22

Design of Intelligent OSI Protocol Monitor

Tomohiko Ogishi, Akira Idoue, Toshihiko Kato and Kenji Suzuki
KDD R&D Laboratories
2-1-15, Ohara, Kamifukuoka-shi, Saitama 356, Japan
Telephone : +81-492-66-7370 Facsimile : +81-492-66-7510
E-mail : {ogishi, idoue, kato, suzuki}@hsc.lab.kdd.co.jp

Abstract

As the OSI protocols come to be widely adopted in various communication systems, the testing of OSI protocol implementations becomes important. In this testing, the interoperability testing is performed to examine the interconnectability which cannot be checked in the conformance testing. In this paper, we propose the Intelligent OSI Protocol Monitor which observes PDUs (Protocol Data Units) exchanged between OSI system and analyzes the protocol behaviors as well as the PDU format according to the protocols of OSI 7 layers. This monitor is effectively used to observe the actual communication for a long period and to detect protocol errors which cannot be detected by the conformance testing. This paper describes the detailed design of the Intelligent OSI Protocol Monitor and shows how it works for actual OSI protocols.

Keywords

Interoperability testing, protocol monitor, OSI

1 INTRODUCTION

As the standardization of OSI (Open Systems Interconnection) progresses, OSI protocols come to be widely adopted in various communication systems. As a result, the testing of OSI protocol implementations becomes important in order to realize the interconnection between OSI based communication systems. When a communication system is developed, it will be tested whether the system implements the relevant OSI protocols correctly. This testing is called the conformance testing and is useful to increase the possibility of the interoperability of OSI systems. However, the conformance testing focuses only on the conformity of an OSI system to the standard protocols, and therefore there are some possibilities that two OSI systems which have passed the conformance

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*

testing independently cannot communicate with each other, for example, because of some incompatibility of the usage of optional parameters. Also, there are some possibilities that the conformance testing fails to detect some errors included in the systems.

The interoperability testing will be performed to resolve these problems of the conformance testing. In this testing, OSI systems are connected through networks and the interoperability will be examined by making them communicate with each other. Currently, there are some studies on the interoperability testing and they are categorized into two types of methods. One method introduces additional testing programs, such as lower tester and upper tester, in the communicating OSI systems [7, 8, 9]. This method assumes that the additional testing programs apply test sequences and observe responses for them. These test sequences can be generated based on the test case generation method for the conformance testing. The other method only monitors PDUs (Protocol Data Units) exchanged between the communicating OSI systems without using controlled test sequences. In this type of testing, protocol monitors are used to analyze exchanged PDUs [5, 10].

However, these two types of interoperability testing have the following problems:

- As for the interoperability testing using additional testing programs, it is possible that the additional programs may change the original behavior of the communication systems.
- Since the test sequences are generated based on the method adopted by the conformance testing, it may be possible that the errors which cannot be detected in the conformance testing cannot be detected again.
- As for the testing by monitoring exchanged PDUs, the currently available protocol monitors have only the functions to analyze the format of PDUs and exchanged data sequence, and protocol errors should be detected manually by test operators.

In order to resolve these problems, we have adopted the following approach. We select the testing method by monitoring exchanged data for avoiding the effects of additional testing programs. We introduce in a protocol monitor the functionality to analyze the protocol behaviors and to detect protocol errors in the communicating systems according to protocols of OSI 7 layers. We call this protocol monitor as Intelligent OSI Protocol Monitor. The Intelligent OSI Protocol Monitor observes the actual communication between OSI systems, checks both the PDU formats and the protocol behavior, and finds errors if the PDU formats and the behavior does not conform to the standard protocols.

This paper describes the design of the Intelligent OSI Protocol Monitor. The next section and section 3 describe the design principle and the detailed design of the Intelligent OSI Protocol Monitor, respectively. Section 4 gives an example on how this monitor works for the OSI Transport and Session protocols [2, 3]. Section 5 gives some discusses on this monitor and section 6 makes the conclusion on our researches.

2 DESIGN PRINCIPLES

We have adopted the following principles on the design of Intelligent OSI Protocol Monitor.

1. The Intelligent OSI Protocol Monitor observes PDUs exchanged between communicating

OSI systems, and emulates the behavior of individual OSI systems separately according to observed PDUs. In the configuration where OSI systems A and B are communicating, an observed PDU in the direction from A to B will be handled as a sent PDU by the emulation of system A and as a received PDU by the emulation of system B.

2. The Intelligent OSI Protocol Monitor emulates the behavior of an OSI system based on the layered structure model, that is, it emulates the behavior of individual layers in one OSI system. The primitives exchanged between the layers are estimated by the monitor.

3. The emulation of individual layers is invoked by the observation of PDUs based on the following procedure.
• When a received PDU of a layer is observed, the monitor emulates the behavior of the layer in the case that the PDU is applied. If the behavior includes sending out of a PDU or a primitive to the lower layer, the monitor expects the sending, and confirms it when corresponding PDU is observed.
• When a sent PDU of a layer is observed in the case that some PDUs are expected to be sent, the monitor searches for the input (primitives from the higher layer or timeouts) which generates the PDU, and it considers that the input is applied to the layer. If it is a primitive from the higher layer, it is reported to the higher layer as an issued primitive.

4. During the emulation, the monitor checks the following protocol errors.
• PDU format error, such that a PDU does not have the mandatory parameters or the order of parameters is wrong.
• PDU parameter value error, such that parameter values are out of range defined by the protocol.
• PDU mapping error, such that a PDU is included in a wrong type of lower layer's PDU.
• State transition error, such that an inopportune PDU or a PDU including invalid parameter values for the current state is sent out.
If an invalid PDU is sent out, the monitor decides that the system which sends the PDU has protocol errors.

5. The Intelligent OSI Protocol Monitor maintains the state of each layer to emulate the behavior of each layer. If the state of a communicating OSI system is known when the monitoring starts, then the state can be given by test operators. If not, the monitor estimates the state by using IO (Input/Output) sequences which determines the state before or after they are observed.

3 DETAILED DESIGN

3.1 Overview

Figure 1 shows the structure of the Intelligent OSI Protocol Monitor. The monitor consists of the modules depicted in Figure 1. The Frame Capturing Module captures frames transmitted in both directions through the transmission line. As described in section 2, the System Emulating Modules which emulate the behavior of OSI systems are provided separately for the individual systems (systems A and B in Figure 1), the System Emulating Module is decomposed into a set of

Emulating Modules for individual layers. The (N) Emulating Module is provided with sent or received (N)-PDUs or issued or received (N-1)-primitives, which are estimated in the (N-1) Emulating Module. It emulates the (N) layer protocol and generates (N+1)-PDUs or (N)-primitives to be reported to the (N+1) Emulating Module. The monitored PDUs and the errors are shown by the Displaying Module.

Figure 2 shows the internal structure of the (N) Emulating Module. It consists of the (N) PDU Analyzing Module, the (N) State Identification Module and the (N) Behavior Control Module. The (N) PDU Analyzing Module decodes (N)-PDUs along with checking the format of (N)-PDUs. The decoded (N)-PDUs and (N-1)-primitives are given to the (N) Behavior Control Module.

The (N) Behavior Control Module emulates the behavior of the (N) protocol based on the State Transition Table of (N) layer. This emulation is performed for the related PDUs, such as PDUs over the same connection of the connection oriented protocol and PDUs with the same

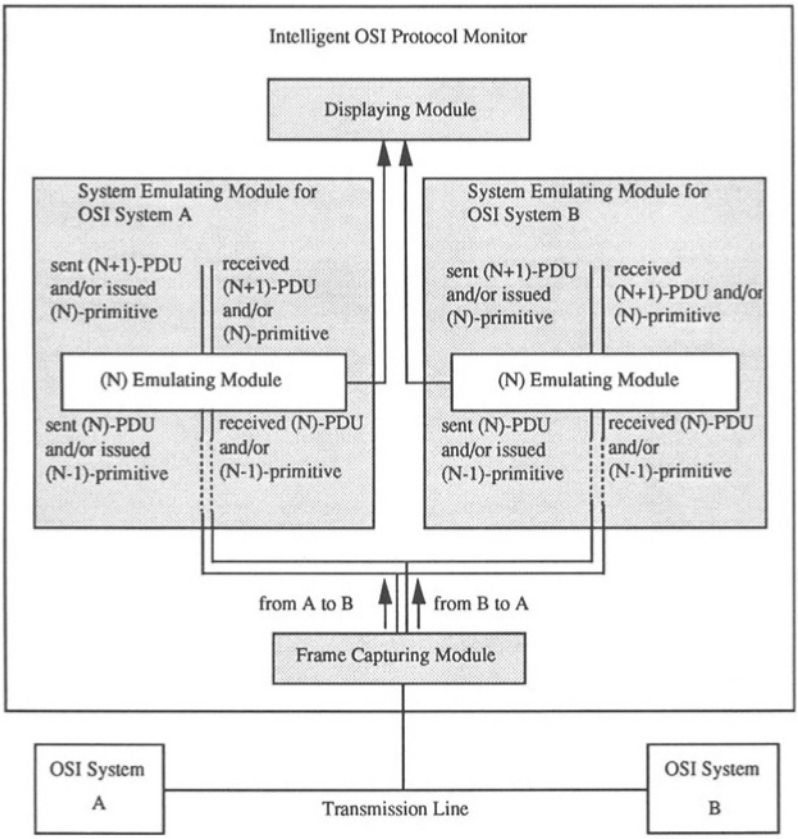

Figure 1 Configuration of Intelligent OSI Protocol Monitor.

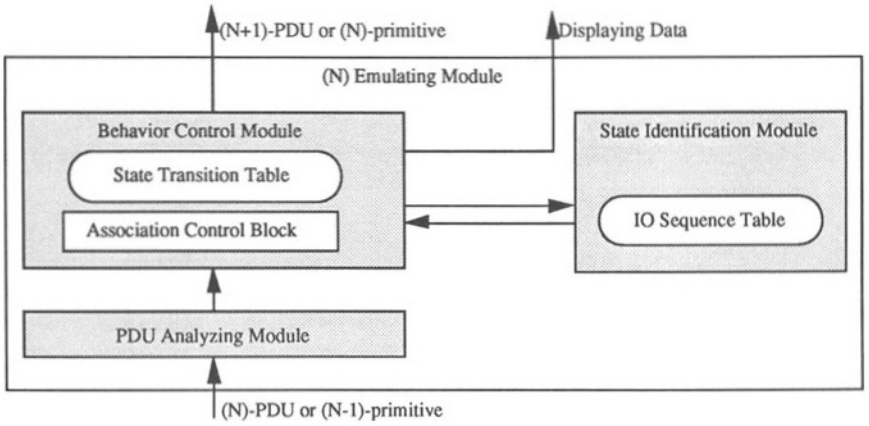

Figure 2 Configuration of (N) Emulating Module.

Data Unit Identifier which CLNP (Connectionless Network Protocol) uses for the segmentation. Therefore, the (N) Behavior Control Module manages the Association Control Block for maintaining the association, such as the connection, between the peer (N) layers. This block includes the emulating-status listing up the candidates of states which the (N) layer may possibly take. In the case that the (N) Behavior Control Module does not know the state of (N) layer, it stores the decoded PDUs and primitives, provides the (N) State Identification Module with the PDUs and/or primitives, and asks for identifying the state of (N) layer. The (N) State Identification Module maintains the IO Sequence Table which specifies the sequences of (N)-PDUs and (N-1)-primitives which can identify the state of (N) layer before or after the sequence is observed. By use of this table, the State Identification Module estimates the state and report it to the Behavior Control Module.

In the case that the (N) Behavior Control Module has identified the state, it emulates the behavior of (N) layer using the State Transition Table with the following approaches.

1. The (N) Behavior Control Module maintains all possible states of (N) layer in the emulating-status of the Association Control Block. When received (N)-PDUs and (N-1)-primitives or sent (N)-PDUs and issued (N-1)-primitives are provided, the transition from each states in the emulating-status is examined. If all transitions from a state are inconsistent with provided (N)-PDUs and (N-1)-primitives, then the state will be deleted from the emulating-status. If all states are deleted from the emulating-status, the (N) Behavior Control Module decides that some protocol errors occurred.

2. As for received (N)-PDUs and (N-1)-primitives, it checks this transition in the State Transition Table. If the transition sends out any (N)-PDUs and/or (N-1)-primitives, this module defers the emulation of this transition until the expected outputs are sent out or until some period of time passes. This is performed by use of the processing-flag in the emulating-status and the processing-

timer. If the transition does not send out any (N)-PDUs and (N-1)-primitives, it is emulated at this stage.

3. As for sent (N)-PDUs and issued (N-1)-primitives, there are two cases of handling by the (N) Behavior Control Module. In the case that the processing-flag is set, the module checks whether the transition focused on sends out the sent (N)-PDU and/or issued (N-1)-primitive. If so, this transition is emulated and the state is changed to the new state for the transition. If not, the (N) Behavior Control Module deletes the state from the emulating-status.

4. In the case that the processing-flag is not set, the (N) Behavior Control Module checks whether the sent out PDUs and/or primitives can be sent in the current state, by referring to the State Transition Table. If so, the module emulates that the event which sends out the PDUs and/or primitives has occurred, and if the event is an (N)-primitive, it is reported as an issued (N)-primitive to the (N+1) Emulating Module. If not, the state is deleted from the emulating-status.

3.2 PDU Analyzing Module

The (N) PDU Analyzing Module is provided with (N)-PDUs together with (N-1)-primitives in which the PDUs are contained. It checks the following items for the PDUs:

- the format of PDUs
- the constraint for PDU parameters defined by the protocol
- the mapping between the (N)-PDU and the (N-1)-primitives.

If there are no errors, the PDUs are handled as valid PDUs sent or received, and the decoded PDUs are provided for the (N) Behavior Control Module together with the (N-1)-primitives. If there are any errors detected, the PDUs are handled as invalid ones and are reported with error information. The PDUs are also provided for the Displaying Module.

3.3 State Identification Module

The (N) State Identification Module checks whether the sequence of (N)-PDUs and (N-1)-primitives provided by the (N) Behavior Control Module match any IO sequence maintained in the IO Sequence Table. If the matched IO sequence is found, the module identifies the state before or after the sequence is observed depending on the type of the IO sequence, and reports the identified state to the (N) Behavior Control Module.

3.4 Behavior Control Module

3.4.1 Data Structure

As described in section 3.1, the (N) Behavior Control Module uses the Association Control Block. In the case of the connection oriented protocol, it includes the following elements:

- the emulating-status which is a list including the state, processing-flag, input-id, transition-id and output-id
- the identifiers for the (N) connection and (N-1) connection
- parameter variables for the connection defined by the (N) protocol, such as reference of the Transport Protocol and the functional unit of the Session Protocol
- the buffer for (N)-PDUs and (N-1)-primitives while the state is not identified
- (N)-SDU buffer for reassembling and resequencing
- the buffer for received (N)-PDUs and (N-1)-primitives.

The emulating-status is used for two purposes. One is to maintain all the possible states which the (N) layer may take and the other is to maintain all the possible transitions caused by received (N)-PDUs and (N-1)-primitives and with (N)-PDUs and/or (N-1)-primitives sent out. The processing-flag shows that a transition is during processing and the observation of outgoing (N)-PDUs and/or (N-1)-primitives is being waited for. The input-id and the transition-id specify the current input and transition in the corresponding entry of the State Transition Table. The output-id indicates which outgoing PDUs and primitives are waited for in the case that the corresponding entry has more than one outputs. The buffer for received (N)-PDUs and (N-1)-primitives is used for deferring the handling of the PDUs and/or primitives as described below.

The State Transition Table used for emulation is different from the table standardized. All possibilities of outputs are written in the action field as alternatives. For example, if an output of a PDU is an option, the alternatives are the output of the PDU and no output.

3.4.2 Emulating Algorithm When State is Identified

When the state of (N) layer is identified, the (N) Behavior Control Module will emulate the behavior of (N) layer based on the following algorithms.

1. Handling of received (N)-PDU and (N-1)-primitive

- The module searches for the entry of the Association Control Block corresponding to the PDU and/or primitive.
- If there is at least one state whose corresponding processing-flag is set in the emulating-status, the handling of the PDU and primitive will be deferred until the handling of the expected sent PDUs and/or issued primitives is completed. The received PDU and/or primitive is stored in the buffer for received (N)-PDUs and (N-1)-primitives in the Association Control Block.
- The received (N)-PDU and/or (N-1)-primitive values are checked with the parameter variables for the connection.
- For each state in the emulating-status, the following procedure is performed for all the transitions specified in the entry for the state and the PDU and/or primitive:

  - If the transition does not send out any (N)-PDUs and (N-1)-primitives, this transition will be emulated. This emulation includes performing the corresponding actions in the transition, such as reassembling and resequencing using (N)-SDU buffers, updating parameter variables corresponding to the parameter values in the PDU and/or primitive, and reporting (N+1)-PDUs and/or (N)-primitives to the (N+1) Emulating Module if they are generated. The current state is changed to the new state transferred by the transition.

• If the transition sends out any (N)-PDUs and/or (N-1)-primitives, the emulation of this transition is deferred by using the following steps:

  • setting the processing-flag of the emulating-status,
  • storing the identifier of the received (N)-PDU and/or (N-1)-primitive in the input-id of the emulating-status,
  • storing the identifier of the transition focused on in the transition-id of the emulating-status,
  • storing the identifier of the outgoing (N)-PDU or (N-1)-primitive expected to be observed in the output-id of the emulating-status, and
  • start the processing-timer.

The handling of this transition is performed when a sent (N)-PDU and/or an issued (N-1)-primitive is observed as described below. In the case that the processing-timer expires, this transition is also handled in the same way as the case that (N)-PDU and/or (N-1)-primitive observed.

2. Handling of sent (N)-PDUs and issued (N-1)-primitives

• The module searches for the entry of the Association Control Block corresponding to the PDU and/or primitive.
• The sent (N)-PDU and/or issued (N-1)-primitive values are checked with the parameter variables for the connection.
• For each state in the emulating-status, the following procedure is performed:

  • If the processing-flag associated with the state is set, the (N) Behavior Control Module checks whether the output specified by the output-id in the transition specified in the State Transition Table which is identified by the state, the input-id and the transition-id is the sent (N)-PDU and/or (N-1)-primitive focussed on. If so, the output-id is set to the next output in the transition if it exists, or the transition is emulated as described above and the next state is stored in the state of the emulating-status with the processing-flag cleared. After the processing-flag is cleared, the buffer for received (N)-PDUs and (N-1)-primitives is checked and the stored PDUs and/or primitives are processed as described above, if they exist.

  If the output is not the same as the PDU and/or primitive focussed on, the state will be deleted from the emulating-status.

  • If the processing-flag is not set, the (N) Behavior Control Module searches for the transition in the state which sends out the PDU and/or primitive focussed on. This is performed by looking up all of the entries for the state in the State Transition Table. If such a transition exists, the (N) Behavior Control Module emulates it. This emulation includes performing the corresponding actions in the transition, such as reassembling and resequencing using (N)-SDU buffers, and updating parameter variables corresponding to the parameter values in the PDU and/or primitive. The current state is changed to the new state transferred by the transition. If the transition is generated by (N)-primitive given by the higher layer, the (N) Behavior Control Module reports it to the (N+1) Emulating Module.

  If the transition sending out the PDU and/or primitive focussed on does not exist, the state will be deleted from the emulating-status.

• If all the states are deleted from the emulating-status as the result of the above procedure, the (N) Behavior Control Module decides that the PDU and/or primitive will be a protocol error.

3.4.3 Handling When State is not Identified

When the state of (N) layer is not identified, the (N) Behavior Control Module stores the (N)-PDUs and (N-1)-primitives in the buffer for (N)-PDUs and (N-1)-primitives in the Association Control Block. The module also provides the identifier of (N)-PDUs and (N-1)-primitives for the (N) State Identification Module. If the state is identified by the (N) State Identification Module, it returns the state before or after a specific (N)-PDU and/or (N-1)-primitive is observed. The (N) Behavior Control Module starts the emulation from the point where the state is identified using the stored (N)-PDUs and (N-1)-primitives, following to the algorithm described in section 3.4.2.

4 EXAMPLE ON HOW INTELLIGENT OSI PROTOCOL MONITOR WORKS

This section demonstrates how the Intelligent OSI Protocol Monitor works by taking as an example OSI Transport Protocol class 4 over CLNS (connectionless network service) and OSI Session Protocol.

4.1 Examples of Tables and Variables

Tables 1 and 2 show a part of the State Transition Tables maintained in the Transport and Session Behavior Control Modules of the Intelligent OSI Protocol Monitor, respectively. An entry of these tables specifies the transition for an input is received in a state. It includes the output both to the higher layer and to the lower layer, and the next state. If there are more than one alternatives for the transition, they are specified in the entry with the identifier such as (trans1).

The Behavior Control Module also maintains the parameter variables which the protocol defines. The Transport Behavior Control Module maintains the following variables corresponding to the sequence number of TPDU:

• V(S) : the sequence number to be sent next
• V(LS) : the lower window edge for sending
• V(US) : the upper window edge for sending
• V(R) : the sequence number to be received next
• V(LR) : the lower window edge for receiving
• V(UR) : the upper window edge for receiving.

Table 3 shows an example of IO Sequence Table maintained in Transport and Session State Identification Module. The estimated state and before/after flag are described together with each IO sequence. '-' for inputs means that the inputs cannot be observed.

Table 1 State Transition Table for Transport Protocol class 4 (partially)

	CLOSED	*WFCC*	*OPEN*	*CLOSING*	*REFWAIT*
TCONreq	CR WFCC				
CC	DR CLOSED	(trans1) TCONconf, AK OPEN (trans2) TDISind,DR CLOSING	AK OPEN	CLOSING	DR REFWAIT
TDTreq			DT OPEN		
DT	CLOSED	DR CLOSING	(trans1) AK OPEN (trans2) OPEN	CLOSING	REFWAIT
DT (eot)	CLOSED	DR CLOSING	(trans1) TDTind, AK OPEN (trans2) TDTind OPEN	CLOSING	REFWAIT
AK	CLOSED	DR CLOSING	OPEN	CLOSING	REFWAIT
timeout	CLOSED	CR WFCC	DT OPEN	DR CLOSING	CLOSED

eot : End of TSDU
DT : DT TPDU with eot=0
DT (eot) : DT TPDU with eot=1

Table 2 State Transition Table for Session Protocol (partially)

	STA01	*STA01B*	*STA02A*	*STA713*
SCONreq	TCONreq STA01B			
TCONconf		CN STA02A		
AC		(trans1) S-P-ABORTind, AB(nr) STA16 (trans2) S-P-ABORTind,AB(r) STA01A	SCONconf STA713	(trans1) S-P-ABORTind, AB(nr) STA16 (trans2) S-P-ABORTind,AB(r) STA01A (trans3) S-P-EXEPTION-REPORTind,ER STA20

AB(r) : AB SPDU with reuse of transport connection
AB(nr) : AB SPDU with no reuse of transport connection

4.2 Examples of Emulation

We show the emulation performed by the Intelligent OSI Protocol Monitor when the PDU sequence depicted in Figure 3 is observed. This sequence represents a normal connection establishment phase in Transport and Session Protocols.

Table 3 An example of IO Sequence Table

IO sequence	*state*	*before/after*
- / CR	WFCC	after
CC / AK	OPEN	after
DT / AK	OPEN	after
- / AK	OPEN	after
CC / DT	OPEN	after
DT / DT	OPEN	after
- / DT	OPEN	after

(a) Transport

IO sequence	*state*	*before/after*
- / TCONreq	STA01	before
TCONind / TCONresp	STA01	before
TCONconf / CN	STA01B	before

"-" means input cannot be observed

(b) Session

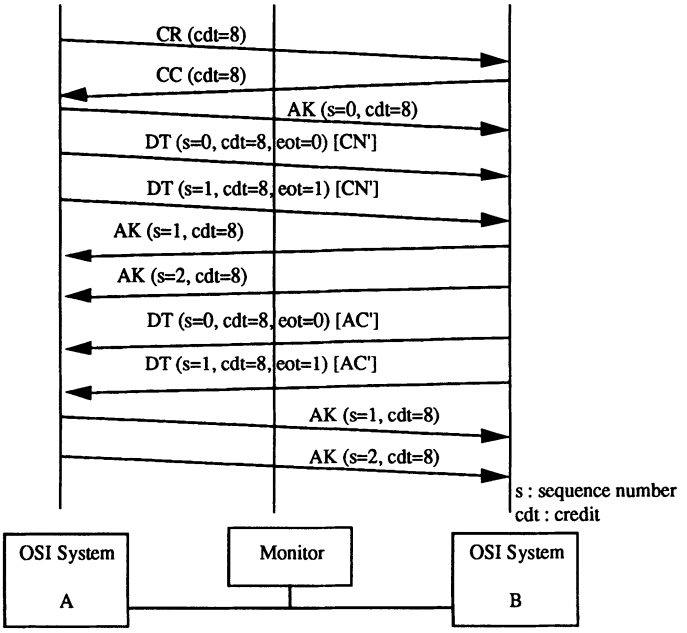

Figure 3 An example of protocol sequences at Transport and Session Protocol

Figure 4 shows how the monitor emulates the behavior of system A, the initiator side. At first, the state is not identified. When CR TPDU is observed, the monitor handles it as a sent PDU. Since the sequence '- / CR' is an IO sequence after observation for state WFCC, the state of Transport layer is identified as WFCC.

Then, the monitor observes the received CC TPDU for system A. Since there are two alternatives for this input, both of which have a sent out TPDU, the following two elements are registered in the emulating-status. They consist of state, processing-flag, input-id, transition-id and output-id.

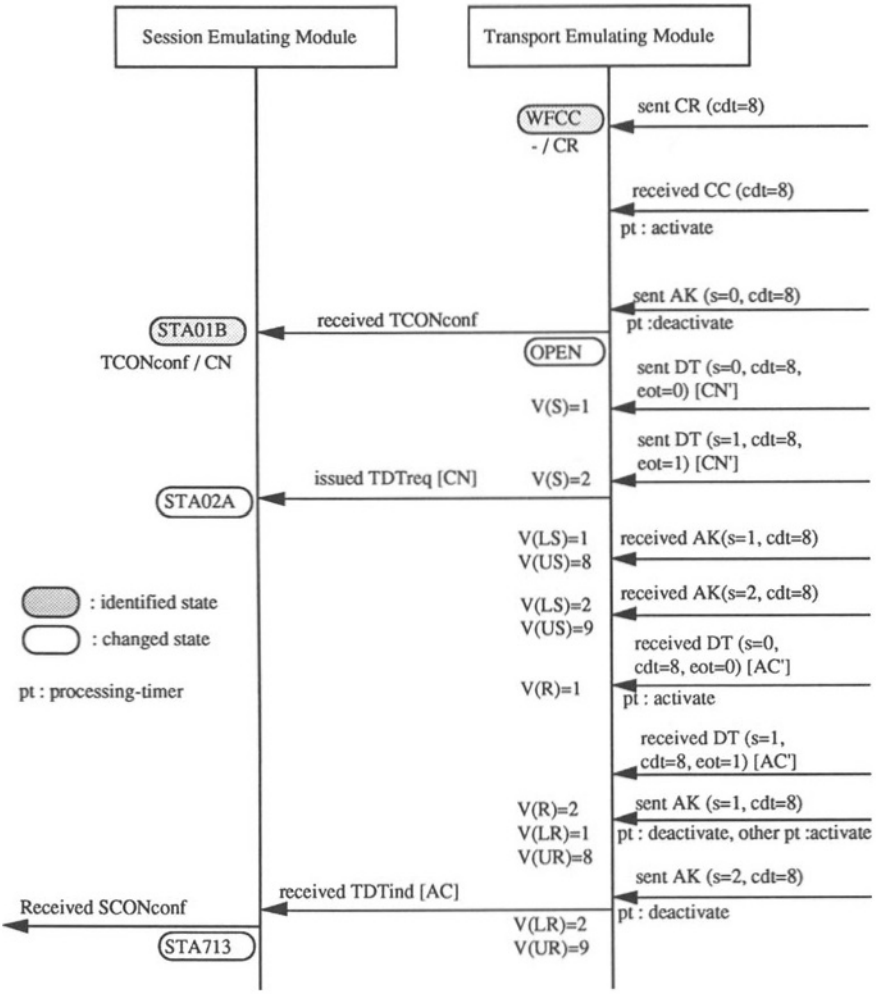

Figure 4 An example of emulation for Transport and Session Protocol

{WFCC, 1, CC, trans1, AK} and {WFCC, 1, CC, trans2, DR}

At this stage, the processing-timer is started in order to wait for the sending of any TPDU.

In the Figure 4, a sent AK TPDU is observed before the processing-timer expires and the monitor emulates that the first element of the emulating-status is selected. A received TCONconf is reported to the Session Emulating Module and the emulating-status is changed to

{OPEN, 0, null, null, null}.

At this time, the state in the Session layer is not identified and TCONconf is not an IO sequence.

Therefore, TCONconf is stored in the buffer.

Then, two DT TPDUs which contain a segmented CN SPDU are observed and V(S) is set to 1 and then to 2. Since the eot parameter of the second DT TPDU is set to 1, the user data is reassembled. At this stage, the Transport Emulating Module will search for the transition which sends DT TPDU in state OPEN and find that the input is TDTreq. TDTreq with CN SPDU is reported to the Session Emulating Module as an issued primitive.

Then, the Session Emulating Module observes the received TCONconf and the sent of TDTreq with CN SPDU and finds that this sequence corresponds to an IO sequence before the observation for state STA01B. The Session Emulating Module determines the state before the received TCONconf is STA01B and performs the emulation for received TCONconf and sent CN. As a result, the state is set to STA02A.

Then, the Transport Emulating Module observes two received AK TPDUs and the V(LS) is updated according to the parameter values. After that, it observes the received DT TPDU and sets V(R) to 1. In this case, two elements of emulated-status,

{OPEN, 1, DT, trans1, AK} and {OPEN, 0, DT, trans2, null},

are registered. Since the next received DT TPDU is observed before the processing-timer expires and next received TPDU is observed, the PDU is stored in the buffer and the handling is deferred. When the sent AK TPDU is observed, the Transport Emulating Module determines the first transition has been selected and handles the DT TPDU stored in the buffer.

Again, the corresponding entry for the DT TPDU has a transition sending out a TPDU, two elements of emulated-status,

{OPEN, 1, DT, trans1, AK} and {OPEN, 0, DT, trans2, null},

are registered. When the last sent AK TPDU is observed, the user data are reassembled and TDTind with is reported to the Session Emulating Module and the variables are updated.

The reported TDTind with AC SPDU is handled by the Session Emulating Module and the received SCONconf is reported to the Presentation Emulating Module and the state is changed.

5 DISCUSSIONS

1. It is considered that our Intelligent OSI Protocol Monitor is applied effectively to the testing of the communication systems which passed the conformance testing. Our monitor can observe the systems' behaviors for a long period and emulate them according to the protocol reference it has by the form of state transition table. If the systems have any errors which are difficult to find, it takes a long period for the errors to appear. Such errors are difficult to find by the conformance testing and our monitor is more appropriate.

2. As described in section 1, some research activities introduce the additional testing programs into the interoperability testing and generate the test sequence based on the generation method of the conformance testing. The test sequence generation method of the conformance testing can be categorized into the transition tour method [6], the UIO (Unique Input Output) sequence method [1] and the state identification method [4]. It is considered that the error detecting capability is the smallest for the transition tour method and the largest for the state identification method. On the other hand, the Intelligent OSI Protocol Monitor can be considered as a testing system which uses uncontrollable sequence as a test sequence. The emulating algorithm adopted

by our monitor trace the behavior of OSI systems according to the state transition table and therefore our algorithm corresponds to the transition tour method. We think that this is reasonable for the monitoring based testing method because of the following reasons:

- Since the sequence is not controllable, the UIO sequence and the distinguishing sequence cannot be applied.
- Our monitor can detect errors by emulating the behavior of a system for a long period and by finding the difference between the system and the reference.

3. There is a delay between the time system A sends a PDU and the time system B receives the PDU, and the monitor captures the PDU in the different timing from them. Therefore, the order of PDUs in which a system actually sends or receives might be different from the order in which the monitor detects.

We handle the PDU crossing depicted in Figure 5 in the following way. In case (a), the Transport Emulating Module of system B understands that CC TPDU is sent out by received CR TPDU in one transition and stores crossed DR TPDU in the buffer. However, in case (b), the Transport Emulating Module of system A detects CC TPDU at WFCC and it differs from the behavior of system A. Then, the Transport Emulating Module would estimate the emulating-status, {WFCC, 1, CC, trans1, AK} and {WFCC, 1, CC, trans2, DR}, and, since the DR TPDU observed next has different reason parameter value from expected DR TPDU, it would be considered as a protocol error. This might be resolved by reordering the emulation for received CC TPDU and sent DR TPDU. In the actual environment, more than one PDUs might cross as shown in case (c). In this case the Transport Emulating Module must consider the possibility of many cases. In order to cope with such a situation, it might be necessary to consider the propagation delay between the monitor and systems and to reordering the emulation.

4. When a protocol error has occurred, the state and variables will be reset or will not be reset according to how severe it is. If the protocol error has been caused by sending an invalid parameter value, only the corresponding variable might be reset, but if it has been caused by sending an invalid type of PDU, the state and all variables would be reset. When a protocol error has occurred in (N) layer, it need to be determined to report the error to (N-1) layer and/or (N+1) layer also for individual errors.

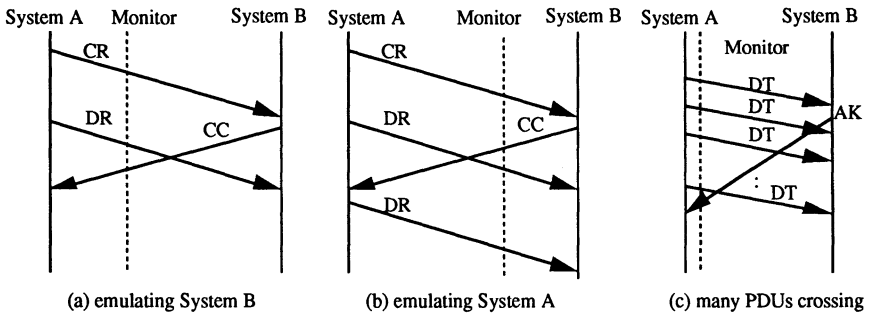

Figure 5 An example of PDU crossing

6 CONCLUSION

In this paper, we have described Intelligent OSI Protocol Monitor which observes PDUs exchanged between OSI system and analyzes the protocol behaviors as well as the PDU format according to the protocols of OSI 7 layers. This monitor would be very useful in the interoperability testing since it can detect protocol errors which cannot be detected by the conformance testing by monitoring the actual communication for a long period.

This monitor emulates the behavior of an OSI system based on the layered structure model. It has the Emulating Module for each layer which emulates the behavior of the protocol in the layer based on the state and the state transition table. This emulation is invoked by the observation of PDUs and all the functions defined in the protocol are emulated. In order to cope with the case when the state of the OSI system is not identified, the monitor provide state identification mechanism by use of IO sequence.

This paper have described the detailed design of the Intelligent OSI Protocol Monitor including the structure and the emulating algorithm. It has also demonstrated how the monitor works for the actual OSI protocols by taking OSI Transport Protocol class 4 over CLNS (connectionless network service) and OSI Session Protocol as examples.

7 ACKNOWLEDGEMENT

The authors wish to thank Dr. Y. Urano, Director of KDD R&D Laboratories, for his continuous encouragement of this study.

8 REFERENCES

[1] Bosik, B.S. and Uyar, M.U. (1991) Finite state machine based formal methods in protocol conformance testing: from theory to implementation, Computer Networks and ISDN Systems, vol 22.

[2] CCITT (1988) Recommendation X.225 - Session Protocol Specification for Open Systems Interconnection for CCITT Applications.

[3] ITU-T (1993) Recommendation X.224 - Protocol for Providing the OSI Connection-Mode Transport Service.

[4] Hennie, F.C. (1964) Fault-detecting experiments for sequential circuits, Proc. of 5th Ann. Symp. on Switching Circuit Theory and Logical Design, 95-110.

[5] Kato, T. and Suzuki, K. (1993) Development of OSI 7 Layer Link Monitor, Proc. of the 46th Annual Convention IPS Japan, vol.1, 211-212.

[6] Naito, S. and Tsunoyoma M. (1981) Fault detection for sequential machines by transition tours, Proc. of IEEE Fault Tolerant Comput. Conf.

[7] Okazaki, N, Park, M.R., Takahashi, K. and Shiratori, N. (1994) A New Test Sequence Generation Method for Interoperability Testing, Proc. of the 7th International Workshop on Protocol Test Systems, 229-245.

[8] Rafiq, O. and Castanet, R. (1990) From conformance testing to interoperability testing, Proc. of the 3rd International Workshop on Protocol Test Systems, 371-385.

[9] Takahashi, K., Suzuki, S., Sawai, K., Hatafuku, M.,Gotoh, K. and Kazama, K. (1994) Design and Implementation of an Interconnectability Testing System - AICTS, Proc. of the 7th International Workshop on Protocol Test Systems, 119-134.

[10] Tekelec (1992) Chameleon User's Manual.

9 BIOGRAPHY

Tomohiko Ogishi is a member of High Speed Communication Group, KDD R&D Labs. Since joining the Labs in 1992, he worked in the field of computer communication. His current research interests include protocol testing and messaging services. He received the B.S. Degree of electrical engineering from the University of Tokyo, in 1992.

Akira Idoue is a research engineer of High Speed Communication Group, KDD R&D Labs. Since joining the Labs in 1988, he worked in the field of computer communication. His current research interests include software and hardware on communication protocols. He received the B.S. and M.E. Degree of electrical engineering from Kobe University, Kobe, Japan, in 1984 and 1986 respectively. He has received IPSJ Convention Award in 1992.

Dr. Toshihiko Kato is the manager of High Speed Communication Group, KDD R&D Labs. Since joining the Labs in 1983, he worked in the field of computer communication. From 1987 to 1988, he was a visiting scientist of CMU Computer Science Department and worked on distributed systems. He received the B.S., M.E. and Dr. Eng. Degree from the University of Tokyo, in 1978, 1980 and 1983 respectively. He has received Moto-oka Award in 1989. Since 1993, he has been a Guest Associate Professor of Graduate School of Information Systems, in the University of Electro-Communications.

Dr. Kenji Suzuki is a deputy director of KDD R&D Labs. Since joining the Labs in 1976, he worked in the field of computer communication. He received the B.S., M.E. and Dr. Eng. Degree of electrical engineering from Waseda University, Tokyo, Japan, in 1969, 1972 and 1976 respectively. From 1969 to 1970, he was with Philips International Inst. of Technological Studies, Eindhoven, The Netherlands as an invited student. He received Maejima Award from Communications Association of Japan in 1988, Achievement Award from the Institute of Electronics, Information and Communication Engineers in 1993, and Commendation by the Minister of State for Science and Technology (Persons of scientific and technological research merit) in 1995. Since 1993, he has been a Guest Professor of Graduate School of Information Systems, in the University of Electro-Communications. He is a member of IEEE.