マルチメディア通信と分散処理　72－3
（１９９５．　９．　８）

形式仕様技術に基づく相互運用性試験について

岡田 康治
kokada@etl.go.jp
電子技術総合研究所情報アーキテクチャ部
〒305 茨城県つくば市梅園 1-1-4

形式記述に基づかない通常のプロトコル開発においては、相互運用性試験はプロトコルの個々の実装について正常に動作するか否かを検査する。個々の実装の元になったプロトコル仕様そのものが、サービス仕様を対して正当か否かを検査することは大変困難なことである。

本稿では、形式仕様技術に基づき、プロトコルの仕様からサービスの振る舞いの記述を合成し、それによって相互運用性を検査する方法を示す。我々は、そのような試験システムを形式記述言語 OBJ を使って実現した。試験システムと例題プロトコルについての実験の概要を示す。本方法はプロトコル開発の費用を軽減するのに非常に有効であると考えられる。

Realizing Interoperability Testing on Formal Specifications

Koji OKADA
Computer Science Division, Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305, JAPAN

In the conventional protocol development, interoperability testing is a way to check whether protocol implementations work poperly, or not. It is very hard to get certainty for the the correctness of the protocol specification which was derived from the given service definition.

Based on the formal description techniques, this paper presents a method to synthesize the description for the service behavior from the description for the protocol and to test interoperability on the obtained description. We have realized it in a formal specification language, OBJ[2, 3, 4]. The paper also shows the outline of the test system and the results of the experiments. The method introduced here would provide a very good conceptual tool to reduce the total development cost.

1 Introduction

In protocol development, there is a phase to derive a protocol specification from the given service definition. it has been a crucial issue how we shall be confident of the correctness of the derived protocol specification, i.e., that any implementations conforming the protocol specification would work, along with the underlying communication medium, as the service definition defines.

There are two ways to validate the correctness of protocols: *protocol verification* and *interoperability testing*. Although protocol verification as an intention means to prove its correctness mathematically, we at present can't prove the *correctness* of protocols but can only check some desirable attributes of them, e.g., reachability, " dead-lock free," etc.[10, 5] Interoperability testing is the more popular way for correctness validation[1, 12]. However, it is indirect and generally very much costly to get certainty.

Some *formal description techniques* (FDTs) were developed[7, 8] in the late 80's and have been believed to have many advantages. In formal methods of protocol development, if we could test interoperability using a formal description for a protocol specification without any implementation in early stages of development, it would reduce the total development cost greatly.

This paper presents a method to synthesize the description for the service from the description for the protocol and to test interoperability on the obtained service description. We have realized it in a formal specification language, OBJ[2, 3, 4]. The paper also shows the outline of the test system and the results of the experiments for a sample protocol, the sliding window protocol[9].

Our experimental realization of an interoperability test system in OBJ proves that the method introduced here is very much advantageous as previously expected because it is very easy, i.e., to require very little text.

2 Fundamental Concepts

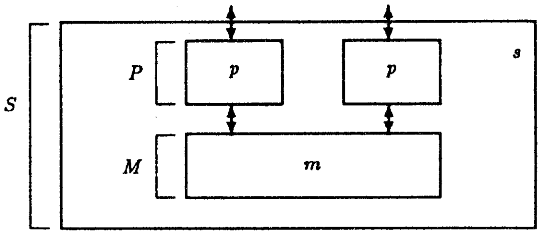
This section explains fundamental concepts of the paper's subject.

2.1 Layer Structure

Based on the layer structure of network architectures, our view for the constructs of each layer are described in the Figure 1. Note that here specifications (or descriptions) and their implementaions are strictly distinguished.

2.2 Protocol Verification

Given upper and lower service definitions S and M, verification of the protocol specification P is to prove that the following expression is satisfied:



p:	protocol entity	P:	protocol specification for p
m:	comm. medium	M:	service definition for m
s:	service	S:	service definition for s

Figure 1: Constructs of a Layer

$$\forall p \triangleleft P, \forall m \triangleleft M : syn(p, m, p) \triangleleft S$$

where "$syn(p, m, p)$" expresses some synthesized entity with left and right *protocol entities* (PEs) and a *communication medium*, and "$\triangleleft$" expresses conformity.

2.3 Interoperability Testing

Interoperability testing in the conventional methods is to test the condition given in the above expression by using some specific implementations p_i, p_j, and m_i for protocol entities and a medium. It is very much difficult to get certainty of the correctness of the protocol specification, not of the specific implementations, by testing for the various combinations of the three elements of the service.

In this paper, being oriented to a formal method for protocol development, we trys to realize interoperability testing on formal descriptions, i.e., without any implementations. It can be done in an early stage of the whole processes and would reduce the total development cost greatly because it would prevent unnecessary feedbacks.

3 Interoperability Testing on FD

The method of interoperability testing on formal descriptions, introduced in this paper, is based on the two following ideas:

- the universal communication entity with a pair of bi-directional ports
- parameterization for synthesizing descriptions in formal languages.

3.1 Universal Entity

We notice many entities to communicate with others have a form of a body and two bi-directional ports. We call it the universal entity with a pair of bi-directional ports. A PE is connected with a user and a medium, and a medium is connected with left and right PEs, via their pairs of ports. Furthermore the service which is constructed with two PEs and Med is also regarded as a univeral entity connected with left and right users via its ports (Figure 1).

This view is very much beneficial, according to which we can treat various entities uniformly. For exmple, if we have a test system for the universal entities in a formal language, it can be used in testing protocol entities and also in testing the service. The test system in OBJ presented in the next section is originally made for the universal entities and this paper shows an application of its to interoperability testing.

3.2 Parameterization in FDTs

In most formal languages including programming languages, the concept of the parameterization is adopted, implicitly or explicitly. This is a programming technique to save text and thought by abstraction .

A parameterized description is a framework-like one having a body which is previously described with formal parameters. Instantiating a parameterized description with actual parameters, we can obtain a rather complicated description easily. It is important that the text required for instantiation is very shorter than the resultant description.

In our formal method for protocol development, the parameterized description is invented to describe the service behavior whose parameters are the left and right PEs and the communication medium. Provided that the parameterized description was previously prepared, specifiers have to write only one sentence to instantiate it with actual parameters of given protocols and medium in order to obtain the description of the behavior of the service. It enables us to realize interoperability testing on formal descriptions very economically.

4 Test System in OBJ

4.1 In What an FDT?

By the discussions in the preceding sections, the base language for an interoperability test system are required to satisfy the conditions:

- to have parameterization facilities,
- to be able "to be executed".

We have not chosen any among FDTs developed for protocols but adopted an algebraic specification language OBJ[2, 3, 4] by customizing it to describe distributed systems.

In OBJ every concept in a problem domain must be expressed uniformly in the form of a data type (set of values) and operators on them, and consequently the language is purely functional, which is different from the so-called FDTs. It may seem very restrictive and not capable of describing communication protocols. OBJ, however, indeed can express dynamic behaviors of protocols and furthermore has advantages to support protocol development with its several features, such as the simple but universal language constructs, modularity and hierarchical structures, parameterization, and clear semantics.

4.2 Algebraic Language OBJ

OBJ[2, 3, 4] is a specification and/or programming language based on the algebraic specification technique for abstract data types. An OBJ text is a sequence of *modules* , most typically *objects* . Each object defines one or more new *sorts* of data and *operators* on them and may refer to some preceding objects. An object's syntax, therefore, consists of declarations of

1) its name,

2) objects referred to,

3) its new sorts and operators, and

4) equations which give semantics to operators.

OBJ provides the parameterization mechanism which facilitates to construct hierarchical specifications in a disciplined way[3]. The objects TRIV, SEQ and SEqInt below illustrate parameterization. The parameterized object SEQ[X] defines the sequences of sort Elt of object X, which would be designated by an actual parameter. The theory TRIV prescribes the conditions for the parameter X. Object SEqInt is an instantiation of SEQ[X] by object INT(eger) and defines the sequences of integers.

```
th TRIV is    sort Elt .   endth

obj SEQ[X :: TRIV] is            --- sequence of X
   sort Seq .   pr BOOL .
   op . : -> Seq .                --- nil sequence
   op (_ _) : Elt Seq -> Seq .      --- cons op.
   op _$_ : Seq Seq -> Seq .         --- concat.
   op _#_ : Seq Elt -> Seq .      --- Elt to end
   vars S S' : Seq .   var E : Elt .
   eq : (. $ S) = S .
   eq : (E S) $ S' = E (S $ S') .
   eq : S # E = S $ (E .) .
endo

obj SEqInt  is   pr SEQ[INT] .   endo
                            --- sequence of integers
```

Besides its declarative semantics of the *Initial algebra* semantics, OBJ has also an operational semantics based on *rewrite rules*. The OBJ executor reduces a given term into its normal form by interpreting equations as left-to-right rewrite rules. In the above description, for example, the term ((1 .) $ (2 3 .)) in SEqInt will be reduced to (1 (2 3 .)).

How to describe distributed systems in OBJ is an interesting problem and would be various for specifiers. Our description technique to specify dynamic behaviors of distributed systems in OBJ cannot explained here because of lack of space but done in detail in [11].

4.3 Interoperability Test System

This subsection is the prime part of the paper which shows an interoperability test system in OBJ by using the OBJ's parameterization mechanism.

In the following, explanations for many subordinate modules are omitted.

4.3.1 Structure of the Description

Object **LAYER** is a parameterized object whose parameters are two PEs and a medium. Theories **PE** and **MEDIUM** defines necessary conditions for actual parameters as a PE or a medium. The relationship among the modules is shown in Figure 2.

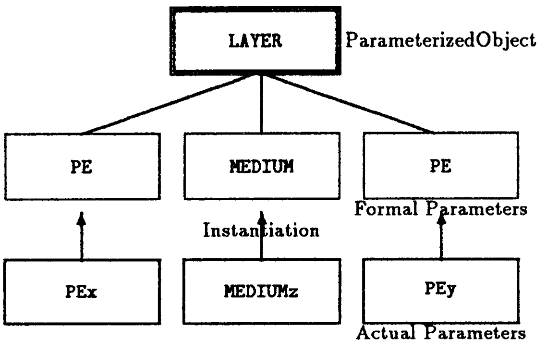


Figure 2: Relationship among Modules for **LAYER**

Similarly, object **SYSTEM** is parameterized, whose parameters are two users and a layer. Theories **USER** and **LAYER-TH** prescribe parameters. Object **SYSTEM** has been introduced in order to give definitions of behaviors of the layer, so that the two users here are objects only holding input and output sequences.

After completing this framework-like description, we provide **LAYER** with PEs' and medium's descriptions of a protocol in question, which results a layer's description. The constructed description for a layer is provided for **SYSTEM** as a parameter. An example of the instantiations for a protocol is shown at the last of this subsection.

4.3.2 Layer as a Parameterized Object

The parameterized object **LAYER** forms a big portion of the description. Its text has about 200 lines. Using constructs enumerated in theories **PE** and **MEDIUM**, dynamic behaviors among them occurred within the layer are completely described although we don't know concretely what the actual parameters for **PE** and **MEDIUM** are. In this sense, theories and their constructs play their roles of interfaces between the abstract, i.e., theories, and the concrete, i.e., actual parameters, very well.

Because **LAYER**'s behaviors have been generally defined, we need not consider specific protocols in each protocol development. It suffices only to provide **LAYER** with the specifications in question for PEs and a medium. The instantiation of **LAYER** will automatically represent the expected service. This is the primary advantage of parameterization, or generally, abstraction.

```
obj LAYER [ X Y :: PE , Z :: MEDIUM ] is
            --- parameterized object for layer
   sort Layer .   pr TIME .
   ..................
   op init-layer : -> Layer .   --- init. layer
   op layer : Pe.X Pe.Y Medium Time -> Layer .
                            --- to construct layer
   op l-transit : Layer -> Layer .
                            --- to make transition
   op l-transit1      : Time Layer -> Layer .
   ..................
   op data-to-pe1 :
           InputFromUser Time Layer -> Layer .
   op data-to-pe2 :
           InputFromUser Time Layer -> Layer .
   ..................
   eq l-transit(layer(PE1, PE2, m(PT1, PT2), T))
     = if    (fin (PE1) and fin (PE2)
                  and fin (PT1) and fin (PT2))
       then layer (PE1, PE2, m (PT1, PT2), T)
       else l-transit1 (
              proceed-time-in-layer (
                out-to-m-time (PE1),
                out-to-m-time (PE2),
                out-time (PT1),
                out-time (PT2), T),
              layer(PE1,PE2,m(PT1,PT2),T)) fi .
   eq l-transit1
         (T', layer(PE1, PE2, m(PT1, PT2), T))
     = if  ... then  ... else  ... fi .
   ...
endo
```

Sort **Layer** has a constructor operator **layer** whose arguments are **Pe.X**, **Pe.Y**, **Medium**, and **Time**. The operator **init-layer** designates one of the terms of **Layer** as an initial one, which is **layer(init-pe, init-pe, init-med, 0)** by its equation.

4.3.3 Whole System

Object **SYSTEM** is defined in the same manner as **LAYER**, i.e., it is a parameterized object having three parameters: the left and right users and the layer. Theories **USER** and **LAYER-TH** are defined to specifying conditions for parameters.

```
obj  SYSTEM [ A B :: USER , C :: LAYER-TH ] is
```

```
                         --- object for system
  sort SysLog .    sort System .    sort Log .
  op  sys : User.A User.B Layer Time -> System .
                          -- to construct system
  op  slg : System Log -> SysLog .
                         --- to construct syslog
  op  transit   : System -> System .
                     --- to make system proceed
  op  s-transit   : SysLog -> SysLog .
                     --- to make syslog proceed
   ......
endo
```

The important sort in this object is System which is a set of the systems' configurations. Its terms have the form of sys(left-user, right-user, layer, time). Operator transit makes a system proceed to its final state as;

```
  transit(sys(lu, ru, l, t))
  ===>    sys(lu', ru', l', t')) .
```

Auxiliary sorts and operators are also declared mainly to facilitate logs. Sort Log is a record of changes of system's configurations and SysLog is a pair of sorts System and Log. It is necessary to keep histories of transitions by logs to analyze behaviors of layers.

Object System represents the whole parameterized description which is generally applicable to any protocol in any layer. To obtain an actual description for a particular protocol, there remains the following steps:

1. to obtain, or describe, objects PEx and MEDIUMx for the protocol and medium specifications which satisfy the conditions by theories PE and MEDIUM,
2. to instantiate LAYER with PEx and MEDIUMx and name it LAYERx,
3. to obtain objects USERx and USERy for the users which satisfy the conditions by theory USER, and
4. to instantiate SYSTEM with USERx, USERy and LAYERx and name it SYSTEMx.

4.3.4 Instantiations with Actual Protocols

The last steps of description are to instantiate LAYER with actual protocols and medium and hence to instantiate SYSTEM with actualized LAYER. We here show such a procedure done with the sample protocol so called the sliding window protocol (SWP)[9]. Its protocols and medium have been described as TRANSMITTER, RECEIVER, and SWP-MEDIUM in OBJ.

```
make  SWP-LAYER  is
   LAYER [ view to TRANSMITTER is
       sort  Pe  to  Transmitter .  ...  endv,
           view to RECEIVER is
       sort  Pe  to  Receiver .     ...  endv,
           view  to  SWP-MEDIUM is
       sort  Medium to SwpMedium .   ...  endv ]
endm

make  SWP-SYSTEM  is
   SYSTEM [ view to TRANSMITTER-USER  is
       sort User to TransmitterUser . ...  endv,
            view  to  RECEIVER-USER  is
       sort User to ReceiverUser . ...  endv,
            view  to  SWP-LAYER  is
       sort Layer to Layer .         ...  endv ]
endm
```

An OBJ construct view makes sorts and operators in theories correspond to those in actual parameters.

Object SWP-SYSTEM, the result of these two consecutive instantiations, is the description for the SWP service which protocol specifications TRANSMITTER and RECEIVER and medium description SWP-MEDIUM realize in combination.

It should be noted that if

- we had already had the OBJ parameterized description whose top object is SYSTEM, and
- OBJ descriptions TRANSMITTER, RECEIVER, and SWP-MEDIUM for the SWP protocols and medium had been supplied independently,

only two make sentences shown here are required to write in order to obtain the interoperability test system for the SWP protocol, which is very much less costly.

4.4 Execution and Its Results

As briefly mentioned in section 4.2, the OBJ system has a facility (the executor) to reduce a given term into its normal form by interpreting equations as left-to-right rewrite rules.

In the case of SWP, sort SysLog in object SWP-SYSTEM represents the whole system configurations and operator

```
    s-transit  : SysLog -> SysLog
```

proceeds a SysLog to a stable one and yield it as its answer. This means that, letting the executor to reduce term s-transit(SysLog) for a certain SysLog, we can simulate how the SWP system goes on. By investigating the result of reduction which contains system Logs, we can know many things about both the functions of the SWP service and its internal behaviors.

```
reduce in SWP-SYSTEM :
   s-transit(
      slg(sys(t-user(
         Tinit ; is(in-u((ut('data1')),1))  ;
                 is(in-u((ut('data2')),3))  ;
                 is(in-u((ut('data3')),5))  ;
                 ....
```

```
                is(in-u((ut('data10')),19))),
             r-user(Rinit), init-layer, 1),
         log))

result SysLog:
   slg(sys(t-user(Tinit),
          r-user(Rinit ;
             os(out-u((ur('data1')),10)) ;
             os(out-u((ur('data2')),12)) ;
             os(out-u((ur('data3')),34)) ;
             ........
             os(out-u((ur('data10'),250))),
          layer( ......
                258)),
   ...
   l(sys(t-user(Tinit ; ......
   ...
```

This is the output of an execution in SWP for a sample data. We can know that transmission of ten data was successfully terminated at time 258. Besides the functions of the service, performances can be also examined by checking times when data emitted and received, execution terminated, and so on.

Furthermore, iterating performance testings with protocol parameters changed, experiments are possible to approximate their desirable values. Because the window size of the transmitter and the time length for timeouts for data re-transmission are important parameters in SWP, we executed testing for various values for them.

Although interoperability testing on formal descriptions shown in this section may not be able to replace conventional in the actual environments, we are convinced that it is a very strong conceptual tool for protocol development, especially in the design stage.

5 Concluding Remarks

We have introduced the idea of interoperability testing on formal descriptions. It can be realized by synthesizing the description for the service from the protocol specification at a lower cost. A test system has been developed by the auther in the OBJ language.

In the experiments in the sliding window protocol (SWP), the author has made various modifications on SWP and examined them. In the original and simplest description, the parameterized one for the whole system is of length of 1089 lines, the actual parameters for SWP's PEs and medium of 1542 lines, and instantiation of 116 lines. [1] Only the text for the instantiation is necessary to be written for starting interoperability testing upon each protocol of interest.

This systematic method to obtain service descriptions and to realize test systems is expected to reduce protocol development cost greatly.

[1]They contain many empty lines for readability

This paper has presented a way to easily synthesize the description for the service. Protocol verification should be managed to prove that such a service description satisfies *all requirements* stated in the service definition. This is a more lucid explanation for protocol verification than conventional ones based on natural language statements. It is a future work of ours to establish protocol verification concepts and methodology in a formal method.

Acknowledgements

This work was supported by the national R&D program "Interoperable Database Systems".

References

[1] N. Arakawa, M. Phalippou, N. Risser, T. Soneoka, "Combination of Conformance and Interoperability Testing," Proc. of Fifth International Conference on Formal Description Techniques, (FORTE'92) pp. 389-419, 1992.

[2] K. Futatsugi, J. A. Goguen, J.-P. Jouannaud, J. Meseguer, "Principles of OBJ2," Proc. of 1985 Symposium of Principles of Programming Languages, ACM, pp. 52-66, 1985.

[3] K. Futatsugi, J. A. Goguen, J. Meseguer, K. Okada, "Parameterized Programming in OBJ2," Proc. of 9th International Conference on Software Engineering, IEEE pp. 51-60, 1987.

[4] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, J.-P. Jouannaud, "Introducing OBJ," Technical Report SRI-CSL-92-03, SRI International, Computer Science Lab, 1992, To appear in J. A. Goguen, editor, Applications of Algebraic Specification Using OBJ, Cambridge University Press.

[5] G. J. Holzmann, "Design and Validation of Computer Protocols," Prentice Hall, International Edition, 1991.

[6] ISO/IEC, "Information Processing System - Open Systems Interconnection - Estelle - A Formal Description Technique Based on an Extended State Transition Model," ISO 9074, 1989.

[7] ISO/IEC, "Information Processing System - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," ISO/IEC 8807, 1989.

[8] ISO/IEC, "Information Processing System - Open Systems Interconnection - Guidelines for the Application of Estelle, LOTOS, and SDL," ISO/IEC TR 10167, 1992.

[9] R. E. Miller, "Protocol Verification: The First Ten Years, the Next Ten Years; Some Personal Observations," Proc. of Protocol Specification, Testing and Verification, 1990.

[10] K. Okada, K. Futatsugi, "Supporting the Formal Description Process for Communication Protocols by an Algebraic Specification Language OBJ2," Proc. of 2nd Int'l. Symposium on Interoperable Information Systems, INTAP, pp. 127-134, 1988.

[11] O. Rafiq, R. Castanet, "From Conformance Testing to Interoperability Testing," Third International Workshop on Protocol Test Systems, p. 15, 1990 .