FOREST: A Systematic Testing Environment
Based on Standardized Formal Description Techniques

Kotaro KATSUYAMA          Fumiaki SATO
Tetsuo NAKAKAWAJI         Tadanori MIZUNO

Information Systems and Electronics Development Laboratory
Mitsubishi Electric Corporation
5-1-1 Ofuna Kamakura-City, Kanagawa 247, JAPAN

Abstract

ISO and CCITT standardize not only protocol specifications for OSI, but also formal description techniques and testing methodology. SDL, LOTOS, and Estelle are standardized as formal specification languages. ASN.1 is a standardized language for protocol data format definition. TTCN is a standardized specification language for testing. These languages are standardized separately and their usage is not standardized. The systematic application of these languages is a future work of the user.

In this paper, we present the application of these languages for systematic conformance testing. We develop the testing environment FOREST (FORMal Environment for Systematic Testing), which uses formal description techniques systematically. FOREST generates the TTCN format test sequence from the specification described in SDL. Furthermore, FOREST generates the test data from protocol data definition described in ASN.1. FOREST has a test execution part which executes the test sequence. We establish the effectiveness of our approach by an application to the test design for CCR protocol.

1. Introduction

ISO and CCITT progressed the OSI (Open Systems Interconnection) and standardized many protocols for interconnection between heterogeneous systems. At the same time, the standardization of the specification description language based upon the formal description techniques (FDT) runs in parallel. SDL (Specification and Description Language)[1], LOTOS (Language of Temporal Ordering Specification)[2], and Estelle (Extended State Transition Language)[3] are standardized. Furthermore, ASN.1 (Abstract Syntax Notation One)[4,5] is standardized for the protocol data definition.

On the other hand, ISO standardizes the techniques for conformance testing, which certify that whether the system is conforming the protocol specification or not. ISO standardizes the methodology and framework for conformance testing and abstract test specification for every OSI protocols. The language for abstract test specification description, TTCN (Tree and Tabular Combined Notation)[6] is also standardized.

On the other hand, ISO standardizes the techniques for conformance testing, which certify that whether the system is conforming the protocol specification or not. ISO standardizes the methodology and framework for conformance testing and abstract test specification for every OSI protocols. The language for abstract test specification description, TTCN (Tree and Tabular Combined Notation)[6] is also standardized.

However, the formal specification languages (SDL, LOTOS and Estelle) and the methodology of conformance are developed separately. Therefore, the standardization research of the application to the conformance testing with FDTs is just starting[7]. Almost existing applications of FDTs are the application of only one language[8]. The effectiveness of the systematic application of the plural FDTs has never seen yet.

In this paper, we propose the systematic testing environment FOREST (FORMal Testing Environment), which uses the specification description language based upon SDL. Furthermore, we implement the FOREST and evaluate from the applying to the real protocol development.

In section 2, we present the outline of the formal description techniques and conformance testing. In section 3, we present the design concept and software structure and function. In section 4, we present how we use the specification description language. In section 5, we present the results from the application to the development of the CCR protocol[9].

0730-3157/90/0000/0483$01.00 © 1990 IEEE
2. Specification description language and conformance testing

2.1 Specification description language

SDL is developed at CCITT to specify switching systems. SDL is based upon an Extended Finite State Machine (EFSM) and has two representations; the graphical representation and text representation.

ASN.1 is standardized by ISO as the language for abstract syntax definition. ASN.1 is able to specify the data structure and format as the tree structure. This data structure and format are independent of the real computer and terminal.

TTCN is the language for test specification. TTCN describes the test sequence as the tree structure and the test specification as the table notation. At the present time, test specification has a tendency to use the TTCN for test sequence definition and the ASN.1 for the data format definition.

2.2 Conformance Testing

Conformance testing (CT) is researched in ISO about its methodology and framework[6]. The conformance testing method is divided into a local test and an external test. In the local test, test program and program under test are implemented in the same system. In the external test, the test system and system under test (SUT) are connected with communication medium.

The structure of the conformance test is indicated in fig.1.

![Diagram of conformance testing](image)

PDU : Protocol Data Unit
ASP : Abstract service primitive

Fig.1 A logical structure of conformance testing

A number of test cases are applied to the IUT by LT in cooperation with UT. A typical activity in test cases is to confirm the ASP (Abstract Service Primitive) from IUT and (N) layer protocol data unit (PDU) when the upper tester or lower tester gives the ASP or PDU to the IUT.

The use of the TTCN for the testing specification is recommended from ISO. For each protocols, the abstract test specification is also standardized in ISO.

2.3 Application of specification description language to CT

Using a formal specification language among the specification languages described above, the two applications to the conformance testing exist.

(1) Static generation

Test sequences are generated from the protocol specification described in the formal specification language before test execution.

(2) Dynamic generation

Test events are generated in the testing with the tester which interprets the state transition described in the formal description language. The NBS system[10] uses this idea.

The first idea has the problem that the test sequence length may become too long, however, it has also the benefit that the test context is found before test.

The second idea has the benefit that the testing system and specification interpreter are in a body, however, if the user wants to review the test specification before testing, the idea is not suitable.

3. FOREST system

3.1 Design concepts

The purpose of FOREST is to verify the effectiveness of the formal specification and the realizability of the conformance testing methodology. The design concepts of FOREST are the following.

(1) This system uses the standardized formal specification language and based upon the conformance testing methodology.
Since this system will be used in the real development cycle, the documentation of the test specification is an important item. Therefore, we use the first idea in the section 2.3.

For efficient testing, the documentation of the test specification, the test execution, and the documentation of the test results are supported by a workstation.

The present target protocols of this system are the application layer and presentation layer of the OSI.

3.2 The architecture of FOREST

FOREST is implemented in a workstation as a system integrating several tools. The architecture of FOREST is shown in Fig. 2. The elements of FOREST are the test execution environment and the following three tools.

(1) TENT [11,12]

TENT realizes the sequence generation function described in section 3.1. The input of TENT is SDL/PR file. TENT generates the test specification from the SDL process definition. TENT translates the SDL file to an internal transition table. TENT has a function to generate test sequences selectively, which makes it easy to generate the test sequences for specific test purposes.

Generated test sequences are placed in the test specification file as shown in Fig. 2. The generated test sequences are manually checked before use.

(2) APRICOT [13]

APRICOT is the tool for test data generation function described in section 3.1. APRICOT has three parts; i) the pre-compiler to generate the layered structure table (type table) from ASD.1 description, ii) the encoder/decoder library to encode/decode the structured protocol data, and iii) the debugger to analyze and display the protocol data structure described in ASD.1 for program testing.

Furthermore, APRICOT is extended the facility to generate the PDU from several ASD.1 descriptions of some serialized protocol layers, i.e. CCR PDU description and Presentation ASP description and Session ASP description. APRICOT uses a mapping file which indicates corresponding PDUs of different protocols.

(3) TESPEC

TESPEC provides the user-interface to describe the test specification and test report. The display format of the test specification is based on the TTCN.GR. The concept of the TESPEC is the parallel work of the description of the test specification and test sequence and test data generation, and the conformed work for the conformance testing methodology standardized in ISO. At first, user input the summary of the test, secondly, select the ASP and PDU used in the test. Then, the definition of the timer, upper tester, and lower tester are described. Next, the group and case specification of the test is described.

The parameters inputted in the definitions of the ASP and PDU are used as the input data to APRICOT. So, the test data is prepared at the same time. On the other hand, user input the information for limitation of the testing area with the definition of the groups and cases. TENT uses the information to generate the test sequence for the restricted area.
3.3 Test execution

3.3.1 Test execution environment

Test execution environment is indicated in Fig.3. Session Simulator[12] provides the function of OSI session layer service. Implementation under test (IUT) realizes the function of the presentation and application layer protocols. Upper tester controls and observes the events at the boundary of the IUT and the user (in this configuration user is lower tester).

The lower tester executes and controls the test. The test sequence generated by TENT is used for this test. The test log file is the protocol data generated by APRICOT.

The test log file is an output from lower tester. TESPEC refers to this file and makes the test reports. Since we aim to support the whole of the software development cycle by a workstation, we selected the local test method. However, when Session Simulator is switched by a real Session service, the test is able to execute the external test.

Furthermore, we use the Ferry method[14] for test coordination protocol which facilitates the coordination procedure for the upper and lower tester. Ferry method has the following features.

(1) Upper tester module on the system under test is simple and easy to implement and has good portability.

(2) It is easy to synchronize upper tester with lower tester and to get test results in lower tester.

We implemented the Ferry control protocol with connectionless service as described in section 3.3.3.

3.3.2 Test execution flow

Test is executed starting the Session Simulator, IUT, Upper tester and Lower tester, in this order. Lower tester reads the data from test sequence file and test data file and interprets the corresponding behaviour and records the test results to the test log file. An action of lower tester and upper tester are the following:

L! CONreq Lower tester gets the test data from test data file corresponding to the event CONreq and sends it to the IUT (Implementation Under Test).

U? CONind Lower tester commands the upper tester to receive the data from IUT through the Ferry control channel. The upper tester sends the received data to the lower tester after receiving. The lower tester compares the received data with prepared CONind data.

U! CONrsp Lower tester sends the test data CONrsp to the upper tester through the Ferry control channel. Upper tester sends the data to the IUT.

L? CONcnf Lower tester receives the event and compares the data with the prepared CONcnf data.

In Fig.4, the dotted lines crossing the box of the Session Simulator and IUT are the Ferry control protocol data. If the test detects an error, the test is stopped and the test results are analyzed.
3.3.3 Ferry control protocol

Ferry method can be summarized as follows: Ferry control channel is established independently of the channel for IUT and the Ferry control protocol is used to get the data from IUT and to send the data to the IUT. Fundamentally, the upper tester is passive and the verdict of the result is decided by the lower tester.

The Ferry control protocol is realized by using the connectionless service of the Session layer. To reduce the program of the upper and lower tester, we use the simple protocol. The state diagram of the upper and lower tester are shown in Fig.5a and Fig.5b. The protocol data units used in this protocol are shown in the Table 1.

4. The application to the OSI protocol

For the evaluation of the effectiveness of the FOREST system, we apply to the CCR[9] protocol development. CCR is one of the OSI application layer protocols.

4.1 Protocol specification

CCR is a protocol to synchronize the distributed system. CCR specification consists of the natural language, state table and ASN.1 description for protocol data formats. We developed an SDL specification from the natural language and state tables, and we used the ASN.1 definitions given in the standard.

In Fig.6, the behaviour of the state S_NONE is specified. In the state S_NONE, there are two types of input, BEG_req and OTHER (not BEG_req). When BEG_req comes in from the environment, if commitment state variable CV is P0, this protocol machine starts the ACT[1] and puts BEG and goes to the state W_BEGACK. If CV is P1, this protocol machine doesn't start ACT[1] and output is BEG and next state is W_BEGACK. When OTHER comes in, no output is produced and state gets unchanged.

The protocol data unit (PDU) is a test data unit and is shown described in ASN.1 in the Fig.7. The example in Fig.7 is a part of the PDU specification for a request of association establishment.

4.2 Test specification

TTCN has two representation styles; TTCN.GR which is the table representation, and TTCN.PR which is the text representation for the machine processing. We select the

![Fig.5a A state transition diagram for upper tester](image)

![Fig.5b A state transition diagram for lower tester](image)

<table>
<thead>
<tr>
<th>PDU</th>
<th>Direction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCP_ACTIVATE_REQ</td>
<td>LT -&gt; UT</td>
<td>Activate UT</td>
</tr>
<tr>
<td>FCP_ACTIVATE_ACK</td>
<td>LT &lt; UT</td>
<td>Responce of the request to activate UT</td>
</tr>
<tr>
<td>FCP_DEACTIVATE_REQ</td>
<td>LT -&gt; UT</td>
<td>Deactivate UT</td>
</tr>
<tr>
<td>FCP_DEACTIVATE_ACK</td>
<td>LT &lt; UT</td>
<td>Responce of the request to deactivate UT</td>
</tr>
<tr>
<td>FCP_RECEIVE_REQ</td>
<td>LT -&gt; UT</td>
<td>Request UT to receive the data from IUT</td>
</tr>
<tr>
<td>FCP_RECEIVE_ACK</td>
<td>LT &lt; UT</td>
<td>Responce of the request to receive the data</td>
</tr>
<tr>
<td>FCP_SEND_REQ</td>
<td>LT -&gt; UT</td>
<td>Request UT to send the data to IUT</td>
</tr>
<tr>
<td>FCP_SEND_ACK</td>
<td>LT &lt; UT</td>
<td>Responce of the request to send the data</td>
</tr>
</tbody>
</table>
Fig. 6 An example of SDL/PR specification.

```plaintext
PROCESS CCR_A;
STATE S_NONE;
INPUT BEG_req:
DECISION CY, 8/Commitment-State Variable Z/ (PB):
TASK ACT(1);
OUTPUT BEG;
NEXTSTATE W_BEGACK;
ENDDECISION;

INPUT OTHER:
OUTPUT netbin;
NEXTSTATE S_NONE;

STATE S_ACT;
INPUT ACT:
DECISION PI-WIN:
(Task) /n
OUTPUT netking;
NEXTSTATE S_ACT;
OUTPUT netbin;
NEXTSTATE S_NONE;
ENDDECISION;

Fig. 7 An example of ASN.1 specification.

```plaintext
```plaintext
AC_See = CHOICE
  ac_SeeLD AC_SAAND
  ac_SeeLAC AC_SAAND
  ac_SeeLAC AC_SAAND

AC_SAAND = APP 8/I3P Seq {
  ac_SAAND: INT, -- DEFAULT (not_set, 0)
  ac_SAAND: OBJ, -- DEFAULT (not_set, 0)

  ac_SAAND: CHOICE {
    ac_SAAND: INT, -- DEFAULT (not_set, 0)
    ac_SAAND: OBJ, -- DEFAULT (not_set, 0)
  }
}

AC_SAAND = CHOICE
  ac_SAAND: INT, -- DEFAULT (not_set, 0)
  ac_SAAND: OBJ, -- DEFAULT (not_set, 0)

AC_SAAND = CHOICE {
  ac_SAAND: INT, -- DEFAULT (not_set, 0)
  ac_SAAND: OBJ, -- DEFAULT (not_set, 0)
}

Fig. 8 An example of test sequence for CCR protocol.

```plaintext
```plaintext
L?  Lower tester receives the event.
L!  Lower tester sends the event.
U?  Upper tester receives the event.
U!  Upper tester sends the event.

A part of the test sequence is shown in Fig. 8.

In Fig. 8, the part following the character # is treated as comments including the state name. The text "(reset_seq)" indicates that this event is a transfer sequence to the initial state. For example, the line 5 in Fig. 9 indicates that the upper tester sends BEG_req in the state S_NONE, and the line 7 indicates that the lower tester sends BEG.

Fig. 9 is the window display image of Fig. 8. The part referred to as "Sequence" of Fig. 10 indicates the test sequence and this area is scrolled and modified. The "Print" button is for printing out the test sequence.

The parameters of the PDU can be set with the window interface indicated in Fig. 10. The user can click the item of the parameter and then set the parameter values. The parameter selected in the Fig. 10 is the underlined part of Fig. 7. The interface uses the indentation for displaying the structure of the ASN.1 description.

APRICOT has another interface to generate the test data, which can generate the parameter values semi-automatically. In the above interface, the parameter values are inputted by user, however, they have many conditions to be satisfied. The latter

```
```
Fig. 10 A display example of setting parameter.

interface has a condition file to describe parameter conditions and three modes to input parameter values: i) Full automatic generation - the parameter values are generated randomly and automatically, ii) Semi-automatic generation - the parameter values described in condition file are generated automatically and other values are set by user, iii) Interactive generation - the parameter values are set by user interactively.

5. Discussion

Because of the Ferry method, the upper tester becomes simple. When we shift to the external test method, the upper tester will have a good portability. Since the protocol between the upper and lower tester is simple, the module size of these testers are 0.5k steps for upper tester and 3k steps for lower tester and a total of 3.5k steps in C language.

Since FOREST is realized in a workstation, it is able to support the activities from test specification description to test execution and test reporting systematically.

There are some problems in the test environment.

(1) Test data generation
In FOREST, the verdict is based on the binary comparison of the PDU. Therefore, the data value of the reserved area of PDUs and the field with no relationship with the test must be agreed. So, the test data generation is difficult.

(2) Multiple connection
Since FOREST system does not support multiple connections, we could not test this feature.

From the viewpoint of the test sequence generation, the following problems exist.

(1) Protocol specification description
The association establishing part of the ACSE protocol is appended with the CCR protocol for real test sequence generation. In the CCR specification, when an accident arises, the protocol machine indicates the accident to the upper and lower layers and shifts to the next state. However, in this case, the reestablishment specification must be added to the CCR protocol specification. In the case of upper layer test, a single target protocol specification is insufficient for real testing.

(2) Non-determinism of the protocol
In FOREST, the variables of the protocol are assumed controllable from the tester. From the assumption, the predicates are divided into a number of alternatives of actions. However, CCR protocol has a non-deterministic variable: Commitment State Variable. Therefore, we removed the transition which handles this variable.

(3) Output notation
The output notation of FOREST uses the dynamic part of TTCN. The tester can read the text of the output notation and execute the test. The output notation has a comment about state and the sort of the sequence (reset sequence or state identifier sequence, etc). The comment is useful for the test result analysis.

Over ten errors are detected in these tests. The errors were arising from the insufficiency of the specification description and the error in the specification itself. That shows that the specification of CCR is not completed yet. TENT checks the state reachability during the test sequence generation from state table from SDL/PR. TENT checks the error of the specification itself. Therefore, the errors before the product testing are detected and this facility is effective in reducing the number of tests.

SDL is useful for the state transition specification, and it is easy to translate from SDL to the state transition table which is used for test sequence generation.

From the point of the test data generation, since we provide the interface for parameter setting of PDUs, the test with different values of the variable is executed easily.

The ASN.1 description of the PDU for test data refers to the specification directly, and it reduces the effort for parameter field design and so on.
The generated test specification is 136 pages for CCR. About 50% of this specification is the part of behaviour, which is generated from this system automatically. The titles of the paragraph and tables of the rest part are generated from this system.

6. Conclusions

We proposed and implemented the systematic testing environment FOREST, for the evaluation of formal description languages.

By using FOREST system, the test specification and test execution and test report description are achieved systematically. Therefore, the cost is reduced of the testing phase of OSI upper layer software development. Especially, the facility for test sequence generation of FOREST realizes high coverage with a reasonable cost. As the reduction of the effort for the test specification description, the facility is efficient for checking protocol specifications. It is the evidence of the effectiveness of the formal description techniques.

In FOREST, we use SDL for protocol specification description, however, Estelle, with the same language model, will be available. The translator of TENT is easy to modify for Estelle syntax. On the other hand, LOTOS can't be applied the test sequence generation method based on the Finite State Machine because of the different language model. However, the temporal order logic can be translated to Finite State Machine, since the before and after of the event can be regarded with the states of the system. In the future, LOTOS, with a different language model, will also be available in our system.

For the real application to the communication systems, we selected the peer-to-peer testing, since ISO standardizes only the communication with single connections. The future problem is the handling of multiple connections and the specification of the protocol with multiple connections.

Acknowledgment

We would like to thank Prof. Kazuo Ushijima and Prof. Behcet Sarikaya for reading and commenting on previous drafts of this paper.

References


