the layered architecture approach and it has four layers: Application, TCP, IP and Link layers. Figure 2.2. contrasts these layers with the layers in the OSI stack. Also note that despite the name, TCP and IP are only two of the layers.

1. The Link Layer performs similar tasks of the Data Link Layer and Physical Layer of the OSI model. It generally consists of IEEE 802 protocol, X.25 protocol or

V.32/V.35, two CCITT standards defining transmission of digital data over telephone lines.

2. The IP performs tasks similar to the ones of the OSI Network Layer, that is, to manage the switching and routing of messages across the network from one end system to the other.
3. The TCP provides a reliable transfer of data between end systems using end-to-end error recovery and flow control.
4. The Application Layer is responsible for implementing all services of the OSI session, presentation and application layers.

2.3 TESTGEN

The TEST suite Generation and selection ENvironment (TESTGEN) is developed by the University of British Columbia (see [Vuo-94]) and is a tool to generate conformance test cases for communication protocols based on their internal representation in a formal language, Estelle, and their data part representation in Abstract Syntax Notation 1 (ASN.1).

2.3.1 Motivation

In order to generate a test suite, the protocol specification must be formalized and accessible by the test generation engine. TESTGEN uses a representation based on extended Transition System (ETS) and ASN.1 to capture syntactical information (type definition of service primitives and parameters) and nondeterminism and implementation choices of the protocol.

The ETS+ASN.1 knowledge is stored in a structure called a Protocol Data Structure (PDS) and will be used by the engine to generate the test suite.

2.3.2 Internal representation of protocol

ETS

The Extended Transition System (ETS) model is based on Keller's Labelled Transition Systems [Kel-76]. Some Estelle terminology is borrowed in naming some of the ETS elements.

An *Extended Transition System* (ETS) is a quadruple $\mathbf{ETS} = (\mathbf{Q}, \mathbf{E}, \rightarrow, \mathbf{q_{init}})$ where

- $\mathbf{Q}$ is the set of states of the ETS,
- $\mathbf{E}$ is the set of events of the ETS,
- $\rightarrow \subseteq \mathbf{Q} \times \mathbf{E} \times \mathbf{Q}$ is the transition relation for the ETS,
- $\mathbf{q_{init}} \in \mathbf{Q}$ is the initial state of the ETS.

The **set of states Q** denotes the set product:

$$Q = STATES \times VAR \times C \times TIMER$$

where STATES is the set of control states and VAR is the set of data states. The control states and the data states respectively correspond to the EFSM major state and minor states. C (constants) is a set of fixed variables used to represent protocol characteristics that are invariant to the protocol execution. TIMER is the set of time constructs used to specify time in the protocol representation. TIMER predicates and operations are described in Section 2.1.4.

$\mathbf{q_{init}}$ represents the initial control state S_{init} and the initial values of all variables ($\in$ VAR) and timers ($\in$ TIMER).

The **set of events E** denotes the set union :

$$E = ISP \cup OSP \cup PDU.$$

where *ISP* is the set of Input Service Primitives (ISPs) accepted at the protocol's Service Access Points (SAPs), *OSP* is the set of Output Service Primitives (OSPs) offered at the protocol's SAPs and *PDU* represent the set of Protocol Data Units (PDUs) that can be embedded in ISPs or OSPs.

A **transition** $\xrightarrow{t}$ is represented by a pair $< P_t, F_t >$, where P_t is the *enabling predicate* on the set product $Q \times E$ and F_t is the *action function* on the set product $Q \times E$, which affects variables in VAR, timers in TIMER and parameters of OSPs and PDUs.

A transition can be executed if and only if the ISP and PDU associated with the transition (if any) are received and if the enabling predicate is true, or a timer expires. When a transition fires the associated action function is executed atomically. Variables and timers are set, OSP(s) and PDU(s) are assembled (their parameters are set) and sent.

The semantics of the enabling predicates and the action functions are similar to the semantics of the Estelle *enabling clauses* and *statements* except for two important protocol aspects: the data part of ISPs, OSPs, PDUs and the time handling.

ASN.1 Data Part

The Abstract Syntax Notation 1 (ASN.1) is an abstract notation [Neu-92] for describing data objects of different OSI protocols as well as non-OSI protocols and applications. It has been widely used in describing the structure of Service Primitives (SPs) and PDUs in higher level protocols such as X.400 (MHS), X.500 (DS), FTAM and SNMP. ASN.1 has also proven viable for the specification of the data part of lower level OSI protocols

[Neu-92]. There are several reasons for using ASN.1 for representing the data part of protocols for the purpose of test suite generation :

- ASN.1 is a standardized abstract syntax notation supported by ISO.
- Many higher level protocols have their PDUs represented in specified in ASN.1.
- ASN.1 is supported in TTCN.
- Many useful tools to convert ASN.1 to other programming language data structures (such as C and C++) are available [Neu-90] [Sam-90].

TESTGEN adopted a subset of the ASN.1 notation defined in the 1988 version of the X.208 recommendation [ISO-8824] to specify the structure of service primitives (ISPs and OSPs) and protocol data units (PDUs) in the ETS formalism. In order to represent a communication protocol in the Protocol Data Structure all the ISPs, OSPs and PDUs of the protocol must be specified in ASN.1. TESTGEN specifies the parameter structure of service primitives and protocol data units with a subset of the basic ASN.1 type notation defined in section 1 and section 2 of [ISO-8824]. In the Extended Transition System (ETS), service primitives and protocol data unit parameters are referenced by enabling predicates and action functions. TESTGEN introduces an intuitive dot notation similar to the PASCAL or C structured type reference mechanism. Thus the SP or PDU parameters described in the ASN.1 part of the protocol specification can be referenced by enabling predicates and action functions as follows:

$$< SPname > \{. < parametername >\}^{+} \qquad or$$

$$< PDUname > \{. < parametername >\}^{+}$$

Note that this notation requires the enforcement of a strong naming convention: all SP parameters and subparameters (usually structured sequences or set types) must be

unambiguously named.

Test suite generation method

TESTGEN generates a test suite (for black-box testing) which verifies an arbitrary number of specified conditions on the external behavior of a protocol implementation for a selected set of subtours (a subtour is a tour of the protocol, which starts and ends at the same state, the initial state) of the protocol graph. A separate test case is generated for each subtour.

Each subtour starts and ends at the initial state, as the initial state is the most trustworthy state and can easily be reached (e.g. with *"reset"* operation), and besides such subtours are natural and meaningful for *"user sessions"* during testing.

In TESTGEN, a subtour can be printed on user's demand. In its format, the states are in the rectangles; a transition from one state to another is represented by vertical bars; the names of the input service primitives or input PDUs (and their parameters) are on the left side; the names of output service primitives or output PDUs are on the right side. The following is an example of a subtour generated by TESTGEN for the simplified Z39.50 protocol:

```
                                  +--------+
                                  | CLOSED |
                                  +--------+
                                       |
ISP: AEINITreq                         |OSP :---
     { parameters...                   |
     }                                 |
IPDU:---                               |OPDU1:INITREQ
```

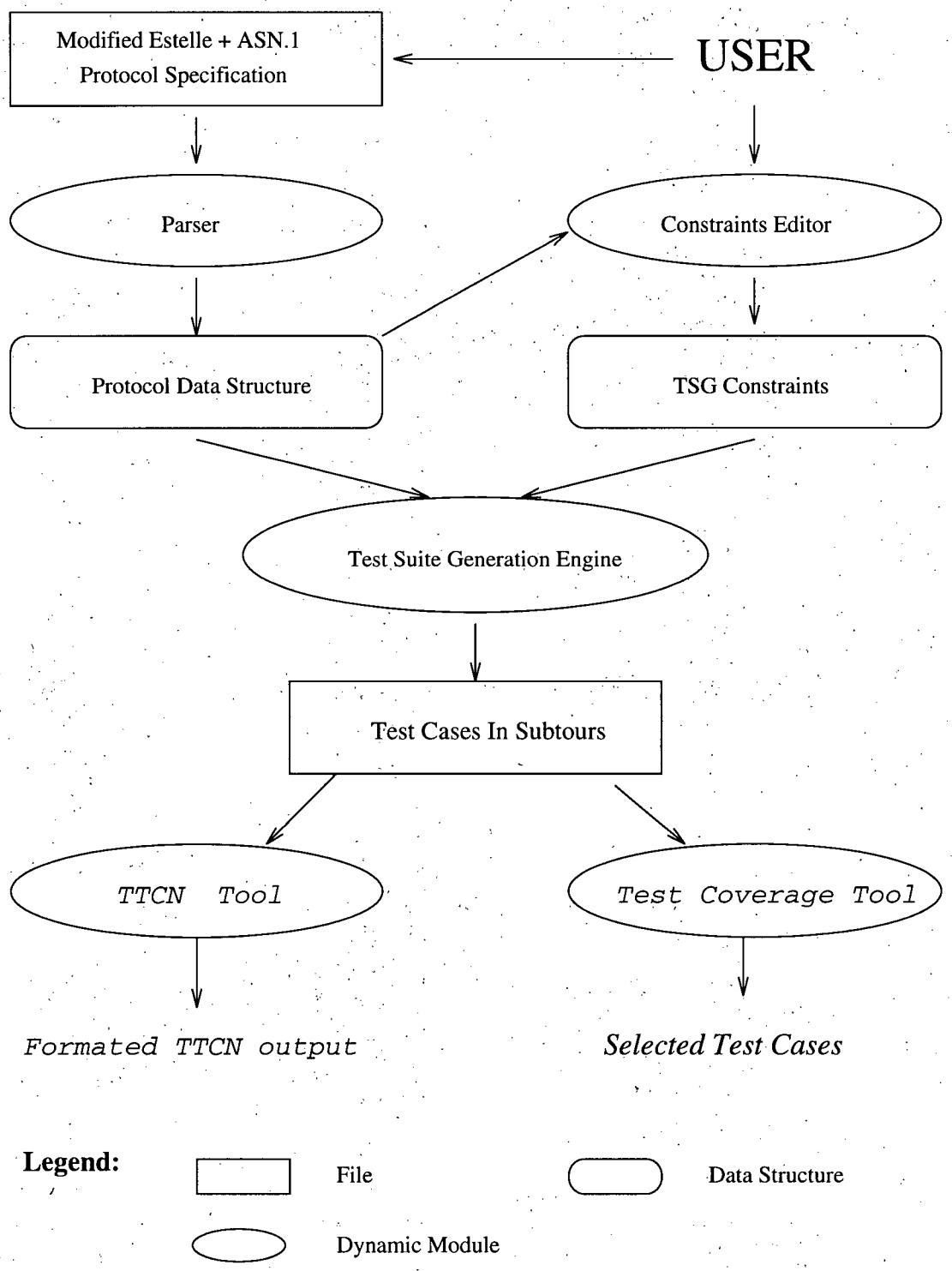

Figure 2.3: The process of generating test cases using TESTGEN

```
                                        |     { parameters...
                                        |     }
                                        |OPDU2:---
                                        |
                                   +-----V-----+
                                   | INIT_SENT |
                                   +-----------+
                                        |
ISP: ---                                |OSP1:AEINITconf
                                        |     { parameters...
                                        |     }
                                        |OSP2:---
                                        |
IPDU:INITRESP                           |OPDU :---
     { parameters...                    |
     }                                  |
                                     +---V--+
                                     | IDLE |
                                     +------+
                                        |
ISP: AESEAHreq                          |OSP :---
     { parameters...                    |
     }                                  |
IPDU:---                                |OPDU1:SEAHREQ
                                        |     { parameters...
                                        |     }
```

```
                                        |OPDU2:---
                                        |
                                  +-----V-----+
                                  | SEAH_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AESEAHconf
                                        |     { parameters...
                                        |     }
                                        |OSP2:---
                                        |
IPDU:SEAHRESP                           |OPDU :---
     { parameters...                    |
     }                                  |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
ISP: AECLOSreq                          |OSP :---
     { parameters...                    |
     }                                  |
IPDU:---                                |OPDU1:CLOSREQ
                                        |     { parameters...
                                        |     }
                                        |OPDU2:---
                                        |
```

```
+----V---+
| CLOSED |
+--------+
```

Since communication protocols usually contain recursive behaviour (state recursion), as well as parameter variation (slight changes of data in different PDUs) on the exchanged service primitives usually leads to a practically infinite number of parameter value combinations, the number of subtours can be infinite. To overcome this difficulty, TESTGEN defines a flexible, user-definable restriction mechanism so-called TSG constraints. Basically, the TSG-constraints allow the user to define a minimal and a maximal value on the number of times an ETS element is reached or used in one subtour. For example, to test the data part of an implementation the user can request that the IDLE state can be reached between 2 and 4 times, the DATA PDU can be used for at most 3 times.

At the beginning, the constraints are set to default values when the PDS is created. Depending on the generated test results, the user can invoke the editor to define the values he/she would like to have and then go and generate the tests again. More information about the TSG-constraint editor can be found in [Vuo-94].

After subtours are identified by TESTGEN they can either be reduced by a coverage tool [Vuo-91], [McA-92] or they can be used to generate test cases in the form of TTCN [Zha-95].

The diagram in Figure 2.3 describes the process of generating test cases using TESTGEN and other supplemental tools.

Chapter 3

INFORMATION RETRIEVAL PROTOCOL Z39.50

Z39.50 (IR) is an ANSI/NISO standard and is formally called **ANSI Z39.50; Information Retrieval Service and Protocol. American National Standard Information Retrieval Application Service Definition and Protocol Specification for Open Systems Interconnection**. The documents of IR are prepared by a NISO (National Information Standards Organization) committee. IR is an application protocol based on a client/server model.

The Z39.50 IR protocol was originally proposed (in 1984) for use with bibliographic information and was later broadened in its domains of applications as manufacturers, vendors, consultants, information providers, universities and others express their interest and they wished to access or to provide access to various types of information, including bibliographic, text, image, financial, public utility, medical and scientific information, chemical, news services and much more.

The version of this protocol we implemented is version 3 released in 1994.

3.1 Basics Of The Protocol

Z39.50 IR is based on an underlying connection-oriented and reliable network. The protocol specifies formats and procedures governing the exchange of messages between a client and server; enabling the client to request that the server search one or more databases and identify records which meet specified criteria, and to retrieve some or all

of the identified records.

We should note that the protocol only addresses communication between the client and the server. It does NOT address the interaction between the client and user, nor the interaction between server and the information databases.

The protocol is logically divided into the procedures pertaining to the client and the procedures pertaining to the server.

Basically, IR permits clients to search databases residing on servers and then retrieve records from these databases. In more details, we can describe IR as follows: IR connects a user of an IR client to an IR server and disconnects when the user is finished. During connection, the user will request some specific information which will be expressed somehow in a form of requests sent from client to server for processing against the server's databases or resources. As an example, the user may request for magazines issued in a specific time frame, or articles written by a particular author, about specific subjects and so on. Based on the results, the user may request retrieval of one or more records.

Within the protocol, the portions of the client and server that carry out those procedures are referred to as the *Z39.50 origin* and the *Z39.50 target*, respectively. They communicate with each other by exchanging *protocol data units* (PDUs). The origin and the target are not interchangeable.

The standard describing Z39.50 is positioned with respect to other related standards by the Open Systems Interconnection (OSI) basic reference model (ISO 7498). Z39.50 protocol is defined within the application layer of the reference model, and is concerned in particular with the search and retrieval of information in databases. The relationship of Application Entities (AE) and Service Providers (SPs) for Z39.50 can be depicted in the model in Figure 3.4. This model formalizes the communications between various

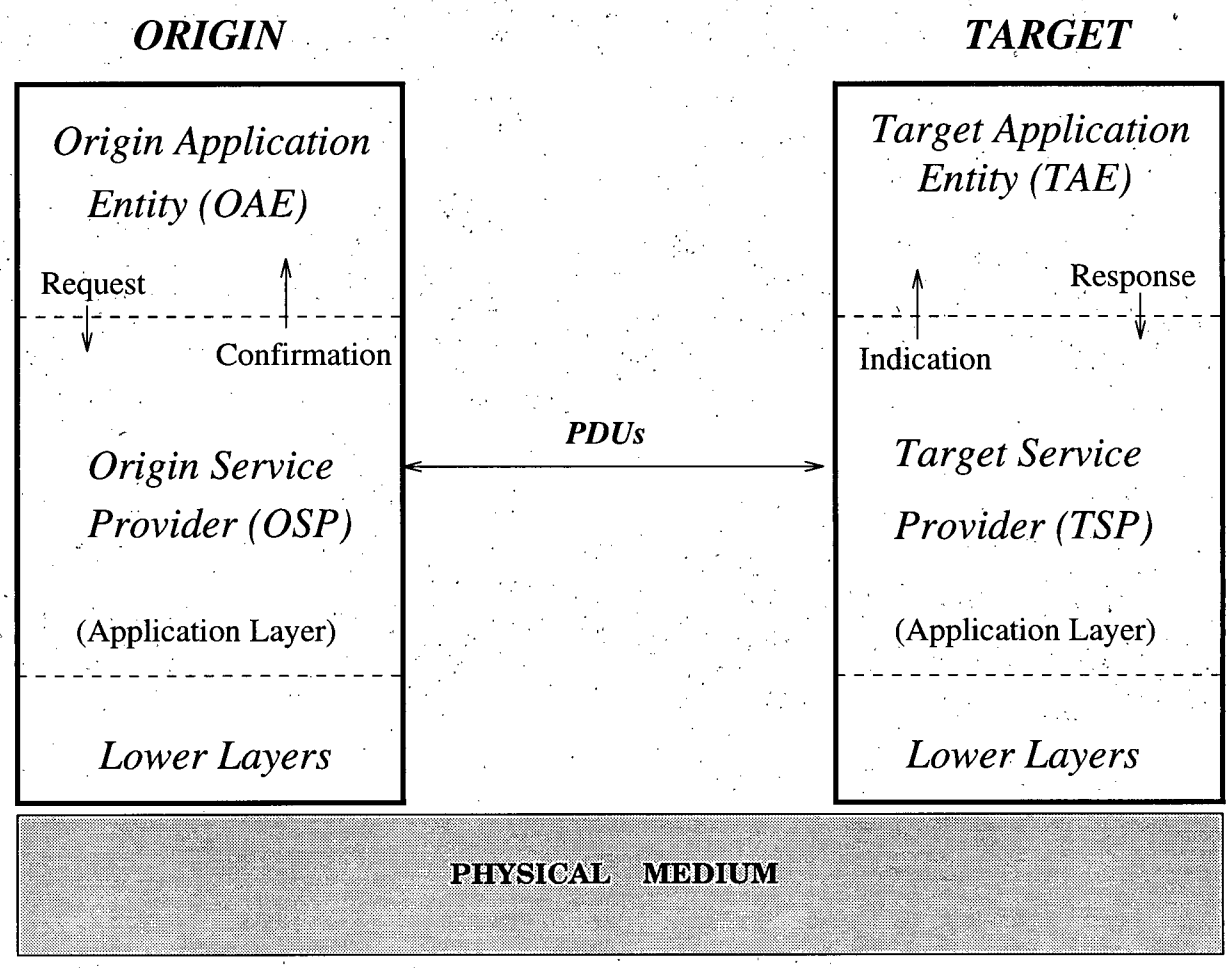

Figure 3.4: The relationship of Application Entities and Service Providers in Z39.50 parts of an OSI supportive system.

In this model, each system (i.e. Origin or Target) is made up of a *service provider* (SP) and an *application entity* (AE). The SP is the application layer software that builds, sends, decomposes and verifies the Z39.50 PDUs. The AE is the software residing outside of the OSI model that makes use of the Z39.50 services provided by the application layer software to search for and retrieve data, in the case of the origin, or to accept requests to perform searches and to provide the result, in the case of the target.

The transfer of a PDU is initiated by one of the AEs issuing a *request* to its SP, specifying the PDU to be transmitted and also specifying any data necessary to create that PDU. The SP reacts to the request by building the PDU using the data provided by the AE in the request and by sending the PDU using the services provided by the lower layer software. The SP within the partner system accepts, decomposes and verifies the PDU, then issues an *indication* to its AE specifying the received PDU and passes data contained in the PDU.

The AE initiates a return PDU by issuing a *response* to its SP specifying the PDU to be sent and specifying any data necessary to create the PDU. The SP creates and sends the PDU. The SP in the partner system receives, decomposes and verifies the PDU, then issues a *confirmation* to its application entity specifying the received PDU and passes data contained in the PDU. Note that in this model, we consider the AEs as a layer above the SP which resides in the application layer of the OSI model.

Even though Z39.50 is defined in terms of the OSI reference model and is considered to fit within the application layer, many institutions implementing it chose to layer it on top of the de facto standard Transmission Control Protocol/Internet Protocol (TCP/IP) for several reasons:

- An OSI implementation could be to large and too complicated. This can outweigh the relatively meager return in functionality the organizations would like to gain.
- OSI supplemental softwares are lacking.
- One of the main goal of implementations is the interoperability with each other over the Internet and implementations over TCP/IP obviously have more advantages in this respect. In fact, a couple of years ago two organizations, the Florida Center For Library Automation and the Data Research Associates, implemented the earlier

version of Z39.50 using OSI stack but they could not talk to each other as they accessed two different OSI networks that were not connected [Jam-95].

3.2 Information Retrieval (IR) Service

This service describes the activity between two applications: an initiating application, the client, and a responding application, the server. The server is associated with one or more information databases.

In the following subsection, I informally describe major services in Z39.50; these services are defined more formally, as in the standard, in the next subsections.

3.2.1 Z39.50 services and their usage

Services in Z39.50 can be informally divided into the following categories:

- Setting up and tearing down connection between the target and the origin (Init and Close).
- Searching and retrieving data from databases (Scan, Search, Present and Segment).
- Checking resources and authority during the whole session (Resource-Control, Access-Control, Trigger-Resource-Control).
- Maintaining the result sets during the session (Delete, Sort).
- Special services allowing Origin to know about the Target implementation, databases information or allowing Target to perform tasks external to the protocol (Explain, Extended).

Before the Z39.50 Origin and the Z39.50 Target can communicate to each other, the *Init Service* must be used to set up the connection. Eventually after the session is

finished, the connection will be closed by the *Close Service.*

One of the most important services is the *Search Service.* It provides a mechanism to search one or more databases on a remote system using a wide variety of selection criteria and uniform search query structures.

The results of a search then can be carried out by another service called the *Present Service.* The search results are maintained in a result set consisting of a set of pointers to the physical records satisfying the criteria. Therefore the present service only needs to know the starting point in the set and the number of records from that point to be returned to complete its task. Note that in this way, a random access retrieval mechanism is provided.

Z39.50 provides a service called the *Scan Service* for browsing a list of available items of the databases thus easing the subsequent searches.

To adapt to the system resources efficiently and to assure the authorization, Z39.50 uses the *Resource-Control Service*, the *Resource-Report Service*, the *Trigger Resource-Control Service* and the *Access-Control Service.*

The *Sort Service*, the *Delete Service* are utilized to maintain the set of results after a search is done.

To allow the Origin to get further information about the Target such as databases available for searching, attribute sets and diagnostic sets used by the Target, record syntax and so forth, Z39.50 provides the *Explain Service.*

Another service called *Extended Services Service* allows Origin to request Target to perform some specific task. This service usually is used by the implementors for their own specific tasks. For example, an Ordering or Purchasing System via the net can use this kind of service.

3.2.2 Model and Characteristics of the Information Retrieval Service

Communication between origin and target (also called association) is explicitly established by the origin and may be explicitly terminated by either origin or target.

Z39.50 assumes that the underlying connection is connection-oriented and reliable. The roles of origin and target may not be reversed.

3.2.3 Types of Z39.50 services

The Z39.50 service is carried out by the exchange of messages between origin and target. Each message is either a request or response. Services may fall to one of 3 following types:

1. Confirmed: in this service, a request must be followed by a response.
2. Non-confirmed: in this service, a request does not have a response to it.
3. Conditionally-confirmed: this service can be invoked either as a confirmed or a non-confirmed service.

3.2.4 Z39.50 Operations

There are eight operations in version 3: Initialization, Search, Present, Delete, Scan, Sort, Resource-report and Extended-services.

A request from the origin of a particular operation type initiates an operation of that type (e.g. Search request initiates a Search operation) which terminates with a corresponding response from the target.

3.2.5 Facilities of the Z39.50 service

There are eleven facilities of the service. Most of them consist of logical groups of services; in a few cases, a facility consists of a single service. Their names and characteristics are described in Figure 3.5.

In more detail, the facilities are:

1. Initialization Facility: consists of the Init Service and it is a confirmed service initiated by Origin. This service not only initiates a connection but also determines QOS parameters as well as some functionalities to be used during the whole connection. (e.g. how large a packet should be, which services are available and so forth).

2. Search Facility: a confirmed service initiated by Origin via the Search operation. Search service permits an Origin to search for particular records in a remote server database. The search may be performed by personal, corporate and conference names, titles, dates, bibliographic numbers such as ISSN, ISBN and so on. Once the search is done, a *Result List* is created and maintained by the Target for later use.

3. Retrieval Facility: consists of two services:

   - Present Service, which is confirmed and initiated by Origin to launch a Present Operation. The Present Service allows Target to retrieve records matching the search and send back to Origin, which in turn presents these results to users.

   - Segment Service, which is non-confirmed and initiated by Target during a Present operation. This is used when Origin requests for very large records which cannot be conveyed in one packet (PDU).

Characteristics / SERVICE NAME	Confirmed	Non Confirmed	Condition-Confirmed	Originator
Init	X			Origin
Search	X			Origin
Present	X			Origin
Segment		X		Target
Scan	X			Origin
Delete	X			Origin
Sort	X			Origin
Access-Control	X			Target
Trigger-Resource		X		Origin
Resource-Control			X	Target
Resource-Report	X			Origin
Extended	X			Origin
Close			X	Target - Origin

Figure 3.5: The Z39.50 services and their characteristics

4. Result-set-delete Facility: consists of the Delete Service, confirmed and initiated by Origin invoking Delete Operation. This is invoked when the Origin asks to delete one or more records from the given Result Set.

5. Browse Facility: consists of Scan Service, a confirmed service originated by Origin launching a Scan Operation. This allows the user to look at some ordered list of terms before he/she wants to perform search and retrieval on some specific terms. As an example, the list may be composed of subjects, authors, names and so on.

6. Sort Facility: consists of the Sort Service, confirmed and initiated by Origin invoking a Sort Operation. This permits Origin to ask Target to sort the result list in particular sequences, such as author/title or subject/date.

7. Access Control Facility: consists of the Access Control Service, a confirmed service and initiated by Target and not invoking any operation. This service usually is a part of another operation. This provides a method of controlling access to databases on remote server and allows Target to serve many users with different levels of authorization.

8. Accounting/Resource Control Facility: consists of three services:

   - Resource-control Service: a conditionally-confirmed service initiated by Target not invoking any operation and which is usually part of another operation.
   - Trigger-resource-control Service: a non-confirmed service initiated by Origin invoking no operation, and which is part of another operation.
   - Resource-report Service: a confirmed service initiated by Origin invoking the Resource-report operation.

9. Explain Facility: does not include any services but uses the services of the Search and Retrieval facilities. This allows the user to obtain details of the nature and extent of the databases available on a remote server. It includes not only the number or the names of the databases but also the information about how to search against them.

10. Extended Services Facility: consists of the Extended-services service, confirmed and originated by Origin invoking the Extended-services operation.

11. Termination Facility: consists of the Close Service, confirmed and which may be initated by either Origin or Target. It does not invoke any operation, nor it is part of any operation. It merely allows Origin or Target to immediately terminate all operations and subsequently finish the current connection.

3.2.6 A typical scenario of Z39.50

Although the protocol is very complicated due to various external services, I just want to present a scenario which is quite typical and common of Z39.50 usage:

The origin explicitly establishes the connection by sending an Init-Request to the target. After getting Init-Response with positive result (i.e. accept), the connection is established and PDUs will be exchanged between the origin and the target.

Usually the origin will begin sending a Search-Request specifying the query in a predefined syntax (e.g. Reverse Polish Notation, ISO8777 or Z39.58 type query and so on). Alternatively, prior to sending Search-request, origin may send a Scan-Request to request Target to send back a list of general terms. Based on that list, origin may form a more reasonable query.

On receipt of Search-response containing the result set of pointers to the records

satisfying the query, the origin will request the target to display some or all records in more details or in the full-text format by sending a Present-request specifying the starting point in the list and the number of records from that point it wants to retrieve.

Eventually the origin should disconnect the connection by sending a Close (understood as a request, though in Z39.50 term, it is just called Close) to the target. It may but does not have to wait for a Close (-response, as an acknowledgment) from target before closing down completely.

Figure 3.6 describes the above scenario.

3.3 The implementation of Z39.50 at EBSCO Publishing company

In the next chapter, I will use a formal method and other tools to generate conformance test suites for Z39.50 which can be tried against some implementation. So I would like to first introduce briefly the implementation I had chance to participate in.

3.3.1 Objective

From September 1994 to August 1995, I was involved in the implementation of Z39.50 for EBSCO Publishing company. This implementation is for library use. The goal is to implement a client/server based software (based on Z39.50) allowing users in different libraries in North America to access and retrieve bibliographic information stored in main databases located on a certain server. On one hand the implementation must follow closely the Z39.50 specification (so that it can inter-operate with other vendors' implementation), on the other hand it also must accommodate the specific needs of the EBSCO company.

The implementation (until August 1995) includes almost all facilities except Explain,

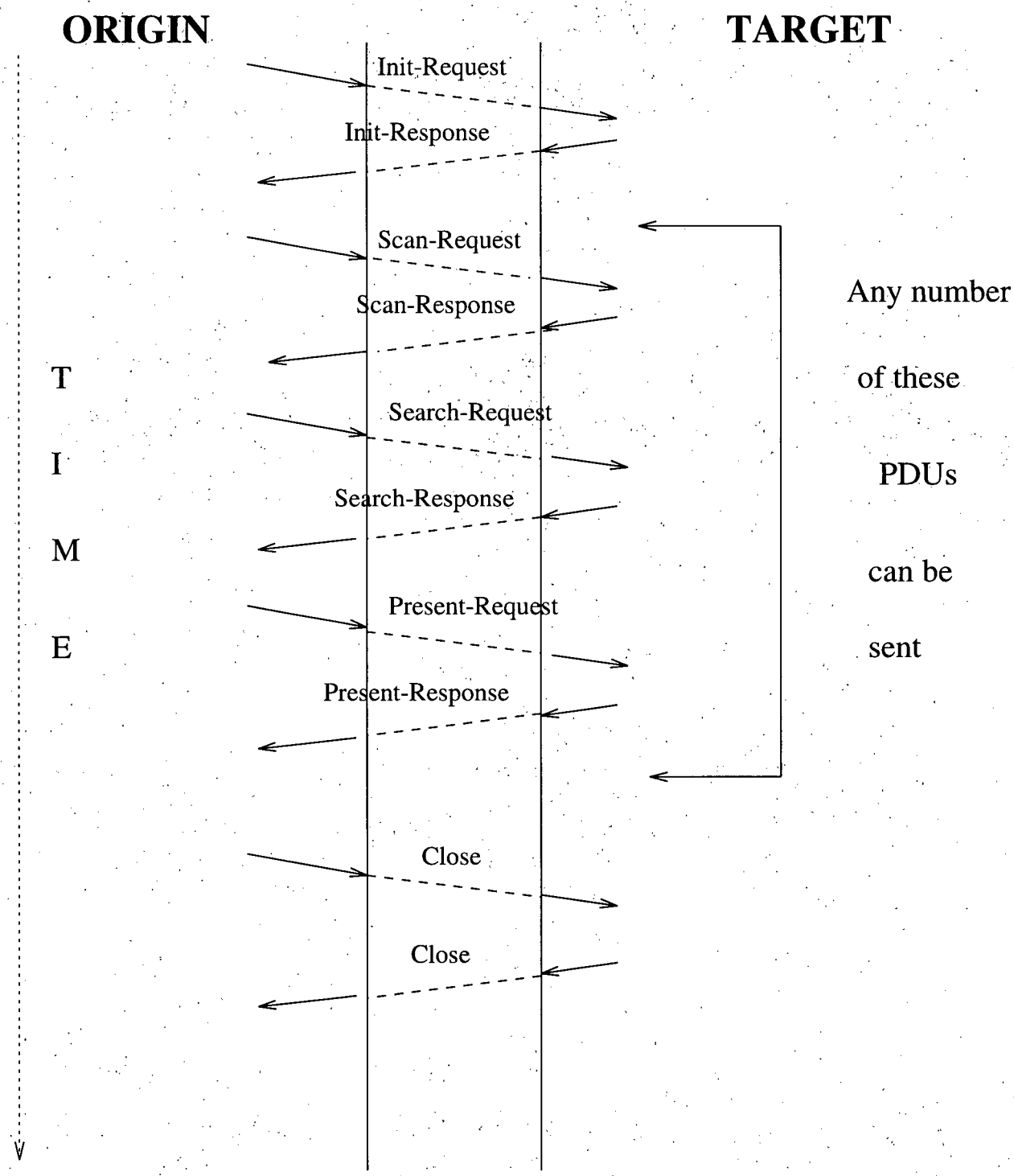

Figure 3.6: A typical scenario of Z39.50

Delete and Sort.

3.3.2 Programming language and tools used in the implementation

We use C (on the target) and C++ (on the origin) to implement Z39.50. We also use the ASN.1 to C compiler "snacc" developed by the University Of British Columbia [Sam-93] and slightly modified (memory allocation and ported to DOS and MS-Windows platforms) to encode and decode the data part in PDUs.

The implementation of the target system alone consists of about 75000 lines of C-code, not including code generated by the "snacc" compiler and code related to sending and receiving data to and from the origin. The target runs on HP Unix system and accommodates the origin running on different platforms such as MS-Windows, Unix and MS-DOS.

Chapter 4

APPLYING TESTGEN TO Z39.50 PROTOCOL

4.1 Motivation for applying TESTGEN

The development of Z39.50 protocol I was involved in could be divided into the following stages:

- System design and software design, which is based mainly on the protocol specification in the standard and the needs of our customers. In this stage we determined the hardware platforms as well as the the services to be included in the initial version.
- Implementation and unit testing, in which a set of programs and program units (such as search engine, retrieval engine, stuffing ASN.1 in PDUs, sending and receiving PDUs between Origin and Target, user's interface and so on) are implemented according to the design and tested.
- Integration and system testing, in which all individual programs and units are merged together into a complete system and this system is tested to insure that all software requirements are met.
- Operation and maintenance, in which the delivered system is installed and improved with performance and new features.

From my own experience, testing phase (both unit testing and integration testing) takes a lot of time and efforts, specifically when no test suite is given in the standard

and all test cases are developed almost heuristically and manually.

Coming back to UBC, I think that we can apply some of the tools developed by our group [Vuo-94] to generate test suite automatically or semi-automatically to speed up this phase.

As I was assigned to implement the portion of coding, decoding data in ASN.1 format and sending and receiving PDUs, I chose to use TESTGEN to generate test cases during the "Implementation and unit testing" phase for several reasons:

- These test cases can be mainly used for the prototype implementation to make sure that the control part is functioning correctly in term of exchanging PDUs.
- The prototype can use the typical scenario described in the previous chapter, thus simplifying things but not losing the essence of the protocol.
- The data part is not very critical in this phase to the extent that some fields can be considered fixed.
- The prototype program can be written as a single unit (e.g. Origin including simulated Target inside) and therefore can be suitable for TESTGEN as it currently supports a single Estelle module.

4.2 The simplified Z39.50 used in TESTGEN

I use the typical scenario of Z39.50 described in Chapter 3 for use with TESTGEN. The reasons for this are:

- The prototype program includes the services in that scenario.
- The emphasis is placed on the control part and not the data part.

- It is possible to ask the company to allow me to use that prototype to test against the test suite in future.

The services included in this simplified version are:

- Initialization
- Search
- Present
- Close

The communication takes place in three different phases: the *connection setup phase*, the *search and retrieval phase* and the *disconnection phase*. After connection phase is established, the search and retrieval phase will be carried out: the Origin will send a search request specifying a query and a database name; only if the search is successful, a present request can be sent subsequently to retrieve the list of records. This process can be repeated until the Origin wishes to disconnect by sending the close request.

In each phase and each operation, only certain PDUs and service primitives are meaningful. Unexpected PDUs and service primitives will cause the system to disconnect for security reasons. To represent this special case, a new PDU called INVALID is introduced in this simplified version.

4.2.1 Estelle.Y and ASN.1 specification

Estelle.Y specification

As TESTGEN can only be applied to a single Estelle module, I modeled the Origin system for it. The finite state machine representing its behavior is depicted in Figure 4.7.

The FSM has 5 states which are defined as follows:

1. CLOSED (marked as Z01): initial state, origin is awaiting the Init request from the user.
2. INIT-SENT (Z02): Init request PDU was sent, awaiting Init response PDU from target.
3. IDLE (Z03): Connection is established, awaiting Search request or Present request from the user.
4. SEAH-SENT (Z04): Search request PDU was sent, awaiting Search response PDU from target.
5. PRES-SENT (Z05): Present request PDU was sent, awaiting Present response PDU from target.

From a particular state, when getting the corresponding request from user or response PDU from target, origin is preparing the corresponding PDU or confirmation to send to target or to user and then entering new state (defined in the FSM).

The Estelle.Y specification can be found in appendix A.

ASN.1 specification

The corresponding protocol data units (PDUs) exchanged between the Origin and Target (protocol layers) and their basic parameters are as follow:

PDU name	Meaning	Parameters	Respective SP's
INITREQ	Connection Setup	Protocol Version	AE-INITreq
		Options	AE-INITind
		Message Size	
		Max Record Size	

STATE / INPUT	CLOSED (Z01)	INIT-SENT (Z02)	IDLE (Z03)	SEAH-SENT (Z04)	PRES-SENT (Z05)
AE-INITreq	Send INITreq PDU (→ Z02)				
INITresp PDU (accept)		Issue positive Init Confirmation (→ Z03)			
INITresp PDU (reject)		Issue negative Init Confirmation (→ Z01)			
AE-SEAHreq			Send SEAHreq PDU (→ Z04)		
SEAHresp PDU				Issue Search Confirmation (→ Z03)	
AE-PRESreq			Send PRESreq PDU (→ Z05)		
PRESresp PDU					Issue Present Confirmation (→ Z03)
AE-CLOSreq			Send CLOSreq PDU (→ Z01)		
INVALID		Send CLOSreq PDU (→ Z01)	Send CLOSreq PDU (→ Z01)	Send CLOSreq PDU (→ Z01)	Send CLOSreq PDU (→ Z01)

Figure 4.7: The FSM for simplified Z39.50 protocol, used in TESTGEN

INITRESP	Connection Confirm	Protocol Version Options Message Size Max Record Size Result	AE-INITresp AE-INITconf
SEAHREQ	Search Service (request)	Database Name Query Result Set Name	AE-SEAHreq AE-SEAHind
SEAHRESP	Search Service (response)	Result Count Number Of Recs returned.	AE-SEAHresp AE-SEAHconf
PRESREQ	Present Service (request)	Starting Point Number Of Recs requested Result Set Name	AE-PRESreq AE-PRESind
PRESRESP	Present Service (response)	Number Of Recs returned Present Status	AE-PRESresp AE-PRESconf
INVALID	Unexpected PDU or service primitive	---	---
CLOSREQ	Termination	Close Reason	AE-CLOSreq AE-CLOSind

The ASN.1 representation for main PDUs are as follows:

```
PduMessages ::= CHOICE
{
   INITREQ,
   INITRESP,
   SEAHREQ,
   SEAHRESP,
   PRESREQ,
   PRESRESP,
   CLOSREQ,
   INVALID
}

INITREQ ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,
   maxRecordSize     INTEGER
}

INITRESP ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,
   maxRecordSize     INTEGER,
```

```
   result            INTEGER
}

SEAHREQ ::= SEQUENCE
{
   databaseName      DataString,
   query             DataString,
   resultSetName     DataString
}

SEAHRESP ::= SEQUENCE
{
   resultCount       INTEGER,
   numOfRecsReturned INTEGER,
   searchStatus      INTEGER
}

PRESREQ ::= SEQUENCE
{
   resultStartPoint  INTEGER,
   numOfRecsReqested INTEGER
}

PRESRESP ::= SEQUENCE
{
   numOfRecsReturned INTEGER,
```

```
   presentStatus      INTEGER
}

CLOSREQ ::= SEQUENCE
{
   closeReason        INTEGER
}

INVALID ::= SEQUENCE
{
   dummy              INTEGER
}
```

Appendix B contains the complete ASN.1 specification for our simplified Z39.50 protocol model.

4.3 Results

4.3.1 Initial result

By default, TESTGEN imposes several constraints on the PDS it parses at the beginning from Estelle.Y and ASN.1 specifications. For example, all states will be traversed at most once; requests and/or PDUs can be sent at most once; the integer components are assigned the values of 1 and 99; string components have one value, etc [Vuo-94].

With the default constraints, I got nearly 200 test cases. They are not good as they do not reflex the reality, for example no "Present" is sent. The reason is that "Present" is supposed to be sent only after a successful "Search" operation, and if so the "IDLE"

state must be traversed twice, which contradicts the default constraints.

So at first, I allowed states "IDLE", "SEAH-SENT" and "PRES-SENT" to appear 4 times (the number of retrieval operations which may occur in a subtour); and the number of cases is more than 15000.

4.3.2 Constraint modification

Obviously, we must impose some reasonable constraints on the PDS. The following constraints were modified in the PDS after it was parsed by the TESTGEN engine:

- As a protocol can send search or present requests many times, the state IDLE should be allowed to appear in a subtour more than once. I chose 4 for the start so that we can allow up to 4 retrieval (search or present) operations in all. Besides, as after a unsuccessful search, another search should be sent as present operation can be launched only after a successful search, so IDLE shoudl be traversed 4 times in this case. The states "SEAH-SENT" "PRES-SENT" are also allowed to be traversed maximally 4.
- The value of "ProtocolVersion" and "Options" in the Init request can be set fixed as they are fixed in the prototyping phase.
- The value of "Result" in Init response can be assigned to one of 2 possible values (0 for success and 1 for failure). Similarly, "searchStatus" is also assigned to one of 2 values, "success" or "failure".
- The values of "preferMessageSize" and "maxRecordSize" in Init-Request and Init response are also assigned to 2 reasonable values (1024 and 2048).

- The number of times some ISPs (e.g. AESEAHreq, AEPRESreq), OSP (AESEAHconf, AEPRESconf), PDUs (SEAHREQ, PRESREQ, PRESRESP) are also modified to be used maximally 4 times (instead of once).
- The values of "resultStartPoint" and "numOfRecsReqested" in "Present request" are set fixed.

Appendix C contains the modifications made above.

With these constraints, the number of test cases dropped to about 300 test cases. Some of them are listed in appendix D.

These test cases should be screened again as some of them are not executable (do not make sense). For example, according to the protocol, the "preferMessageSize" must be less than or equal to the "maxRecordSize" in the Init-Request and/or the Init-Response. However, TESTGEN does not care about that and so, subtours violating this condition must be eliminated from the test suite. In this special case, the number of test cases will be reduced to 210.

4.3.3 Test coverage

Even with imposed constraints on the minimum and maximum number of times a particular state, transition, ISP, OSP, PDU can be used to form subtours, the number of test cases generated by TESTGEN is still large.

The reason is as follows: the control part of a test case consists of the action sequence affected by the environment, e.g. by requests from user or various messages from adjacent layers. Consecutive actions may be identical and may originate from a single source (data packets arriving from the network) or from multiple sources (connection request from several users). The control sequence is concerned only with the actions and their order

and not with the source of the actions. Data variation for each control sequence also increases the number of test cases.

The idea of test coverage is to define a reasonable (user-defined) metrics distance between two control sequences (test cases) and all test cases which are within some distance are considered as sufficiently similar that only one test case is needed to be tested and thus it is selected and put in the set of selected test cases. The details of test coverage can be found in [Vuo-91], [McA-92].

Adopting [Vuo-91], we use the distance of the Z39.50 test cases generated by TESTGEN as follows:

1. For each subtour, we use the "concise notation" by grouping consecutive identical input PDUs (IPDU) or ISPs into a single tuple: the IPDU/SP I and its multiplicity (recursion depth) α giving the tuple (I, α). For example, the subtour AEINITreq - INITRESP - AESEAHreq - SEAHRESP - AEPRESreq - PRESRESP - AECLOSreq will be represented by the concise form as (AEINITreq,1) (INITRESP,1) (AESEAHreq,1) (SEAHRESP,1) (AEPRESreq,1) (PRESRESP,1) (AECLOSreq,1).

2. In this context, we are referring to a subtour by its concise form. We use the testing distance between two subtours $A = (a_1, \alpha_1) \ldots (a_n, \alpha_n)$ and $B = (b_1, \beta_1) \ldots (b_n, \beta_n)$ (if A and B are of different lengths then the shorter subtour is padded with a null IPDU to match the length of the longer subtour) as

$$dt(A, B) = \sum_{i=1}^{n} p(i) \times r(\delta_i(A, B))$$

where the functions p and r are user configurable and must satisfy the following properties

- $\{p(i)\}_{i=1}^{\infty}$ is a sequence of positive numbers such that $\sum_{i=1}^{\infty} p(i)$ converges

- $\{r(i)\}_{i=0}^{\infty}$ is a nondecreasing sequence in $[0,1]$ such that $r(0) = 0$ and $\lim_{i\to\infty} r(i) = 1$. Furthermore, the sequence $\{\frac{r(i)}{i}\}_{i=1}^{\infty}$ is non-increasing.
- $$\delta_i(A,B) = \begin{cases} 0 & \text{if } a_i = b_i \text{ and } \alpha_i = \beta_i \\ |\alpha_i - \beta_i| & \text{if } a_i = b_i \text{ and } \alpha_i \neq \beta_i \\ \infty & \text{if } a_i \neq b_i \end{cases}$$

The metric space defined by dt on the set of all subtours starting at the initial protocol state is both complete and convergent (i.e. every Cauchy sequence converges) [Vuo-91].

3. We use a coverage measure for a set of test cases C relative to a test suite (set of subtours) TS from the metric distance dt as follows:

   Let $d(x,S) = \inf\{dt(x,y) | y \in S\}$

   The normalized maximum distance from C to TS is

   $$m(C) = \frac{\sup\{d(x,C) | x \in TS \backslash C\}}{\sum_{i=1}^{\infty} p(i)}$$

   and the coverage of C with respect to TS is defined as $1 - m(C)$.

4. The following function definitions for p, r, and δ are used to compute the testing distance between two subtours:

   $$p(i) = \frac{1}{2^i}$$

   $$r(k) = \frac{k}{k+1}$$

   $$\delta_i(A,B) = \begin{cases} 0 & \text{if } a_i = b_i \text{ and } \alpha_i = \beta_i \\ |\alpha_i - \beta_i| & \text{if } a_i = b_i \text{ and } \alpha_i \neq \beta_i \\ \infty & \text{if } a_i \neq b_i \end{cases}$$

Now the task is to generate a set of selected test cases C which are far enough from each other from the test suite (of subtours) TS generated by TESTGEN.

We use the greedy algorithm for test case selection. Let ϵ be an accuracy measurement (test case radius) where each subtour in TS must not be more that ϵ units from some test case. Since all Cauchy sequences converge in the metric space and repeated application of the algorithm with successively narrower test case radii produces Cauchy sequences, the greedy approach yields a set of test cases with general coverage of the test suite and guarantees convergence to the test suite.

The algorithm begins with an empty set of test cases. For each subtour in the test suite, we compute its minimum distance to the sequences already present in the set of test cases. The control sequence is added to our set of test cases if this minimum distance is greater than the test case radius ϵ. Once all the subtours in the test suite have been examined we shrink the test case radius ϵ and re-iterate the algorithm without emptying the set of test cases. The re-iteration continues until we reach some minimum test case radius ϵ_{min} or the number of selected test cases exceeds a user-defined maximum value. The detailed algorithm can be found in [Vuo-91] and [McA-92].

The set of selected test cases can be found in appendix E. There are 14 of them.

4.4 Summary

- The test cases generated by TESTGEN and coverage tool can be applied against the prototype for simplified Z39.50 without being modified, because the data part can be retained in this phase and emphasis is placed on the control part of the machine.
- For later phases, these cases can be modified manually to include the actual data.

In this specific scenario of Z39.50, I used TESTGEN cases in appendix D and the selected test cases in appendix E (concerned purely with the control part) to generate the test cases which can be applied against the actual implementation. The number of these test cases is about 50 and some of them are found in appendix I.

Chapter 5

FSM-BASED APPROACH TO PROTOCOL CODING

The finite state machine (FSM) notation was mentioned in several places in previous chapters. Almost all protocols, specifically communication or client/server protocols are specified using FSM models.

In this chapter, I present a practical FSM-based method for protocol coding.

5.1 Motivation

There are several reasons why we chose to apply a FSM approach in the coding of the Z39.50:

- State machine analysis allows programmers to simplify their code by getting rid of a lot of very complicated and spaghetti-coded loops.
- All services of the protocol can be included in the table at once with their corresponding functions empty and as the implementation goes on, these functions will be added or modified accordingly.
- The protocol specification defines a simple and very generic state table, which we can readily make use of.
- Recent Z39.50 standards include the state tables which are in fact another form of FSM. Even though they do not constitute a formal definition of the standard, they provide a more concise specification of the protocol procedures. In fact, looking

at the new tables one can determine if his/her implementation should add/remove any operation.

5.2 State tables

To create the FSM for the implementation, first we should write down the state table for the system we want to implement. In the case of Z39.50, we have state tables for both Origin and Target.

Each state table shows the relationship between the state of an operation, the incoming events that occur in the protocol, the actions taken, and finally, the resulting state.

For the state tables in Figures 5.8, 5.9, 5.10 we use the following convention: the intersection of an incoming event (row) and a state (column) forms a cell. A non-blank cell represents an incoming event and state that is defined by the standard. Such cell should contain an action and a transition to a resulting state, in parentheses. A blank cell represents the combination of an incoming event and a state that is not specified in the standard. It is up to the implementors to treat those cells. For example, according to Figure 5.8, in case an Init request PDU is coming from the Origin when the Target is in "Closed" state, the implementors must execute the "Init-Indication" action and then transfer the protocol to a new state, "Init-Received", but standard does not tell the implementors to do anything if Init request PDU is coming when the Target in another state, say Idle.

Here are the definitions of the states in the tables:

1. CLOSED: awaiting an Init PDU from the origin.
2. INIT-RECVD: awaiting an Init Response from the service-user.

3. ACC-SENT: during initialization (in 5.8) or search operation (in 5.9) or present operation (in 5.10) the target has sent an Access-Control PDU and is awaiting an Access Control response from the origin.
4. RSC-SENT: during initialization (in 5.8) or search operation (in 5.9) or present operation (in 5.10) the target has sent an Resource-Control PDU and is awaiting an Resource-Control response from the origin.
5. IDLE: the connection is established, awaiting for a request from the origin.
6. ACTIVE: a request is received (search, present etc) and target is processing it.
7. Close-Sent: awaiting a Close PDU from the origin.
8. Close-Recvd: awaiting a Close response from the service-user
9. Present RECVD: present operation is processed.
10. <OP> RECVD: an operation other than Present is processed.

Here are the definitions of the events in the tables:

1. Init-PDU: Init request PDU is coming from the origin.
2. Init-Resp+: the service-user issues Init Response (accept)
3. Init-Resp-: the service-user issues Init Response (reject)
4. Acc-REQ: the service-user issues Access Control request
5. AccResp PDU: Access Control response is coming from the origin.
6. Rsc-REQ: the service-user issues Resource Control request
7. RscResp PDU: Resource Control response is coming from the origin.
8. Op-PDU: Any operation request (search, present, ...) is coming from origin

STATE / EVENT	CLOSED (0)	INIT-RECVD (1)	ACC-SENT (2)	RSC-SENT (3)	IDLE (4)	ACTIVE (5)	Close-Sent (6)	Close-Recvd (7)
Init-PDU	Init-Ind; (1)							
Init-RESP+		InitRespPDU+; (4)						
Init-RESP-		InitRespPDU-; (0)						
Acc-REQ		Acc-PDU; (2)						
AccRespPDU			Acc-Conf; (1)					
Rsc-REQ		RscPDU; (3)						
RcsRespPDU				Rsc-Conf (1)				
Op-PDU					Initiate OP (5)			
Close-REQ					Close-PDU (6)	Close-PDU Kill OP (6)		
Close-PDU					Close-Ind (7)	Close-Ind Kill OP (7)		
CloseResp								Close-PDU (0)
Close-PDU							Close-Conf (0)	

Figure 5.8: The state table for Z39.50 Target main processing

9. Close-REQ: the service-user issues Close request

10. Close-PDU: Close PDU is coming from origin.

11. CloseResp: the service-user issues Close in response to Close PDU from origin.

Tables can be represented in many ways, but we choose the matrix forms, as our programs can make use of it in finding the cells quickly.

We will use the Target for illustrations, as the Origin will be similar. Figures 5.8, 5.9, 5.10 contain the tables describing the main processing phase and the search and retrieval phase for a Z39.50 Target [IR-Z3950-95].

EVENT \ STATE	<OP> RECVD	Rsc Sent	Acc Sent
Rsc-REQ	Rsc PDU (2)		
Rsc Resp PDU		Rsc Conf (1)	
Acc-REQ	Acc PDU (3)		
Acc Resp PDU			Acc Conf (1)
Trigrc PDU	Trigrc Ind (1)		
<OP> Resp	<OP> Resp PDU; EndOP Ind; Exit		

Figure 5.9: The state table for Z39.50 Target search processing

5.3 How to create a finite state machine

By the definition of a FSM, at any given time, the protocol can be in only one defined state. From that state, if a corresponding event occurs, then some specific task should be performed and the protocol will likely (but not always) move to one of the next states. As an example, after getting "Init-Request" from Origin, the Target will record the fact and enters state "receive an Init-Request". Then, it will process the Request and eventually sends "an Init-Response" to the Client and moves to state "IDLE" waiting for another request. Processing requests such as Search and Present means that program should take care of behavior of another subtable.

The move from a state to one of its next states is called a *transition*.

Nearly all functions processing the tasks will be empty at the beginning and will be filled out gradually later as needed.

EVENT \ STATE	Present RECVD	Rsc Sent	Acc Sent
Rsc-REQ	Rsc PDU (2)		
Rsc Resp PDU		Rsc Conf (1)	
Acc-REQ	Acc PDU (3)		
Acc Resp PDU			Acc Conf (1)
Trigrc PDU	Trigrc Ind (1)		
Segment REQ	Segment PDU (1)		
Present Resp	Present Resp PDU; EndOP Ind; Exit		

Figure 5.10: The state table for Z39.50 Target retrieval (present) processing

On the Target system, we define one main state table to deal with the functionality of the whole server and several subtables to deal with the corresponding operations such as Search, Present and so on. This way the main program does not have to be rewritten but only new subtable and the functions dealing with this subtable must be created whenever another operation is added to the implementation.

In each state table (in fact an array of structures), each element contains the following items:

1. State determining the current state.
2. Event determining the coming event.
3. Subtable (if not NULL) determining the address of the subtable the protocol should

go in and execute the particular operation.

4. To_do_Function determining the actions to be taken in this case (or after returning from the Subtable).
5. Parameters1 determining the parameters for the above function.
6. Next State determining the next state the protocol will be in after executing the above function.
7. Event_Generate_Function (if not NULL) determining the function generating the next appropriate event other than the PDUs coming from Client side.
8. Parameters2 determining the parameters for the Event_Generate_Function.

To speed up the search of the appropriate element, we use a 2-dimension matrix called Transition_Matrix which for each State and Event will yield the position of the element in the state table. Therefore beside the above items, a state table also has another item called Transition_Number, enumerating the current element, and we just need to increase the address of the table with value from Transition_Matrix to get to the right element.

The main state table, subtables for Search, Present can be found in appendix G.

The function Process_StateMachine is much smaller and can be used to process any similar table in other programs or even in different places in the program (e.g. processing subtables). It is found in appendix H.

5.4 Summary

Applying the FSM approach in actual coding proves to be a very good point. The main advantages are:

- The huge project can be divided into small individual tasks which then can be implemented, tested independently by different members. Some of the tasks can even be left undone and added later without affecting the whole system.
- The protocol is easier to understand. By studying the FSM and the related state tables, one can see which functions or elements should be implemented.
- The logic of the program at earlier stage and specially in prototyping phase can be tested by manually running each case through the table even before the real coding. In fact even customers with non-programming background can look at the table and give feedback to software team members.
- The program codes, namely the control part, are extremely clearer and thus much easier to modify. Since FSM approach is introduced and used in our coding, for myself the controlling part of the protocol becomes quite easy to maintain (I was responsible for that part). Looking at some older versions (version 2) without FSM plugged in I had to spend so much time understanding the too many IF-ELSE and/or SWITCH-CASE statements that at the end I did not know where and how to find the main operational functions. With FSM, it took me only half a day to plug a new service and everything works very smoothly.

Chapter 6

SOME EXPERIENCE AND REMARKS

This thesis presents the application of a formal conformance testing method and the TESTGEN tool to generate a test suite for the Search and Retrieval Information Protocol Z39.50.

Here are some of my experience and remarks about the tool.

6.0.1 Advantages of TESTGEN

TESTGEN proves to be useful in generating test cases as a skeleton or initial test suite.

1. First of all, using TESTGEN requires the implementor to specify the English written specification in a formal specification language (Estelle.Y in this case). Thus many ambiguities can be found prior to the development phase.

2. As the test cases can be generated automatically, they can be created quickly if there are changes in the English specifications. We would have to modify Estelle.Y or ASN.1 specs, but these changes are closer to the English spec and thus can be more straight forward. Moreover, ASN.1 is usually presented together with the modifications.

3. Passing test cases from TESTGEN increases the possibility of being inter-operable with other implementations. In fact, if TESTGEN can be improved in several points (see chapter 7), it can be used to generate a set of standard conformance

test cases in TTCN. Such a set for Z39.50 protocol is not available for the time being. For now, to ensure interoperability in the Z39.50 world, there have been three main kinds of efforts:

- Testbeds: From time to time, some of the ZIG members have been developing implementations that other organizations can check their software against. But as implementing takes time you can rarely get the right testbeds for your current implementation.
- Bilateral interoperability: The large vendors routinely make deals to test each other's software. For example, this is how my previous employer wants to make sure that their system can talk to another company's system. A few vendors are more generous and let anyone, not just privileged partners, run tests against their servers. But testing against one system does not mean that your system is interoperable with another (even if tested against the same system).
- Conformance definitions outside the standard: Because it is so cumbersome to amend standards once they are in place, it was initially suggested to write the conformance levels outside of the standard itself. However, it is not practical as new services may be added to the standard and even some services considered non conformant in the past may become conformant in the future (e.g. EXPLAIN service). With TESTGEN, the implementor can be relieved from tedious generating new test cases as new services are added as mentioned above.

4. Test cases generated by TESTGEN can also be specified in TTCN form. For example, using another tool [Zha-95], we may have the test cases in machine-readable TTCN form. Some of them are in appendix F for reference.

As mentioned before, TESTGEN can help in generating a skeleton of test cases where the emphasis is placed on the control part and which include simple data part. The more complex types of data may be added manually later.

For the specific scenario of Z39.50 in this thesis, I used TESTGEN cases in appendix D and the selected test cases in appendix E (concerned purely with the control part) to generate the test cases which can be applied against the actual implementation. The number of these test cases is about 50 and some of them are found in appendix I.

6.1 Disadvantages of TESTGEN

TESTGEN has also its disadvantages and needs some more improvements so that it can be applied fully to a protocol.

1. TESTGEN supports only one single Estelle module and it is not practical for the cases when multiple entities are required to be present. For example, my test cases are generated just for single Origin with a simulated Target built inside, and therefore these test cases are used only for that Origin's behavior and they contain little information about the corresponding real Target. It is one of the reasons why such test cases can be used only in my prototyping.

2. The data part of test cases is not dealt with coherently. As TESTGEN does not execute some Estelle assign statement, data values are not passed from one layer to another layer, I must comment out some of the values so that the test cases make sense.

3. The "provided" statement is not executed in some cases, thus making the decision not applicable. To overcome that I had, for example, to break one possible PDU into another two PDUs with each having different flag. For instance, the INIT response PDU was broken down into two PDUs INIT accept-response and INIT reject-response. This bug is quite annoying and should be fixed in the next release.

4. Similarly as there is no way to control data values coming from other modules (single module problem), I also was forced to remove the conditional statements (e.g. checking the sequence numbers of a Search request and Search Response). These test cases then are useful to the cases when the control part is concerned rather than the data part.

5. TESTGEN supports a too small set of data types of ASN.1 and thus it is very difficult to embed even the most basic data components in PDUs. In fact I had to modify the data in PDUs which then do not reflect reality.

6. TESTGEN does not have the capability of retaining the recent constraints made to a PDS before the source files are reloaded (possibly with changes). Therefore it is very painful to go and impose the constraints every time a small change is made in the Estelle.Y or ASN.1 file. I suggested that recent changes should be kept somewhere in memory or external storage (file on hard-disk) so that they can be imported again if necessary according to the user.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary of important results

The thesis tried to apply TESTGEN to generate conformance test suite for a real-life protocol (Search and retrieval Z39.50 protocol).

Analyzing the results and the TESTGEN tool, the thesis also discussed its advantages and its limitations.

Due to the time constraints and the delay in formal permission, the test cases cannot be tried against a real prototype implementation. Besides, there is no standard set of conformance tests for Z39.50 available, so it is difficult to judge the quality of our test cases. However, the cases cover almost all situations from the control point of view and I am positive that they can become a backbone for a more complete test suite enhanced with more complex data parts.

A small but useful practical technique using FSM approach in actual coding is also presented and it can be used in other applications.

7.2 Future work

The following tasks could be done in the future:

- Apply the test cases against an implementation. It is the best way to measure the effectiveness of TESTGEN and to tell how far the test cases are from the traditional

test cases generated in adhoc manner.

- Specify the whole protocol in a formal description language, such as Estelle [Bud-87]. With this specification, a semiautomatic implementation in C can be generated [Vuo-88]. This is very useful as we can have at least one implementation to test against the test cases.
- Improve the TESTGEN tool in the following areas:
  1. Multi-module treatment. As mentioned in previous chapters, this test suite can be applied to a prototyping implementation (e.g. Origin with simulated Target inside). In reality, the real test will include both systems (Origin and Target) at the same time and TESTGEN currently does not support this scenario.
  2. Add more ASN.1 data types. Even though TESTGEN allows data variation via its Constraint Editor, the data types currently supported are too primitive and thus more complex data structures cannot be represented. For Z39.50, data part is very essential as it deals with information search and retrieval.
  3. Allowing constraints to be retained for later use and/or after slightly modification in Estelle.Y or ASN.1 specification.

Bibliography

[Bud-87] S. Budkowski, P. Dembinski, *An introdution to Estelle: a specification language for distributed systems,* Computer Networks and ISDN Systems, 14 (1), pp. 3-23, North Holland, Amsterdam, 1987.

[Cha-89] W.Y.L. Chan, S. T. Vuong, and M. R. Ito, *An Improved Protocol Test Generation Procedure Based on UIOs* Proceedings of the ACM SIGCOMM'89 Symposium on Communication Architectures and Protocols, September 1989.

[Cho-78] T. S. Chow, *Testing Software Design Modeled by Finite State Machines,* IEEE Transactions on Computer, Vol. 4, No. 3, pp. 178-187, March 1978.

[Gon-70] G. Gonenc, *A Method for the Design of Fault Detection Experiments,* IEEE Transactions on Computers, Vol. 19, No. 6, June 1970.

[Hsi-71] E. P. Hsieh, *Checking Experiment for Sequential Machine,* IEEE Transactions on Computers, Vol. C-20, No.10, October 1971.

[ISO-8807] Information Processing System - Open System Interconnection - *LOTOS -A Formal Description Technique Based on the Temporal Ordering of Observational Behavior,* ISO 8807, September 1987.

[ISO-8824] Information Processing System - Open System Interconnection - *Specification of Abstract Syntax Notation One,* ISO 8824, 1987.

[ISO-9074] Information Processing System - Open System Interconnection - *Estelle - A Formal Description Technique Based on an Extended State Transition Model,* IS 9074, 1989.

[IR-Z3950-95] *Information Retrieval: Application Service Definition and Protocol Specification,* Preliminary Final Text, April 1995.

[ISO-9646] *Information Technology - OSI Conformance Testing Methodology and Framework,* Draft International Standard, ISO/IEC DIS 9646 (5 Parts).

[ISO-TTCN] *Information Technology - OSI Conformance Testing Methodology and Framework,* Part 3:The Tree and Tabular Combined Notation, ISO/IEC DIS 9646-3, 1990.

[Jam-95] James J. Michael and Mark Hinnebusch *From A to Z39.50. A Networking Primer,* Mecklermedia, Westport, London 1995.

[Kel-76] R. M. Keller, *Formal Verification of Parallel programs,* Communication of the ACM 19, pp371-384, 1976.

[LiG-92] Lisa Stapleton, with Gary Sarasin, *Pacific Bell Gets Competitive,* An MT & T Publishing Supplement, November 1992.

[Lu-91] Ying Lu, *On TESTGEN, An Environment for Protocol Test Sequence Generation, And Its Application to the FDDI MAC Protocol,* M.Sc. Thesis, Aug. 1991, University of British Columbia, Canada.

[McA-92] M.McAllister, S.T.Vuong and J.Curgus, *Automated Test Case Selection Based on Test Coverage Metrics,* Proceeding of International Workshop on Protocol Test Systems, Montreal, Canada, September 1992.

[Nai-81] S. Naito and M. Tsunoyama, *Fault Detection for Sequential Machines by Transition Tours,* Proceedings of the 11th IEEE Fault-Tolerant Computing Symposium. pp.138-243, June 1981.

[Neu-90] G. W. Neufeld and Y. Yang, *The Design and Implementation of an ASN.1 Compiler,* IEEE Transactions on Software Engineering, Vol.16, No. 10, Oct. 1990.

[Neu-92] G. Neufeld and S.Vuong, *An Overview of ASN.1,* Journal of Computer Networks and ISDN Systems, Vol. 23, No. 5, pp. 393-418, Feb. 1992.

[Sam-90] M. Sample and G. Neufeld, *Support for ASN.1 within a Protocol Testing Environment,* The Third International Conference on Formal Description Techniques (FORTE '90), Madrid Spain, November 1990.

[Sam-93] Michael Sample, *Snacc 1.1: A High Perfomance ASN.1 to C/C++ Compiler*, University Of British Columbia, 1993.

[Sab-88] K.K. Sabnani and A. T. Dahbura, *A Protocol Test Generation Procedure,* Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, September 1988.

[Sar-87] B. Sarikaya, G. v. Bochman, and E. Cerny, *A Test Design Methodology for Protocol Testing,* IEEE Transactions on Software Engineering. Vol. 13, No. 5, pp. 518-531, May 1987.

[Sid-89] D.P. Sidhu and T. -K. Leung, *Formal Methods for Protocol Testing: A Detailed Study,* IEEE Transactions on Software Engineering, Vol. 15, No. 4, pp. 413-426, April 1989.

[Ste-90] Douglas Steedman, *Abstract Syntax Notation One: The Tutorial & Reference*, Technology Appraisals.

[Ura-87] H. Ural, *A Test Derivation Method for Protocol Conformance Testing,* University of Ottawa, Technical Report - TR-87-04, Jan. 87.

[Ura-88] H. Ural, B. Yang, R. L. Probert, *A Test Sequence Selection Method for Protocols Specified in Estelle,* Technical Report TR-88-18, Department of Computer Science, University of Ottawa, June 1988.

[Vuo-88] S.T. Vuong, A.C. Lau and R.I. Chan, *Semiautomatic Implementation of Protocols Using an Estelle-C Compiler,* IEEE Transactions on Software Engineering, Vol 14, No.3, March 1988.

[Vuo-89] S. T. Vuong, W. Y. L. Chan, and M. R. Ito, *The UIOv-Method for Protocol Test Sequence Generation,* Proceedings of the Second International Workshop on Protocol Test Systems, October 1989.

[Vuo-91] S.T.Vuong and J.Curgus, *Test Coverage Metrics for Communication Protocols,* Invited paper, Proceedings of the IWPTS IV - Int. Workshop on Protocol Test Systems, Leischendam, The Netherlands, 1991.

[Vuo-94] S T Vuong, H Janssen, Y Lu, C Mathieson and B Do, *TESTGEN: An environment For Protocol Test Suite Generation and Selection,* Computer Communications, V. 17, No. 4, April 1994.

[Zha-95] Rui Zhang *Master Thesis,* to be released in 1996.

Appendix A

Estelle.Y specification of the simplified Z39.50 used in TESTGEN

```
specification Z3950Origin;

CONST
   accept = 0: int;
   reject = 1: int;
   success = 0: int;
   failure = 1: int;
   prefMessageSize = 1024: int;
   normalCloseReason = 0: int;
   securityCloseReason = 1: int;

VAR
   counter: int;
   searchStatus: int;

ISP
   AEINITreq   User;
   AESEAHreq   User;
   AEPRESreq   User;
   AECLOSreq   User;
```

```
   MSMessage   Net;

OSP
   AEINITconf  User;
   AESEAHconf  User;
   AEPRESconf  User;
   MSMessage   Net;

PDU
   INITREQ   recv_in AEINITreq,
             sent_in MSMessage;
   INITRESPACCEPT  recv_in MSMessage,
             sent_in AEINITconf;
   INITRESPREJECT  recv_in MSMessage,
             sent_in AEINITconf;
   SEAHREQ   recv_in AESEAHreq,
             sent_in MSMessage;
   SEAHRESPSUCCESS  recv_in MSMessage,
             sent_in AESEAHconf;
   SEAHRESPFAILURE  recv_in MSMessage,
             sent_in AESEAHconf;
   PRESREQ   recv_in AEPRESreq,
             sent_in MSMessage;
   PRESRESP  recv_in MSMessage,
             sent_in AEPRESconf;
   CLOSREQ   recv_in AECLOSreq,
```

```
            sent_in MSMessage;
   INVALID  recv_in MSMessage;

TIMER
   none   0;

STATE
   CLOSED, INIT_SENT, IDLE, SEAH_SENT, PRES_SENT;

INITIALIZATION
   TO   CLOSED
      BEGIN
         counter := 0;
         searchStatus := failure;
      END;

trans
   FROM CLOSED to INIT_SENT
      when AEINITreq
         output INITREQ
            begin
               INITREQ.protocolVersion := AEINITreq.protocolVersion;
               INITREQ.options := AEINITreq.options;
               INITREQ.prefMessageSize := AEINITreq.prefMessageSize;
               INITREQ.maxRecordSize := AEINITreq.maxRecordSize;
            end;
```

```
FROM INIT_SENT to IDLE
   when INITRESPACCEPT
         output AEINITconf
            begin
               AEINITconf.protocolVersion := INITRESPACCEPT.protocolVersion;
               AEINITconf.options := INITRESPACCEPT.options;
               AEINITconf.prefMessageSize := INITRESPACCEPT.prefMessageSize;
               AEINITconf.maxRecordSize := INITRESPACCEPT.maxRecordSize;
               AEINITconf.result := accept;
               counter := counter + 1;
            end;
FROM INIT_SENT to CLOSED
   when INITRESPREJECT
         output AEINITconf
            begin
               AEINITconf.protocolVersion := INITRESPREJECT.protocolVersion;
               AEINITconf.options := INITRESPREJECT.options;
               AEINITconf.prefMessageSize := 0;
               AEINITconf.maxRecordSize := 0;
               AEINITconf.result := reject;
               counter := 0;
            end;
FROM IDLE to SEAH_SENT
   when AESEAHreq
      output SEAHREQ
         begin
```

```
            SEAHREQ.databaseName := AESEAHreq.databaseName;
            SEAHREQ.query := AESEAHreq.query;
            SEAHREQ.resultSetName := AESEAHreq.resultSetName;
         end;
FROM SEAH_SENT to IDLE
   when SEAHRESPSUCCESS
      output AESEAHconf
         begin
            AESEAHconf.resultCount := SEAHRESPSUCCESS.resultCount;
            AESEAHconf.numOfRecsReturned := SEAHRESPSUCCESS.numOfRecsReturned;
            AESEAHconf.searchStatus := success;
            counter := counter + 1;
            searchStatus := success;
         end;
FROM SEAH_SENT to IDLE
   when SEAHRESPFAILURE
      output AESEAHconf
         begin
            AESEAHconf.resultCount := SEAHRESPFAILURE.resultCount;
            AESEAHconf.numOfRecsReturned := SEAHRESPFAILURE.numOfRecsReturned;
            AESEAHconf.searchStatus := failure;
            counter := counter + 1;
            searchStatus := failure;
         end;
FROM IDLE to PRES_SENT
   when AEPRESreq
```

```
        provided (searchStatus = success)
            output PRESREQ
                begin
                    PRESREQ.resultStartPoint := AEPRESreq.resultStartPoint;
                    PRESREQ.numOfRecsReqested := AEPRESreq.numOfRecsReqested;
                end;
    FROM PRES_SENT to IDLE
        when PRESRESP
            output AEPRESconf
            begin
                AEPRESconf.numOfRecsReturned := PRESRESP.numOfRecsReturned;
                AEPRESconf.presentStatus := PRESRESP.presentStatus;
                counter := counter + 1;
            end;
    FROM IDLE to CLOSED
        when AECLOSreq
        output CLOSREQ
            begin
                CLOSREQ.closeReason := normalCloseReason;
                counter := 0;
            end;
    FROM INIT_SENT to CLOSED
        when INVALID
        output CLOSREQ
            begin
                CLOSREQ.closeReason := securityCloseReason;
```

```
            counter := 0;
         end;
FROM INIT_SENT to CLOSED
   when PRESRESP
      output CLOSREQ
         begin
            CLOSREQ.closeReason := securityCloseReason;
            counter := 0;
         end;
FROM INIT_SENT to CLOSED
   when SEAHRESPSUCCESS
      output CLOSREQ
         begin
            CLOSREQ.closeReason := securityCloseReason;
            counter := 0;
         end;
FROM IDLE to CLOSED
   when INVALID
      output CLOSREQ
         begin
            CLOSREQ.closeReason := securityCloseReason;
            counter := 0;
         end;
FROM SEAH_SENT to CLOSED
   when INVALID
      output CLOSREQ
```

```
            begin
               CLOSREQ.closeReason := securityCloseReason;
               counter := 0;
            end;
   FROM PRES_SENT to CLOSED
      when INVALID
         output CLOSREQ
            begin
               CLOSREQ.closeReason := securityCloseReason;
               counter := 0;
            end;
end.
```

Appendix B

ASN.1 specification of the simplified Z39.50 used in TESTGEN

```
Z3950Origin DEFINITIONS ::=

BEGIN

DataString ::= OCTET STRING


User ::= CHOICE
{
   AEINITreq,
   AEINITind,
   AEINITresp,
   AEINITconf,
   AESEAHreq,
   AESEAHind,
   AESEAHresp,
   AESEAHconf,
   AEPRESreq,
   AEPRESind,
   AEPRESresp,
```

```
   AEPRESconf,
   AECLOSreq,
   AECLOSind
}

AEINITreq ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,    -- fixedValue (1024)
   maxRecordSize     INTEGER     -- fixedValue (1024)
}

AEINITind ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,
   maxRecordSize     INTEGER
}

AEINITresp ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,     -- {fixedValue (1024)}
```

```
   maxRecordSize     INTEGER,     -- {fixedValue (1024)}
   result            INTEGER      -- { accept (0), reject (1) }
}

AEINITconf ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,
   maxRecordSize     INTEGER,
   result            INTEGER
}


AESEAHreq ::= SEQUENCE
{
   databaseName      DataString,
   query             DataString,
   resultSetName     DataString
}

AESEAHind ::= SEQUENCE
{
   databaseName      DataString,
   query             DataString,
   resultSetName     DataString
```

```
}

AESEAHresp ::= SEQUENCE
{
   resultCount       INTEGER,    -- { fixedValue (18) }
   numOfRecsReturned INTEGER     -- { fixedValue (5) }
}

AESEAHconf ::= SEQUENCE
{
   resultCount       INTEGER,
   numOfRecsReturned INTEGER,
   searchStatus      INTEGER
}

AEPRESreq ::= SEQUENCE
{
   resultStartPoint  INTEGER,    -- { fixedValue (1) }
   numOfRecsReqested INTEGER     -- { fixedValue (9) }
}

AEPRESind ::= SEQUENCE
{
   resultStartPoint  INTEGER,
   numOfRecsReqested INTEGER
}
```

```
AEPRESresp ::= SEQUENCE
{
   numOfRecsReturned INTEGER,
   presentStatus     INTEGER
}

AEPRESconf ::= SEQUENCE
{
   numOfRecsReturned INTEGER,
   presentStatus     INTEGER
}

AECLOSreq ::= SEQUENCE
{
   closeReason       INTEGER    -- {normalCloseReason (0)}
}

AECLOSind ::= SEQUENCE
{
   closeReason       INTEGER
}

Net ::= CHOICE
{
   MSMessage
```

```
}

MSMessage ::= SEQUENCE
{
   dummy              INTEGER
}

PduMessages ::= CHOICE
{
   INITREQ,
   INITRESPACCEPT,
   INITRESPREJECT,
   SEAHREQ,
   SEAHRESPSUCCESS,
   SEAHRESPFAILURE,
   PRESREQ,
   PRESRESP,
   CLOSREQ,
   INVALID
}

INITREQ ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,
```

```
   maxRecordSize     INTEGER
}

INITRESPACCEPT ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,     -- {fixedValue (1024)}
   maxRecordSize     INTEGER,     -- {fixedValue (1024)}
   result            INTEGER      -- {accept (0)}
}

INITRESPREJECT ::= SEQUENCE
{
   protocolVersion   DataString,
   options           DataString,
   prefMessageSize   INTEGER,     -- {fixedValue (1024)}
   maxRecordSize     INTEGER,     -- {fixedValue (1024)}
   result            INTEGER      -- {reject (1)}
}

SEAHREQ ::= SEQUENCE
{
   databaseName      DataString,
   query             DataString,
   resultSetName     DataString
```

```
}

SEAHRESPSUCCESS ::= SEQUENCE
{
   resultCount        INTEGER,   -- {fixedValue (18)}
   numOfRecsReturned INTEGER,   -- {fixedValue (9)}
   searchStatus       INTEGER    -- {success (0)}
}

SEAHRESPFAILURE ::= SEQUENCE
{
   resultCount        INTEGER,   -- {fixedValue (0)}
   numOfRecsReturned INTEGER,   -- {fixedValue (0)}
   searchStatus       INTEGER    -- {failure (1) }
}
PRESREQ ::= SEQUENCE
{
   resultStartPoint  INTEGER,
   numOfRecsReqested INTEGER,
   resultSetName     DataString
}

PRESRESP ::= SEQUENCE
{
   numOfRecsReturned INTEGER,   -- {fixedValue (9)}
   presentStatus     INTEGER    -- {alwaysOK (0)}
```

```
}

CLOSREQ ::= SEQUENCE
{
   closeReason        INTEGER
}

INVALID ::= SEQUENCE
{
   dummy              INTEGER -- {onlyZero (0)}
}



END
```

Appendix C

Constraints defined in TESTGEN menus

```
#=====================================================================#
#                 TSG Constraints Editor ( STATE )                    #
#                 ================================                    #
#                                                                     #
#                                                (a)       (b)        #
# +-----+---------------------------------------+---------+---------+ #
# | key | Name                                  | min-use | max-use | #
# +-----+---------------------------------------+---------+---------+ #
# | 0   | CLOSED                                | 0       | 1       | #
# | 1   | INIT_SENT                             | 0       | 1       | #
# | 2   | IDLE                                  | 0       | 4       | #
# | 3   | SEAH_SENT                             | 0       | 4       | #
# | 4   | PRES_SENT                             | 0       | 4       | #
# +-----+---------------------------------------+---------+---------+ #
#                                                                     #


#=====================================================================#
#                  TSG Constraints Editor ( ISP )                     #
#                  ==============================                     #
#                                                                     #
```

```
#                                                (a)       (b)        #
# +-----+-------------------------------------+---------+---------+ #
# | key | Name                                | min-use | max-use | #
# +-----+-------------------------------------+---------+---------+ #
# | 0   | AEINITreq                           | 0       | 1       | #
# | 1   | AESEAHreq                           | 0       | 4       | #
# | 2   | AEPRESreq                           | 0       | 4       | #
# | 3   | AECLOSreq                           | 0       | 1       | #
# | 4   | MSMessage                           | 0       | 1       | #
# +-----+-------------------------------------+---------+---------+ #

#=====================================================================#
#                   TSG Constraints Editor ( OSP )                    #
#                   ==============================                    #
#                                                                     #
#                                                (a)       (b)        #
# +-----+-------------------------------------+---------+---------+ #
# | key | Name                                | min-use | max-use | #
# +-----+-------------------------------------+---------+---------+ #
# | 0   | AEINITconf                          | 0       | 1       | #
# | 1   | AESEAHconf                          | 0       | 4       | #
# | 2   | AEPRESconf                          | 0       | 4       | #
# | 3   | MSMessage                           | 0       | 1       | #
# +-----+-------------------------------------+---------+---------+ #

#=====================================================================#
```

```
#                    TSG Constraints Editor ( PDU )                    #
#                    ==============================                    #
#                                                                      #
#                                                (a)       (b)         #
# +-----+-------------------------------------+---------+---------+ #
# | key | Name                                | min-use | max-use | #
# +-----+-------------------------------------+---------+---------+ #
# | 0   | INITREQ                             | 0       | 1       | #
# | 1   | INITRESPACCEPT                      | 0       | 1       | #
# | 2   | INITRESPREJECT                      | 0       | 1       | #
# | 3   | SEAHREQ                             | 0       | 4       | #
# | 4   | SEAHRESPSUCCESS                     | 0       | 4       | #
# | 5   | SEAHRESPFAILURE                     | 0       | 4       | #
# | 6   | PRESREQ                             | 0       | 4       | #
# | 7   | PRESRESP                            | 0       | 4       | #
# | 8   | CLOSREQ                             | 0       | 1       | #
# | 9   | INVALID                             | 0       | 1       | #
# +-----+-------------------------------------+---------+---------+ #


#=====================================================================#
#                Interactive Service Primitive Parameter Definition   #
#                ==================================================   #
#                                                                     #
# ISP: AEINITreq                    ( 4 parameter ) => 4    instances #
#=====================================================================#
```

```
+-------------------------------------------------------------------+
| Key Type      Parameter name                                      |
|                  values                                           |
+-------------------------------------------------------------------+
| 0   char* AEINITreq.protocolVersion                               |
|  .0       '111'                                                   |
| 1   char* AEINITreq.options                                       |
|  .0       '111111'                                                |
| 2   int   AEINITreq.prefMessageSize                               |
|  .0       1024                                                    |
|  .1       2048                                                    |
| 3   int   AEINITreq.maxRecordSize                                 |
|  .0       1024                                                    |
|  .1       2048                                                    |
+-------------------------------------------------------------------+


#===================================================================#
#               Interactive Service Primitive Parameter Definition  #
#               ==================================================  #
#                                                                   #
# ISP: AESEAHreq                    ( 3 parameter ) => 1    instances #
#===================================================================#


+-------------------------------------------------------------------+
| Key Type      Parameter name                                      |
|                  values                                           |
```

```
+-------------------------------------------------------------------+
| 4   char* AESEAHreq.databaseName                                  |
|  .0       'searchDatabase'                                        |
| 5   char* AESEAHreq.query                                         |
|  .0       'searchQuery'                                           |
| 6   char* AESEAHreq.resultSetName                                 |
|  .0       'searchResultSetName'                                   |
+-------------------------------------------------------------------+


#===================================================================#
#               Interactive Service Primitive Parameter Definition  #
#               ==================================================  #
#                                                                   #
# ISP: AEPRESreq                    ( 2 parameter ) => 1    instances #
#===================================================================#


+-------------------------------------------------------------------+
| Key Type      Parameter name                                      |
|                  values                                           |
+-------------------------------------------------------------------+
| 7   int   AEPRESreq.resultStartPoint                              |
|  .0       1                                                       |
| 8   int   AEPRESreq.numOfRecsReqested                             |
|  .0       9                                                       |
+-------------------------------------------------------------------+
```

```
#=====================================================================#
#              Interactive Service Primitive Parameter Definition     #
#              ================================================       #
#                                                                     #
# ISP: AECLOSreq                    ( 1 parameter ) => 1    instances #
#=====================================================================#


+---------------------------------------------------------------------+
| Key Type      Parameter name                                        |
|                  values                                             |
+---------------------------------------------------------------------+
| 9   int   AECLOSreq.closeReason                                     |
|  .0       0                                                         |
+---------------------------------------------------------------------+


#=====================================================================#
#          Interactive Protocol Data Unit Parameter Definition        #
#          ===================================================        #
#                                                                     #
# PDU: INITRESPACCEPT               ( 5 parameter ) => 4    instances #
#=====================================================================#


+---------------------------------------------------------------------+
| Key Type  Parameter name                                            |
|              values                                                 |
+---------------------------------------------------------------------+
```

```
|25   char* INITRESPACCEPT.protocolVersion                          |
|  .0         '111'                                                 |
|26   char* INITRESPACCEPT.options                                  |
|  .0         '111111'                                              |
|27   int   INITRESPACCEPT.prefMessageSize                          |
|  .0         1024                                                  |
|  .1         2048                                                  |
|28   int   INITRESPACCEPT.maxRecordSize                            |
|  .0         1024                                                  |
|  .1         2048                                                  |
|29   int   INITRESPACCEPT.result                                   |
|  .0         0                                                     |
+-------------------------------------------------------------------+


#===================================================================#
#           Interactive Protocol Data Unit Parameter Definition     #
#           ==================================================      #
#                                                                   #
# PDU: INITRESPREJECT              ( 5 parameter ) => 1    instances #
#===================================================================#


+-------------------------------------------------------------------+
| Key Type  Parameter name                                          |
|             values                                                |
+-------------------------------------------------------------------+
|30   char* INITRESPREJECT.protocolVersion                          |
```

```
|  .0          dummy_val1                                          |
|31   char* INITRESPREJECT.options                                 |
|  .0          dummy_val1                                          |
|32   int   INITRESPREJECT.prefMessageSize                         |
|  .0          0                                                   |
|33   int   INITRESPREJECT.maxRecordSize                           |
|  .0          0                                                   |
|34   int   INITRESPREJECT.result                                  |
|  .0          1                                                   |
+-------------------------------------------------------------------+


#===================================================================#
#           Interactive Protocol Data Unit Parameter Definition     #
#           ==================================================      #
#                                                                   #
# PDU: SEAHRESPSUCCESS              ( 3 parameter ) => 2    instances #
#===================================================================#


+-------------------------------------------------------------------+
| Key Type  Parameter name                                          |
|              values                                               |
+-------------------------------------------------------------------+
|38   int   SEAHRESPSUCCESS.resultCount                            |
|  .0          9                                                   |
|  .1          18                                                  |
|39   int   SEAHRESPSUCCESS.numOfRecsReturned                      |
```

```
|  .0          5                                                    |
|40   int   SEAHRESPSUCCESS.searchStatus                            |
|  .0          0                                                    |
+-------------------------------------------------------------------+


#===================================================================#
#           Interactive Protocol Data Unit Parameter Definition     #
#           ===================================================     #
#                                                                   #
# PDU: SEAHRESPFAILURE              ( 3 parameter ) => 1    instances #
#===================================================================#


+-------------------------------------------------------------------+
| Key Type  Parameter name                                          |
|              values                                               |
+-------------------------------------------------------------------+
|41   int   SEAHRESPFAILURE.resultCount                             |
|  .0          0                                                    |
|42   int   SEAHRESPFAILURE.numOfRecsReturned                       |
|  .0          0                                                    |
|43   int   SEAHRESPFAILURE.searchStatus                            |
|  .0          1                                                    |
+-------------------------------------------------------------------+


#===================================================================#
#           Interactive Protocol Data Unit Parameter Definition     #
```

```
#           ==================================================        #
#                                                                     #
# PDU: PRESRESP                      ( 2 parameter ) => 2    instances #
#=====================================================================#


+---------------------------------------------------------------------+
| Key Type  Parameter name                                            |
|              values                                                 |
+---------------------------------------------------------------------+
|47   int   PRESRESP.numOfRecsReturned                                |
|  .0          9                                                      |
|48   int   PRESRESP.presentStatus                                    |
|  .0          0                                                      |
|  .1          1                                                      |
+---------------------------------------------------------------------+


#=====================================================================#
#           Interactive Protocol Data Unit Parameter Definition       #
#           ==================================================        #
#                                                                     #
# PDU: INVALID                       ( 1 parameter ) => 1    instances #
#=====================================================================#


+---------------------------------------------------------------------+
| Key Type  Parameter name                                            |
|              values                                                 |
```

```
+-----------------------------------------------------------------------+
|50   int   INVALID.dummy                                               |
|  .0          0                                                        |
+-----------------------------------------------------------------------+
```

Appendix D

Some test cases with defined constraints without coverage

```
                                  +--------+
                                  | CLOSED |
                                  +--------+
                                       |
ISP: AEINITreq                         |OSP :---
     { protocolVersion = '111'         |
       options = '111111'              |
       prefMessageSize = 1024          |
       maxRecordSize = 1024            |
     }                                 |
                                       |
IPDU:---                               |OPDU1:INITREQ
                                       |     { protocolVersion = '111'
                                       |       options = '111111'
                                       |       prefMessageSize = 1024
                                       |       maxRecordSize = 1024
                                       |     }
                                       |
                                       |OPDU2:---
                                       |
```

```
                                  +-----V-----+
                                  | INIT_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AEINITconf
                                        |     { protocolVersion = '111'
                                        |       options = '111111'
                                        |       prefMessageSize = 1024
                                        |       maxRecordSize = 1024
                                        |       result = 0
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:INITRESPACCEPT                     |OPDU :---
     { protocolVersion = '111'          |
       options = '111111'               |
       prefMessageSize = 1024           |
       maxRecordSize = 1024             |
       result = 0                       |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
```

```
ISP: AESEAHreq                          |OSP :---
     { databaseName = 'searchDatabase' |
       query = 'searchQuery'            |
       resultSetName                    |
         = 'searchResultSetName'        |
     }                                  |
                                        |
IPDU:---                                |OPDU1:SEAHREQ
                                        |     { databaseName = 'searchDatabase'
                                        |       query = 'searchQuery'
                                        |       resultSetName
                                        |         = 'searchResultSetName'
                                        |     }
                                        |
                                        |OPDU2:---
                                        |
                                  +-----V-----+
                                  | SEAH_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AESEAHconf
                                        |     { resultCount = 9
                                        |       numOfRecsReturned = 5
                                        |       searchStatus = 0
                                        |     }
                                        |
```

```
                                        |OSP2:---
                                        |
IPDU:SEAHRESPSUCCESS                    |OPDU :---
     { resultCount = 9                  |
       numOfRecsReturned = 5            |
       searchStatus = 0                 |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
ISP: AEPRESreq                          |OSP :---
     { resultStartPoint = 1             |
       numOfRecsReqested = 9            |
     }                                  |
                                        |
IPDU:---                                |OPDU1:PRESREQ
                                        |     { resultStartPoint = 1
                                        |       numOfRecsReqested = 9
                                        |       resultSetName = def_string
                                        |     }
                                        |
                                        |OPDU2:---
                                        |
                                  +-----V-----+
```

```
                                  | PRES_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AEPRESconf
                                        |     { numOfRecsReturned = 9
                                        |       presentStatus = 0
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:PRESRESP                           |OPDU :---
     { numOfRecsReturned = 9            |
       presentStatus = 0                |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
ISP: AECLOSreq                          |OSP :---
     { closeReason = 0                  |
     }                                  |
                                        |
IPDU:---                                |OPDU1:CLOSREQ
                                        |     { closeReason = 0
                                        |     }
```

```
      |
      |OPDU2:---
      |
+----V---+
| CLOSED |
+--------+
```

Appendix E

The set of selected test cases using test coverage tool

```
coverage is 1.000000
coverage guarantee is 0.999023
effective radius is 0.000000
number of test cases is 14
cost of the test cases is 14

Pass 1, tolerance 2.000000
Overall cost is now 0

Pass 2, tolerance 1.000000
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (AEPRESreq, 1) (PRESRESP, 1) (AECLOSreq, 1)
Overall cost is now 1

Pass 3, tolerance 0.500000
1/ (AEINITreq, 1) (INITRESPREJECT, 1)
Overall cost is now 2

Pass 4, tolerance 0.250000
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AECLOSreq, 1)
```

```
2/ (AEINITreq, 1) (INVALID, 1)
3/ (AEINITreq, 1) (PRESRESP, 1)
4/ (AEINITreq, 1) (SEAHRESPSUCCESS, 1)
Overall cost is now 6

Pass 5, tolerance 0.125000
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPFAILURE, 1)
   (AECLOSreq, 1)
2/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (INVALID, 1)
3/ (AEINITreq, 1) (INITRESPACCEPT, 1) (INVALID, 1)
Overall cost is now 9

Pass 6, tolerance 0.062500
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (AECLOSreq, 1)
Overall cost is now 10

Pass 7, tolerance 0.031250
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (AEPRESreq, 1) (INVALID, 1)
2/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (INVALID, 1)
3/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPFAILURE, 1)
   (INVALID, 1)
Overall cost is now 13
```

```
Pass 8, tolerance 0.015625
Overall cost is now 13


Pass 9, tolerance 0.007812
1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (AEPRESreq, 1) (PRESRESP, 1) (INVALID, 1)

Overall cost is now 14

Pass 10, tolerance 0.003906
Overall cost is now 14

Pass 11, tolerance 0.001953
Overall cost is now 14


And the test cases are:

1/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1)
   (AEPRESreq, 1) (PRESRESP, 1) (AECLOSreq, 1)
2/ (AEINITreq, 1) (INITRESPREJECT, 1)
3/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AECLOSreq, 1)
4/ (AEINITreq, 1) (INVALID, 1)
5/ (AEINITreq, 1) (PRESRESP, 1)
6/ (AEINITreq, 1) (SEAHRESPSUCCESS, 1)
```

7/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPFAILURE, 1) (AECLOSreq, 1)

8/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (INVALID, 1)

9/ (AEINITreq, 1) (INITRESPACCEPT, 1) (INVALID, 1)

10/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1) (AECLOSreq, 1)

11/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1) (AEPRESreq, 1) (INVALID, 1)

12/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1) (INVALID, 1)

13/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPFAILURE, 1) (INVALID, 1)

14/ (AEINITreq, 1) (INITRESPACCEPT, 1) (AESEAHreq, 1) (SEAHRESPSUCCESS, 1) (AEPRESreq, 1) (PRESRESP, 1) (INVALID, 1)

Appendix F

Test cases in TTCN form

```
$Suite
  $SuiteId   suite0001
  $DeclarationsPart
      $Begin_TS_ConstDcls
       $TS_ConstDcl
        $TS_ConstId  prefMessageSize
        $TS_ConstType  INTEGER
        $TS_ConstValue  1024
       $End_TS_ConstDcl
      $End_TS_ConstDcls
      $Begin_TS_VarDcls
       $TS_VarDcl
        $TS_VarId  counter
        $TS_VarType  INTEGER
        $TS_VarValue  0
       $End_TS_VarDcl
      $End_TS_VarDcls
    $Begin_PCO_Dcls
      $PCO_Dcl
        $PCO_Id  L
```

```
      $PCO_TypeId User
    $End_PCO_Dcl
    $PCO_Dcl
      $PCO_Id  L
      $PCO_TypeId Net
    $End_PCO_Dcl
    $PCO_Dcl
      $PCO_Id  L
      $PCO_TypeId USer
    $End_PCO_Dcl
  $End_PCO_Dcls
  $ASP_TypeDefs
    $ASN1_ASP_TypeDefs
        $Begin_ASN1_ASP_TypeDef
          $ASP_Id  AEINITreq
          $PCO_TypeId  User
          $Comment
          $ASN1_TypeDefinition
            SEQUENCE
            {
            protocolVersion  OCTETSTRING,
            options  OCTETSTRING,
            prefMessageSize  INTEGER,
            maxRecordSize  INTEGER
            }
          $End_ASN1_TypeDefinition
```

```
$End_ASN1_ASP_TypeDef
$Begin_ASN1_ASP_TypeDef
  $ASP_Id  AESEAHreq
  $PCO_TypeId  User
  $Comment
  $ASN1_TypeDefinition
    SEQUENCE
    {
    databaseName  OCTETSTRING,
    query  OCTETSTRING,
    resultSetName  OCTETSTRING
    }
  $End_ASN1_TypeDefinition
$End_ASN1_ASP_TypeDef
$Begin_ASN1_ASP_TypeDef
  $ASP_Id  AEPRESreq
  $PCO_TypeId  User
  $Comment
  $ASN1_TypeDefinition
    SEQUENCE
    {
    resultStartPoint  INTEGER,
    numOfRecsReqested  INTEGER
    }
  $End_ASN1_TypeDefinition
$End_ASN1_ASP_TypeDef
```

```
$Begin_ASN1_ASP_TypeDef
  $ASP_Id  AECLOSreq
  $PCO_TypeId  User
  $Comment
  $ASN1_TypeDefinition
    SEQUENCE
    {
    closeReason  INTEGER
    }
  $End_ASN1_TypeDefinition
$End_ASN1_ASP_TypeDef
$Begin_ASN1_ASP_TypeDef
  $ASP_Id  AEINITresp
  $PCO_TypeId  User
  $Comment
  $ASN1_TypeDefinition
```

....

Appendix G

State tables used for Z3950 protocol

```
/* definition of the state tables */
/*
The items on one line of each table (from left to right) are:

    int transition_name ... the enumerated name of the transition.
                            It must be present in the transition matrix.
    int State           ... the current state of the FSM
    int Event           ... the coming event
    STATE_TABLE Subtable... the address of the subtable, or NULL if none
    FUNC        Todo()  ... function executing actions in this situation
                            This function is integer-type and it must
                            return either SUCCESS (0) or non-zero value.
                            Non-zero value will cause Target to stop.
    int         Todo-Parm.. the parameter for the function Todo()
    int         Next-State. the next state of the machine after Todo()
    FUNC        Event_Maker() if not NULL, will generate new Event
                              (other from Origin's PDUs)
    int         E-M-Parm... the parameter for this Event-Maker()

Notes:
```

```
        - if the next state is EXIT (subtable), then the next table will be
        the main table, and the current state will be restored from the
        previous one. The coming event will be END_OP_IND.
        - in the subtables, the first line do not correspond to the spec. This
        line only serves as the entry state and event for the subtable.
*/

/* the main table */
STATE_TABLE MAINTable[] =
{
/* CLOSED state */
ININD, CLOSED, INIT_PDU, NULL, InitInd, 0,     INITRECVD, DoInit, INIT_PDU,
ERR10, CLOSED, ANYELSE,  NULL, Error,   CLOSED,CLOSED,    Kill_OP,CLOSED,

/* INITRECVDstate */
ACEPT,INITRECVD, RESP_ACCEPT,NULL, InitResp,ACCEPT,    IDLE,     NULL,   0,
REJCT,INITRECVD, RESP_REJECT,NULL, InitResp,REJECT,    STOPFSM,  NULL,   0,
SENAC,INITRECVD, ACC_REQ,    NULL, InitResp,ACCREQ,    ACC_SENT, NULL,   0,
SENRC,INITRECVD, RSC_REQ,    NULL, InitResp,RESREQ,    RSC_SENT, NULL,   0,
ERR20,INITRECVD, ANYELSE,    NULL, Error,   INITRECVD, INITRECVD,Kill_OP,INITRECVD,

/* Init's ACC_SENT state */
ACCNF, ACC_SENT, ACC_PDU, NULL, ACC_Conf,0,       INITRECVD,DoInit,  ACC_PDU,
ERR30, ACC_SENT, ANYELSE, NULL, Error,   ACC_SENT,ACC_SENT, Kill_OP, ACC_SENT,

/* Init' RSC_SENT state */
```

```
RSCNF, RSC_SENT, RESC_PDU,NULL,RESC_Conf,INIT_OP, INITRECVD,DoInit, RESC_PDU,
ERR40, RSC_SENT, ANYELSE, NULL,Error,    RSC_SENT,RSC_SENT, Kill_OP,RSC_SENT,

/* IDLE state */
DOSEA, IDLE, SEAH_PDU, SEAHTable, NULL,    0,   ACTIVE,   DoOP,        SEAH_PDU,
DOPRE, IDLE, PRES_PDU, PRESTable, NULL,    0,   ACTIVE,   DoOP,        PRES_PDU,
CLOIN, IDLE, CLOS_PDU, NULL,      CloseInd,0,   CLOSRECVD,DoCLOS,      IDLE,
CLO45, IDLE, CLOS_REQ, NULL,      AbClose, IDLE,CLOSSENT, FakeCLOS_PDU,IDLE,
ERR50, IDLE, ANYELSE,  NULL,      Error,   IDLE,IDLE,     Kill_OP,     IDLE,

/* ACTIVE state */
ENDIN, ACTIVE,END_OP_IND,NULL, OP_End_Ind, 0,     IDLE,    NULL,         0,
CLO52, ACTIVE,CLOS_RESP, NULL, SendClose,  ACTIVE,STOPFSM, NULL,         0,
CLO55, ACTIVE,CLOS_REQ,  NULL, AbClose,    ACTIVE,CLOSSENT,FakeCLOS_PDU, ACTIVE,
ERR60, ACTIVE,ANYELSE,   NULL, Error,      ACTIVE,ACTIVE,  Kill_OP,      ACTIVE,

/* CLOSSENT state */
CLONF, CLOSSENT, CLOS_PDU, NULL, CloseConf, 0, STOPFSM, NULL, 0,
CLO65, CLOSSENT, ANYELSE,  NULL, NULL,      0, CLOSSENT,NULL, 0,

/* CLOSRECVD state */
SENCL, CLOSRECVD,CLOS_RESP, NULL, SendClose, CLOSRECVD, STOPFSM,   NULL, 0,
CLO70, CLOSRECVD,CLOS_REQ,  NULL, AbClose,   CLOSRECVD, STOPFSM,   NULL, 0,
CLO75, CLOSRECVD,ANYELSE,   NULL, NULL,      0,         CLOSRECVD, NULL, 0,

TAB15, TABLEND
```

```
};

/* subtable for SEARCH operation */
STATE_TABLE SEAHTable[] =
{
/* The entry state and event, i.e. whenever this subtable is entered, the
   current state is OP_START, and the coming event will be OP_PDU */
OPBEG, OP_START, OP_PDU,    NULL, NULL,     0,        OP_RECVD, DoSEAH,OP_PDU,

/* The OP_RECVD state */
OPRSC, OP_RECVD, OPRSCREQ,  NULL, SendRESC, SEAH_OP, OPRSCSENT,GetRESC,0,
OPACC, OP_RECVD, OPACCREQ,  NULL, SEAHResp, OPACCREQ,OPACCSENT,NULL,   0,
OPTRG, OP_RECVD, TRIGRC_PDU,NULL, TRIG_Conf,SEAH_OP, OP_RECVD, DoSEAH, TRIGRC_PDU,
OPEND, OP_RECVD, OP_RESP,   NULL, SEAHResp, OP_RESP, EXIT,  Exit_OP,OP_RECVD,
EXI02, OP_RECVD, CLOS_PDU,  NULL, CloseInd, OP_RECVD,EXIT,  DoCLOS, OP_RECVD,
EXI05, OP_RECVD, CLOS_REQ,  NULL, NULL,     0,        EXIT,  Kill_OP,OP_RECVD,
EXI10, OP_RECVD, ANYELSE,   NULL, Error,    OP_RECVD,EXIT,  Kill_OP,OP_RECVD,

/* the OPACCSENT state */
OPACF, OPACCSENT,ACC_PDU, NULL, ACC_Conf, SEAH_OP,  OP_RECVD,DoSEAH, ACC_PDU,
EXI12, OPACCSENT,CLOS_PDU,NULL, CloseInd, OPACCSENT,EXIT,    DoCLOS, OPACCSENT,
EXI15, OPACCSENT,CLOS_REQ,NULL, NULL,     0,        EXIT,    Kill_OP,OPACCSENT,
EXI20, OPACCSENT,ANYELSE, NULL, Error,    OPACCSENT,EXIT,    Kill_OP,OPACCSENT,

/* the OPRSCSENT state */
OPRCF, OPRSCSENT,RESC_PDU,  NULL, RESC_Conf,SEAH_OP,  OP_RECVD, DoSEAH,  RESC_PDU,
```

```
EXI22, OPRSCSENT,CLOS_PDU,  NULL, CloseInd, OPRSCSENT,EXIT,     DoCLOS,  OPRSCSENT,
EXI25, OPRSCSENT,CLOS_REQ,  NULL, NULL,     0,        EXIT,     Kill_OP, OPRSCSENT,
EXI27, OPRSCSENT,TRIGRC_PDU,NULL, TRIG_Conf,SEAH_OP,  OPRSCSENT,NULL,    0,
EXI30, OPRSCSENT,ANYELSE,   NULL, Error,    OPRSCSENT,EXIT,     Kill_OP, OPRSCSENT,

TAB05, TABLEND
};

/* subtable for PRESENT operation */
STATE_TABLE _far PRESTable[] =
{
/* The entry state and event, i.e. whenever this subtable is entered, the
   current state is PR_START and the incoming event will be PR_PDU */
PRBEG, PR_START, PR_PDU,     NULL, NULL,     0,        PR_RECVD,   DoPRES, PR_PDU,

/* the PR_RECVD state */
PRRSC, PR_RECVD, PRRSCREQ,   NULL, SendRESC, PRES_OP, PRRSCSENT,NULL,                0,
PRACC, PR_RECVD, PRACCREQ,   NULL, PRESResp, PRACCREQ,PRACCSENT,NULL,                0,
PRTRG, PR_RECVD, TRIGRC_PDU, NULL, TRIG_Conf,PRES_OP, PR_RECVD,   DoPRES, TRIGRC_PDU,
PREND, PR_RECVD, PR_RESP,    NULL, PRESResp, PR_RESP, EXIT,       Exit_OP,PR_RECVD,
PRSEG, PR_RECVD, PRSEGREQ,   NULL, PRESResp, PRSEGREQ,PR_RECVD,   DoPRES, PRSEGREQ,
EXI32, PR_RECVD, CLOS_PDU,   NULL, CloseInd, PR_RECVD,EXIT,       DoCLOS, PR_RECVD,
EXI35, PR_RECVD, CLOS_REQ,   NULL, NULL,     0,        EXIT,       Kill_OP,PR_RECVD,
EXI40, PR_RECVD, ANYELSE,    NULL, Error,    PR_RECVD,EXIT,       Kill_OP,PR_RECVD,

/* the PRACCSENT state */
```

```
PRACF, PRACCSENT,ACC_PDU,  NULL, ACC_Conf, PRES_OP,   PR_RECVD,   DoPRES, ACC_PDU,
EXI42, PRACCSENT,CLOS_PDU, NULL, CloseInd, PRACCSENT, EXIT,       DoCLOS, PRACCSENT,
EXI45, PRACCSENT,CLOS_REQ, NULL, NULL,     0,         EXIT,       Kill_OP,PRACCSENT,
EXI50, PRACCSENT,ANYELSE,  NULL, Error,    PRACCSENT, EXIT,       Kill_OP,PRACCSENT,

/* the PRRSCSENT state */
PRRCF, PRRSCSENT,RESC_PDU, NULL, RESC_Conf,PRES_OP,   PR_RECVD,   DoPRES, RESC_PDU,
EXI52, PRRSCSENT,CLOS_PDU, NULL, CloseInd, PRRSCSENT, EXIT,       DoCLOS, PRRSCSENT,
EXI55, PRRSCSENT,CLOS_REQ, NULL, NULL,     0,         EXIT,       Kill_OP,PRRSCSENT,
EXI60, PRRSCSENT,ANYELSE,  NULL, Error,    PRRSCSENT, EXIT,       Kill_OP,PRRSCSENT,

TAB10, TABLEND
};
```

Appendix H

Function processing the State Machine

```
/*********************************************************************

    Function:   Process_StateMachine()

    Purpose:    this function looks at the state table and
                takes the required action. It is assumed
                that the current state and incoming event
                are stored in global "gState" and global "gEvent"
                of the global table "gTable".
    Input:      the PDU to be processed

    Ret.value:  either STOPFSM, if after this the program should
                stop, or OUTSIDE if the processing should go on
                with another PDU.

                Note: during processing, if aRet == INSIDE, the loop
                will repeated (as the event is generated on the Target
                side).
```

```
*********************************************************************/

int Process_StateMachine(PDU *thePdu)
{
    STATE_TABLE *aTable_ptr;
    int         aRet;
    int         anIndex;

Look_At_Matrix:

    if ((anIndex = gTransMatrix[gEvent][gState]) != FATAL)
    {
        /* there is a matching event/state in the table */
        aTable_ptr = gTable + anIndex;

        /* execute "todo" function if needed */
        if (aTable_ptr->todo != NULL)
        {
            aRet = (*aTable_ptr->todo)(aTable_ptr->option1);
            /* if error occurs in function todo(), then Target stops */
            if (aRet != SUCCESS)
               return (STOPFSM);
        }
```

```
/* transfer to the next state  */
gState = aTable_ptr->next_state;

/* must go to subtable or not? */
if (aTable_ptr->sub_tbl != NULL)
   gTable = aTable_ptr->sub_tbl;  /* change table */

if (gState == EXIT)  /* if exit subtable to main table */
{
    /* back to main table */
    gState = gPreviousState;
    gTable = gMainTable;
}

 /* look at the event_maker to see if should exit
    waiting to PDUs coming from Origin or not */
if (aTable_ptr->event_maker != NULL)  /* must stay inside */
{
    gEvent = (*aTable_ptr->event_maker)(thePdu, aTable_ptr->option2);
    aRet = INSIDE;
}
else
{
    aRet = OUTSIDE;
```

```
        }

    }

    else   /* A fatal error occurs */
    {
        PrintError("Fatal error, state/event not matched");
        return (STOPFSM);
    }

    if (gState == STOPFSM)        /* if state = STOPFSM, the job is done /
        return (STOPFSM);

    if (aRet == INSIDE)     /* stay inside this function  */
        goto Look_At_Matrix;

    return (aRet);
}
```

Appendix I

Test cases generated semiautomatically

```
                                  +--------+
                                  | CLOSED |
                                  +--------+
                                       |
ISP: AEINITreq                         |OSP :---
     { protocolVersion = '111'         |
       options = '111111'              |
       prefMessageSize = 1024          |
       maxRecordSize = 1024            |
     }                                 |
                                       |
IPDU:---                               |OPDU1:INITREQ
                                       |     { protocolVersion = '111'
                                       |       options = '111111'
                                       |       prefMessageSize = 1024
                                       |       maxRecordSize = 1024
                                       |     }
                                       |
                                       |OPDU2:---
                                       |
```

```
                                  +-----V-----+
                                  | INIT_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AEINITconf
                                        |     { protocolVersion = '111'
                                        |       options = '111111'
                                        |       prefMessageSize = 1024
                                        |       maxRecordSize = 1024
                                        |       result = 0
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:INITRESPACCEPT                     |OPDU :---
     { protocolVersion = '111'          |
       options = '111111'               |
       prefMessageSize = 1024           |
       maxRecordSize = 1024             |
       result = 0                       |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
```

```
ISP: AESEAHreq                          |OSP :---
     { databaseName = 'searchDatabase' |
       query = 'searchQuery'            |
       resultSetName                    |
         = 'searchResultSetName'        |
     }                                  |
                                        |
IPDU:---                                |OPDU1:SEAHREQ
                                        |     { databaseName = 'searchDatabase'
                                        |       query = 'searchQuery'
                                        |       resultSetName
                                        |         = 'searchResultSetName'
                                        |     }
                                        |
                                        |OPDU2:---
                                        |
                                  +-----V-----+
                                  | SEAH_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AESEAHconf
                                        |     { resultCount = 9
                                        |       numOfRecsReturned = 5
                                        |       searchStatus = 0
                                        |     }
                                        |
```

```
                                        |OSP2:---
                                        |
IPDU:SEAHRESPSUCCESS                    |OPDU :---
     { resultCount = 9                  |
       numOfRecsReturned = 5            |
       searchStatus = 0                 |
     }                                  |
                                        |
                                   +---V--+
                                   | IDLE |
                                   +------+
                                        |
ISP: AEPRESreq                          |OSP :---
     { resultStartPoint = 1             |
       numOfRecsReqested = 9            |
     }                                  |
                                        |
IPDU:---                                |OPDU1:PRESREQ
                                        |     { resultStartPoint = 1
                                        |       numOfRecsReqested = 9
                                        |       resultSetName = def_string
                                        |     }
                                        |
                                        |OPDU2:---
                                        |
                                  +-----V-----+
```

```
                                  | PRES_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AEPRESconf
                                        |     { numOfRecsReturned = 9
                                        |       presentStatus = 0
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:PRESRESP                           |OPDU :---
     { numOfRecsReturned = 9            |
       presentStatus = 0                |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
ISP: AECLOSreq                          |OSP :---
     { closeReason = 0                  |
     }                                  |
                                        |
IPDU:---                                |OPDU1:CLOSREQ
                                        |     { closeReason = 0
                                        |     }
```

```
                                        |
                                        |OPDU2:---
                                        |
                                   +----V---+
                                   | CLOSED |
                                   +--------+


                                   +--------+
                                   | CLOSED |
                                   +--------+
                                        |
ISP: AEINITreq                          |OSP :---
     { protocolVersion = "111"          |
       options = "111111"               |
       prefMessageSize = 1024           |
       maxRecordSize = 1024             |
     }                                  |
                                        |
IPDU:---                                |OPDU1:INITREQ
                                        |     { protocolVersion = "111"
                                        |       options = "111111"
                                        |       prefMessageSize = 1024
                                        |       maxRecordSize = 1024
                                        |     }
                                        |
                                        |OPDU2:---
```

```
                                        |
                                  +-----V-----+
                                  | INIT_SENT |
                                  +-----------+
                                        |
ISP: ---                                |OSP1:AEINITconf
                                        |     { protocolVersion = dummy_val1
                                        |       options = dummy_val1
                                        |       prefMessageSize = 0
                                        |       maxRecordSize = 0
                                        |       result = 1
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:INITRESPREJECT                     |OPDU :---
     { protocolVersion = dummy_val1     |
       options = dummy_val1             |
       prefMessageSize = 0              |
       maxRecordSize = 0                |
       result = 1                       |
     }                                  |
                                        |
                                   +----V---+
                                   | CLOSED |
                                   +--------+
```

```
                                  +--------+
                                  | CLOSED |
                                  +--------+
                                       |
ISP: AEINITreq                         |OSP :---
     { protocolVersion = '111'         |
       options = '111111'              |
       prefMessageSize = 1024          |
       maxRecordSize = 1024            |
     }                                 |
                                       |
IPDU:---                               |OPDU1:INITREQ
                                       |     { protocolVersion = '111'
                                       |       options = '111111'
                                       |       prefMessageSize = 1024
                                       |       maxRecordSize = 1024
                                       |     }
                                       |
                                       |OPDU2:---
                                       |
                                  +-----V-----+
                                  | INIT_SENT |
                                  +-----------+
                                       |
ISP: ---                               |OSP1:AEINITconf
```

```
                                        |     { protocolVersion = '111'
                                        |       options = '111111'
                                        |       prefMessageSize = 1024
                                        |       maxRecordSize = 1024
                                        |       result = 0
                                        |     }
                                        |
                                        |OSP2:---
                                        |
IPDU:INITRESPACCEPT                     |OPDU :---
     { protocolVersion = '111'          |
       options = '111111'               |
       prefMessageSize = 1024           |
       maxRecordSize = 1024             |
       result = 0                       |
     }                                  |
                                        |
                                    +---V--+
                                    | IDLE |
                                    +------+
                                        |
ISP: AECLOSreq                          |OSP :---
     { closeReason = 0                  |
     }                                  |
                                        |
IPDU:---                                |OPDU1:CLOSREQ
```

```
      |     { closeReason = 0
      |     }
      |
      |OPDU2:---
      |
+----V---+
| CLOSED |
+--------+
```