Modeling and Interoperability Test Case Generation of a Real-Time QoS Monitoring Protocol

Byoung-Moon Chin[a)], Sung-Un Kim, Sung-Won Kang, and Chee-Hang Park

QoS monitoring is a kind of real-time systems which allows each level of the system to track the ongoing QoS levels achieved by the lower network layers. For these systems, real-time communication between corresponding transport protocol objects is essential for their correct behavior. When two or more entities are employed to perform a certain task as in the case of communication protocols, the capability to do so is called interoperability and considered as the essential aspect of correctness of communication systems. This paper describes a formal approach on modeling and interoperability test case generation of a real-time QoS monitoring protocol. For this, we specify the behavior of flow monitoring of transport layer QoS protocol, i.e., METS protocol, which is proposed to address QoS from an end-to-end's point of view, based on QoS architecture model which includes ATM network in lower layers. We use a real-time Input/Output Finite State Machine to model the behavior of real-time flow monitoring over time. From the modeled real-time I/OFSM, we generate interoperability test cases to check the correctness of METS protocol's flow monitoring behaviors for two end systems. A new approach to efficient interoperability testing is described and the method of interoperability test cases generation is shown with the example of METS protocol's flow monitoring. The current TTCN is not appropriate for testing real-time and multimedia systems. Because test events in TTCN are for message-based system and not for stream-based systems, the real-time in TTCN can only be approximated. This paper also proposes the notation of real-time Abstract Test Suite by means of real-time extension of TTCN. This approach gives the advantages that only a few syntactical changes are necessary, and TTCN and real-time TTCN are compatible. This formal approach on interoperability testing can be applied to the real-time protocols related to IMT-2000, B-ISDN and real-time systems.

Manuscript received April 8; revised September 16, 1999.

[a)] Electronic mail: bmchin@pec.etri.re.kr

I. INTRODUCTION

Over the past several years there has been a considerable amount of research with the field of Quality-of-Service (QoS) support for distributed multimedia systems. To date, most of the work has been within the context of individual architectural layers such as the distributed system platform, operating system, transport subsystem and network layers [1], [2].

Much less progress has been made in addressing the issue of overall end-to-end support for multimedia communications. In recognition of this, a number of research teams have proposed the development of QoS architectures which incorporate QoS configurable interfaces and QoS driven control and management mechanisms across all architectural layers [2]. As illustrated in Fig. 1, this generally requires end-to-end admission testing and resource reservation in the first instance, followed by careful coordination of disk and thread scheduling in the end-system, packet/cell scheduling and flow control in the network and, finally, active monitoring and maintenance of the delivered QoS.

QoS monitoring allows each level of the system to track the ongoing QoS levels achieved by the lower network layers. QoS monitoring often plays an integral part in a QoS maintenance feedback loop that maintains the QoS achieved by resource modules [2]. QoS monitoring is a kind of real-time systems which stem from the use of computers for controlling physical processes. For these systems, real-time communication between corresponding transport protocol objects is essential for their correct behavior.

When more than one object are employed to perform a certain function, there arises the problem of whether they together behave correctly. This is the problem of interoperation in a general sense and thus it can be said that two or more objects interoperate if they together behave as expected in specifications. Usually the expectations about interoperation should be inferred or derived from the relevant specifications [3], [4]. Within the context of communication networks, an object mentioned above can be a network node, a layer of a node or a component of a layer or a plane or even a network, i.e. anything we decide to view as a whole.

An abstract view of communication can be conceived when protocols of network nodes are layered and underlying layers are regarded as the service provider for the layer above. Then by similarly abstracting from the underlying layers, we can focus on the behavior of a certain layer and think about interoperation of the objects which realize the particular layer under consideration. Once the notion of interoperation is clearly understood, there arises the problem of verifying interoperation for target implementations which interact with each other. This is the task of interoperability testing.

Work on interoperability testing can be classified into two categories depending on whether it is more geared to practical things such as clarification and implications of interoperability testing or to systematic generation of interoperability test suite. As work along the former line are [5]–[8]. [5], [7] present interoperability testing experiences. [6] gives a comprehensive discussion on various aspects related to interoperability testing.

For the latter line of work, there are [9]–[14]. All these base interoperability test suite derivation on some sort of reachability analysis. For interoperability test architecture, [9] uses upper testers as well as lower testers. [10], [13] introduces notions of stable state to reduce the size of relevant state space. [12] develops interoperability test suite method for synchronous models. [14] shows how to derive test suites for dynamic testing of interoperability.

The previous work, however, did not provide a coherent framework for interoperability testing in that the notions of interoperability, interoperability testing, interoperability test case and interoperability test architecture were not presented in an integrated manner nor were interrelated for the purpose of interoperability test suite development.

This paper, which belongs to the second line of work on interoperability testing, addresses these issues. Starting from the general definition of interoperability, we carefully select an interoperability test architecture. The chosen architecture, combined with natural assumptions and inherent limitations for testing, is shown to induce a notion of interoperability test case. And this notion of interoperability test case allows us to focus on genuine interoperability aspect and at the same time to derive interoperability test suite in a cost-effective manner.

Formal protocol test generation becomes an indispensable part of protocol specification and implementation. Formal test generation often requires a formal specification of the protocols, yet a large number of the protocols in practice are specified informally, mostly in natural language [3], [4].

For this, as a case study, we specify formally the behavior of flow monitoring of transport layer QoS protocol, i.e., multimedia enhanced transport service (METS) protocol, which is proposed to address QoS from an end-to-end's point of view, based on QoS architecture model which includes ATM network in lower layers [19]. We use a real-time I/OFSM to model the behavior of real-time flow monitoring over time. As a formal method, real-time I/OFSM treats QoS monitoring components as mathematical objects and provide mathematical models to describe and predict the observable properties and behaviors of these objects. It includes syntax for describing models, semantics and meaning relations.

From the modeled real-time I/OFSM, we generate interoperability test cases to check the correctness of METS protocol's flow monitoring behaviors for two end systems. A new approach to efficient interoperability testing is described and the method of interoperability test cases generation is shown with the example of METS protocol's flow monitoring.

This paper also presents the translation of the test cases obtained by our generation algorithm to real-time tree and tabular canonical notation (TTCN). The current TTCN is not appropriate for testing real-time and multimedia systems, because test events in TTCN are for message-based system and not for stream -based systems. And also, real-time can be approximated in TTCN. In this paper we propose the notation of real-time abstract test suite (ATS) by means of real-time extension of TTCN.

This paper is organized as follows: in Section II, we present QoS architecture proposed by Campbell [2], a QoS architecture based on ATM network and METS protocol mechanism. In Section III, we define a real-time I/OFSM and explain a real-time modeling of METS protocol by using message sequence chart (MSC). Following this, we give the description of the method in Section IV, to generate interoperability test cases from an intermediate real-time reference model equivalent to the corresponding behavior of flow monitoring over time. We then present the translation of the obtained test cases to real-time TTCN form in Section V. Finally, in Section VI we conclude this paper.

II. BASIC MODEL OF QOS ARCHITECTURE

1. Quality-of-Service Architecture

The QoS-A is layered structure of service mechanism for

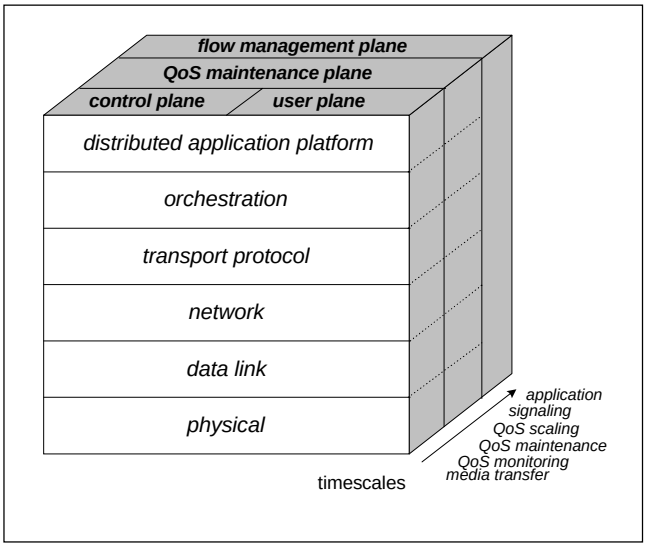

Fig. 1. QoS architecture.

QoS management and control of continuous media flow in multimedia network [20]. The QoS architecture presented in Fig. 1 is composed of a number of layers and planes. The upper layer consists of a distributed applications platform augmented with services to provide multimedia communications and QoS specifications in an object-based environment. Below the platform level is an orchestration layer that provides multimedia synchronization services across multiple related application flows.

Supporting this is a transport layer that contains a range of QoS configurable services and mechanisms. Below this, an internetworking layer and lower layers form the basis for end-to-end QoS support.

QoS management is realized in three vertical planes in the QoS architecture. The protocol plane consists of distinct user and control sub-planes. Separate protocol profiles are used for the control and data components of flows because of the different QoS requirement of the control and data: control generally requires a low latency full duplex assured, high throughput and low latency simplex services.

The QoS maintenance plane contains a number of layer specific QoS managers. These are each responsible for the fine-grained monitoring and maintenance of their associated protocol entities. Based on flow monitoring information and a user supplied service contract, QoS managers maintain the level of QoS in the managed flow by means of fine-grained resource tuning strategies.

The final QoS-A plane pertains to flow management, which is responsible for *flow establishment* (including QoS based routing, end-to-end admission control and resource reservation), *QoS mapping* (which translates QoS representations between layers) and *QoS scaling* (which correctly describes *QoS adaptation* and *QoS filtering* for coarse grained QoS management)

2. QoS Architecture Based on ATM Network

In this section QoS architecture model which includes ATM network in lower layers is illustrated. Figure 2 shows QoS architecture model based on ATM network [14].

Note that Fig. 2 only shows the transport layer and below because the scope of this paper for real-time modeling is to implement transport layer of QoS architecture model. The protocol plane of transport layer uses METS protocol [19]. The physical

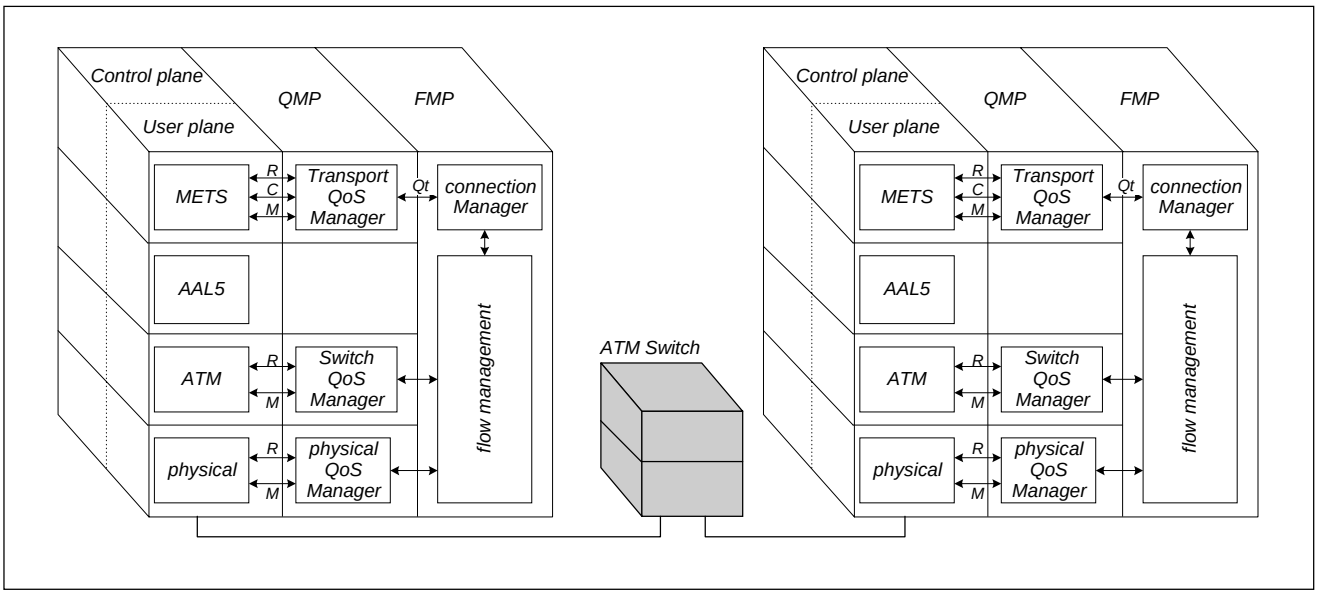

Fig. 2. QoS architecture model based on ATM.

and ATM layer are based on ATM switching, the primary functions of ATM adaptation layer (AAL) are to segment IP packets into 53 bytes ATM cells and to reassemble ATM cells into IP packets. The QoS maintenance plane (QMP) is composed of a transport QoS manager responsible for managing QoS of on-going flows, application layer interact with flow management plane (FMP) for establishment and dynamic QoS management of multimedia flows.

At each layer in Fig. 2, the various mechanism in each plane well defined interfaces to their peer. At the transport layer the support of QoS is dependent on interactions between the transport protocol, transport QoS manager, the flow management plane and AAL.

The QoS-A defines the following internal interfaces between the three planes at the transport layer and below as illustrated in Fig. 2:

- R (resource control) interface used to allocate, tune and release transport protocol resources, and alert the QoS management plane if protocol resources are short. It contains the following primitives: alloc_resource, tune_resource, resource _alert and free_resource;
- M (monitor) interface used by the transport QoS manager to configure and control monitoring of flows in the transport protocol and to receive reports of actually achieved QoS performance over a preceding interval. It contains the following primitives: start_monitor, set_rate, qos_assessment and stop_ monitor;
- C (control) interface used by the QoS manager to set, modify and read internal transport protocol states during flow connection, data transfer and re-negotiation. The interface contains the following primitives: set_state and report_state;
- Qt (maintenance) interface is supported by the transport QoS manager and used by the flow management plane. It contains the following primitives: start_maintenance, set_attributes, free _attributes, assessment, qos_alert and stop_maintenance;

3. METS Protocol Mechanism

The METS is a transport protocol and provides an ordered but non-assured, connection oriented transport communication service. It allows the user to negotiate QoS parameters such as bandwidth, jitter, delay and the tightness of synchronization between multiple related connections [14].

It is the responsibility of the protocol to share communications resources in end-to-end systems among flows with widely diverse QoS requirements. To meet this need, the METS protocol mechanism is composed of flow monitor, buffer manager, flow regulator and flow scheduler. Also included is a resource manager responsible for overseeing the allocation and adaptation of the various protocol resources.

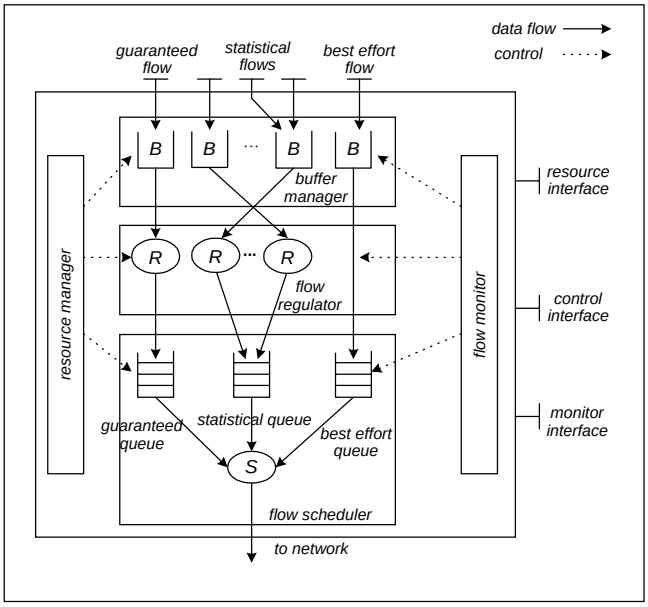

Fig. 3. METS protocol mechanism and interface.

Figure 3 describes the mechanism of METS protocol and interface [14]. Table 1 presents the performance parameters together with the mechanisms required for their support and the resource configuration required for the three types of service commitment. Flow monitoring is a core mechanism in the QoS-A. The transport protocol flow monitoring component gathers statistical information regarding the ongoing flow of media, both at the source and the sink of the transport flow.

The information given in Table 1 is used by the transport QoS manager during the on-going QoS maintenance of flows. Based on a flow's measured performance the transport QoS manager and transport protocol interact over the resource, control and monitor interfaces to maintain the flow's QoS [15].

III. REAL-TIME MODELING

Formal methods treat system components as mathematical objects and provide mathematical models to describe and predict the observable properties and behaviors of these objects [3], [4]. It includes syntax for describing models, semantics and meaning relations. There are several advantages to use formal methods for the specification and analysis of real-time systems [16], [17]. These are:

- the early discovery of ambiguities, inconsistencies and incompleteness in informal requirements.
- the automatic or machine-assisted analysis of the correctness of specifications with respect to requirements.
- the evaluation of design alternatives without expensive prototyping.

Table 1. Mechanism and performance parameter for flow type.

		QoS Commitment		
QoS parameter	**QoS mechanism**	**Guaranteed**	**Statistical**	**Best effort**
Loss	Buffer Management	Fixed buffer Allocation based on the peak rate	Shared buffers based on average rate	No guaranteed buffer allocation
Throughput	Regulation (flow shaping)	Eligibility time based on peak rate	Eligibility time based on average rate	No regulation resources committed
Delay and Jitter	scheduling	Flow always Scheduled at eligibility time	Flow scheduled at eligibility time resource permitting	Flow scheduled if scheduler idle

In formal methods, we can categorize three following sets:

- Logic based: High Order Logic, Temporal Logic, Z, VDM, Real-Time Logic, Timed Temporal Logic, Metric Temporal Logic;
- State/Net based: Statecharts, Modechart, Petri-Net, Timed-Petri-Net, I/OFSM, Communicating State Machine, Timed I/OFSM;
- Process Algebra based: CSP, CCS, Timed CSP, ACP, ATP, ACSR.

In this paper, we use a real-time I/OFSM to model the behaviors of real-time flow monitoring over time.

1. Real-Time I/OFSM

We define real-time I/OFSM (Timed I/OFSM) to model the behavior of real-time systems over time [16]. A timed I/OFSM is an extension of a finite I/OFSM with a finite set of real-valued clock variables.

Definition 1 A real-time I/OFSM is a 5-tuple <L, S, S_0, C, Tr> where:

① L = {m_0, ..., m_n} is a finite message.
② S = {s_0, ..., s_m} is a finite set of states.
③ $S_0 \subseteq$ S is a finite set of initial states.
④ C = {c_1, ..., c_p} is a finite set of clocks, and
⑤ $\mathrm{Tr} \subseteq \mathrm{S} \times \mathrm{S} \times \mathrm{L} \times 2^{c} \times \Phi(C)$ is the set of transitions. An edge $< s, s', a, \lambda, \delta >$ represents a transition from state s to state s' on input message a. The set $\lambda \subseteq C$ gives the clocks to be reset with this transition, and δ is a clock constraint over C.

Definition 2 Characterization of Clock

① The domain of clocks is the set of real numbers.
② The initial values for all clocks are zero.
③ The values of all clocks increase at the same rate.
④ Any subset of them can be reset to zero when a transition is executed

Definition 3 Communicating system $\sum$ of n real-time I/OFSM's is $< \{(\mathrm{M_i}, \mathrm{Q_i}) | 1 \le i \le n\}, L_{\Sigma,in}, L_{\Sigma,out}, s_{\Sigma,0} >$ where:

① $\mathrm{M_i}$, $1 \le i \le n$, is a real-time I/OFSM <L, S, S_0, C, Tr> as in Definition 1.
② $\mathrm{Q_i}$, $1 \le i \le n$, is an input queue for $\mathrm{M_i}$.
③ $L_{\Sigma,in}$ is a set of external input symbols.
④ $L_{\Sigma,out}$ is a set of external output symbols.
⑤ The initial state of the system is $s_{\Sigma,0}$ = <($s_{1,0}$, Q_0), ..., ($s_{n,0}$,Q_n)> where $\mathrm{Q_i}$ is empty and $s_{i,0}$ is the initial state of $\mathrm{M_i}$, $1 \le i \le n$.

In this model, there is one explicitly defined input queue for each real-time I/OFSM and implicit bi-directional communication channels between each pair of real-time I/OFSM's. A system state in which input queues are all empty is called a *stable state* [12]. The communicating system as defined above takes a sequence of inputs from the environment one by one. The next input from the environment is processed only when the system is in a stable state. This definition of stable state makes it unnecessary in later sections to describe input queues to show stable states of a communicating system.

2. Real-Time Modeling of METS Protocol

We specify the behavior of flow monitor using MSC which is one of formal methods, as in Fig. 4. The current version of MSC does not handle real-time constraints. In this paper MSC is used as a complementary formalism to show a first design of the protocol message exchanges, and real-time constraints are modeled using real-time I/OFSMs.

In Fig. 4, flow monitoring is initiated when a start_monitor_req command is received by the transport protocol from the transport QoS manager at the protocol's monitor interface. When monitoring is enabled, the monitor operates in one of the two defined modes; the mode is implicitly based on the direction of the flow. In essence, the transport protocol monitors a flow's on-going performance and the transport QoS manager maintains it. Table 2 identifies the states of flow monitoring in METS protocol, input and output messages of flow monitoring are given in Fig. 4, and timers and counters are represented in Table 3 respectively.

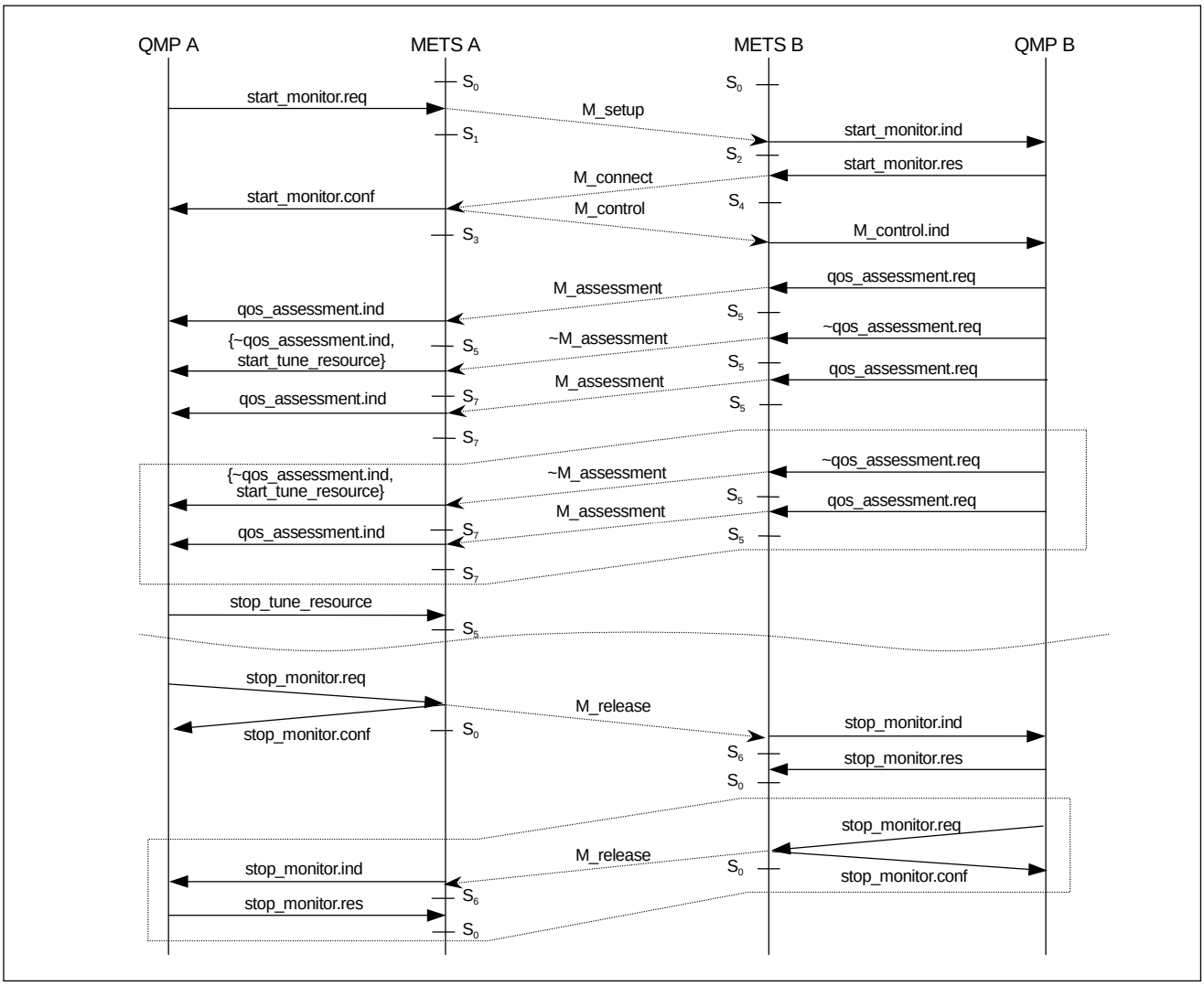

Fig. 4. MSC of flow monitor.

Table 2. State of flow monitor.

state	Meaning
S_0	NULL
S_1	Monitoring initiated
S_2	Monitoring present
S_3	First assessment waiting
S_4	Monitoring request
S_5	Active
S_6	Stop request
S_7	Adjustment

Table 3. Timer and counter of flow monitor.

variable	Meaning
α	Transmission time of unidirectional message
β	Maximum processing time of message
γ	Maximum transmission time of one message
D_{max}	Maximum return transmission time
TC	Timeout counter (maximum value: TC_{max})
VC	Violation counter (maximum value: VC_{max})
CC	Conformance counter (maximum value: CC_{max})

Considering all the points described above, Real-time reference I/OFSM model of flow monitor is generated as in Fig. 5. According to MSC in Fig. 4, the flow monitor using the timers and counters in Table 3 behaves as follows:

- In order to establish the flow monitoring, METS protocol receives the message start_monitor.req from the QoS manager via the interface M.

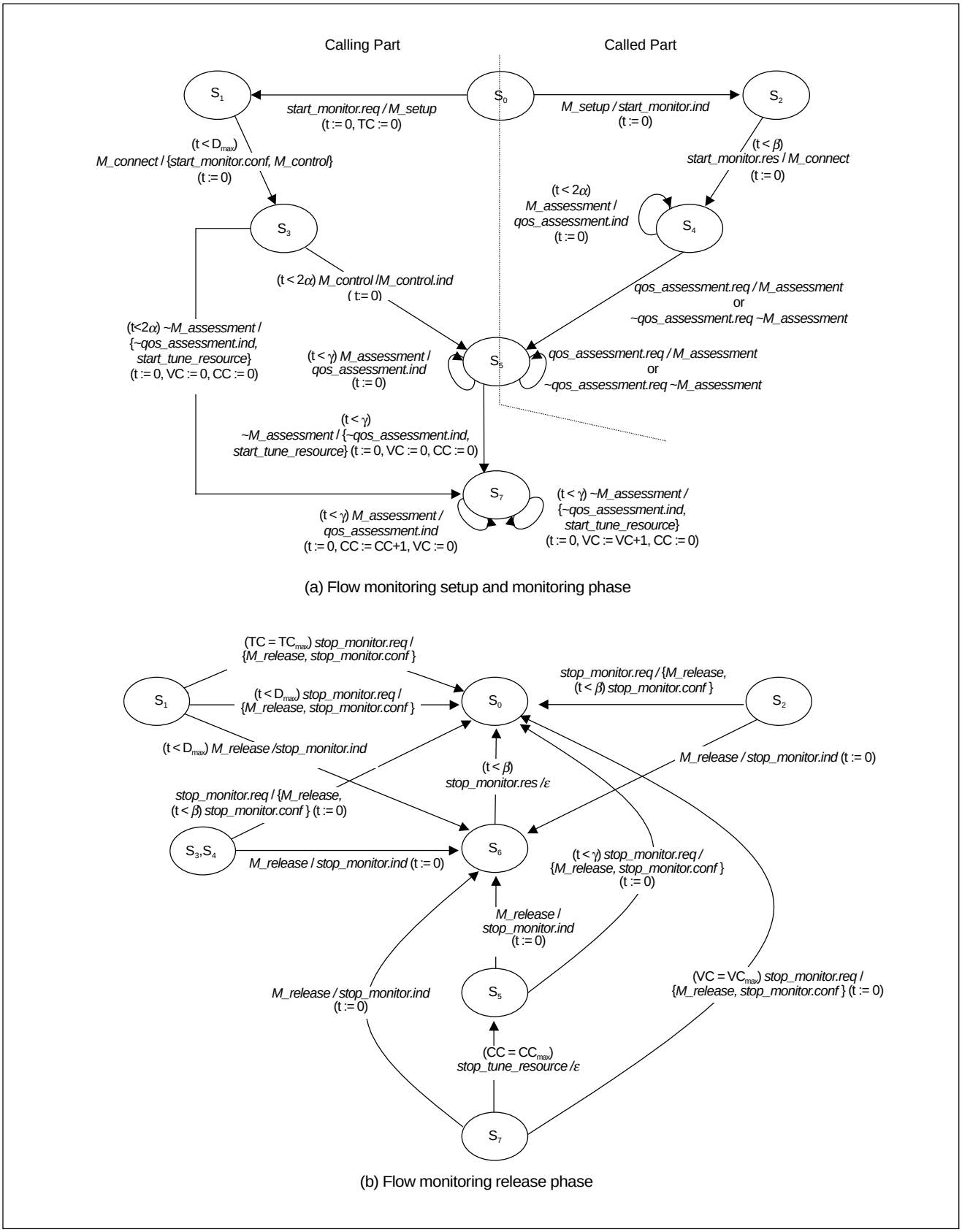

Fig. 5. Real-time I/OFSM of flow monitor.

- After transmission of the METS M_setup message to the remote METS protocol (timer starts with a reset to $t = 0$), METS protocol should receive M_connect message within the maximum return transmission time D_{max} from the remote METS protocol.
- If the confirmation message M_connect for start_monitor has not reached the METS protocol from the remote counterpart within the time limit D_{max}, the timeout counter TC is increased. When TC reached the maximum TC_{max}, it is assumed that flow monitoring cannot be established, and METS protocol sends the message stop_monitor.req (M_release).
- When M_connect arrives with the time limit as usual, METS protocol notifies the setup of flow monitoring to the transport QoS manager, and sends M_control to the remote METS protocol (starts with $t = 0$) which orders the remote METS protocol to start the transmission of assessment message.
- After the setup of flow monitoring and the receipt of M_control informing the transmission of the assessment message, the remote transport QoS manager sends qos_assessment containing statistical information about the current media flow with a period $t < \gamma$. When the first qos_assessment arrives in time $t < 2\alpha$ from the remote QoS manager, METS protocol measures the receiver side QoS considering the sender's QoS in the qos_assessment message.
- If the first assessment message is qos_assessment meaning no problem in QoS rather than ~qos_assessment implying the existence of a QoS problem, METS protocol sends transport QoS manager a request start_tune_resource for starting tune_resource via R interface.
- If the second and the following messages are qos_assessment, the conformance counter CC is increased by one. When CC is equal to CC_{max}, METS protocol sends transport QoS manager a request stop_tune_resource for freeing the tune_resource. If, however, ~qos_assessment messages are received instead of qos_assessment consecutively with a period of $t < \gamma$, the violation counter VC is increased. When VC amounts to VC_{max}, stop_monitor.req (M_release) is sent to the remote transport protocol requesting the end of flow monitoring.
- If a qos_assessment is received after a number of successive ~qos_assessment's, VC is reset to zero; similarly, CC is reset to zero upon receipt of a ~qos_assessment after a sequence of qos_assessment's.
- Transport QoS manager can send anytime stop_monitor message requesting the termination of flow monitoring. After a receipt of stop_monitor.req from the transport QoS manager, the transport protocol should transmit M_release to the remote transport protocol (starting with a reset $t = 0$), and send stop_monitor.conf to transport QoS manager with time $t < \beta$.

Figure 5-(a) represents the flow monitoring setup and monitoring phase and Fig. 5-(b) the flow monitoring release phase, respectively.

IV. GENERATION OF TEST CASE

1. Interoperability Test

Conformance testing checks correctness for a single communication protocol entity whereas interoperability testing checks correctness for a system of two or more communication protocol entities [3], [4] [18]–[20].

In our approach to interoperability test suite derivation, it is assumed that (1) the communication protocol is structured as a real-time I/OFSM and that (2) a complete set of conformance abstract test cases has been already developed (elsewhere) based on the a real-time I/OFSM structure. Because of this second assumption, our approach will yield an interoperability test suite which has no overlapping with conformance test suites as we will see later. Furthermore, (3) we adopt the test architecture in Fig. 6. We call an individual object involved in interoperation an implementation under test (IUT). In spite of the term IUT, it is important to note that the target of testing here is not the individual objects (which are the targets of conformance testing) but the system as a whole which consists of those objects. Still, IUT is a convenient term and will be used throughout the paper.

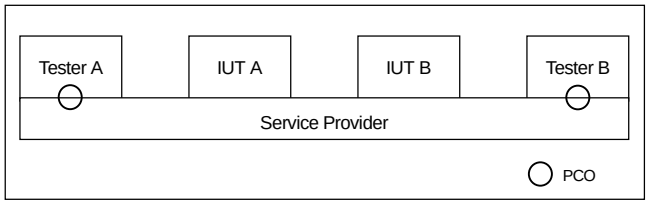

Fig. 6. Test architecture of interoperability testing.

Note that there are only two points of control and observation (PCOs) in Fig. 6. Often in practice a monitoring point or point of observation (PO) is set between IUTs. We do not set such PO. For with such PO, (1) it would be more costly to generate test suite, (2) it would be more costly to perform testing and (3) interoperability testing would have much overlapping with conformance testing.

As with conformance testing, interoperability testing is restricted by observability and controllability that is allowed in a chosen test architecture. In addition, it is usually assumed that the slowness of the environment [12] places practical limit on observability and controllability. That is, transient states cannot be observed or controlled in a predictable way and hence are considered useless for the purpose of testing. This justifies basing any notion of test case on stable states as we do in this paper.

```
Input: I/O FSM's S1 and S2
Output : Stable [j] contains all the stable states of S1 × S2
J := 0;
Stable[j] := {(s1.0, s2.0)}; /* All stable states generated up to Stage j */
New := {(s1.0, s2.0)};     /* Stable states to be expanded at Stage j+1 */

While New ≠ Ø do begin
   <New, (s1, s2)> := delete-one-element(New);
   OldFrontier := Ø;              /* Already expanded nodes at Stage j */
   NewFrontier := {(s1,s2)};      /* Nodes to be expanded at Stage j+1 */
   While (NewFrontier ≠ Ø) do begin
      <NewFrontier, (s1, s2)> := delete-one-element(NewFrontier);
      OldFrontier := OldFrontier ∪ {(s1,s2)};
      Event_Seq_Set := interaction-sequences ((s1,s2));
        While (Event_Seq_Set≠ Ø) do begin
          <Event_Seq_Set, Event_Seq> := delete-one element(Event_
                                        Seq_Set);
            if Event_Seq_Set ends with a stable state (s1,s2)
            then begin
                    if s'∉(Stable[j] ∪ New ∪ OldFrontier)
                    then NewFrontier := NewFrontier ∪ {(s1,s2)};
            End

            Else begin
                    There is an error in the specifications.
                    Stop and fix the error
            End
        End while;
   End while;
   j := j+1;
   Stable [j] := Stable[j-1] ∪ OldFrontier;
End while;
```

Fig. 7. Algorithm of generate stable states.

The interoperability test approach of this paper is to test interoperability of implementations against interoperability of specifications. In this approach, interoperability testing is a (yet another) conformance testing, i.e. conformance testing of the system which is composed of the actual subsystem implementations. Any notion of interoperation, if any, should be expressed in principle in a given specification or should be agreed upon in advance by specification writers and implementers. Ascertaining correctness of specifications in this respect is the task of verification and validation, which goes beyond the proper scope of testing. For the target of testing is the relation between specification and implementation rather than between specifications. However, if specific interoperability requirements are subject to validation and verification at the specifications level, then the same validation and verification method can be applied to a system of implementations for testing.

As the result of interoperability testing of implementations, problems residing in specifications may be found and reported. Then the additional ingredient interoperability testing provides in addition to verification of interoperability of specifications is to fill the gap left by usual conformance testing. The real work of interoperability assurance should come at the stage of specification development. But still the fine points that may have been missed at the time of specification writing (including validation and prototyping) can be augmented through testing.

2. Generation of Interoperability Test Case

A system state in which input queues are all empty is called a stable state [12]. The communicating system as defined in Definition 3 takes a sequence of inputs from the environment one by one. The next input from the environment is processed only when the system is in a stable state.

In interoperability testing, interoperability test suite is developed in such a way that for the given specification with real-time I/OFSM structure, absence of operation errors (or correctness of input/output behavior) and absence of transfer errors (correctness of the state reached after the transition) are examined with respect to each transition of the real-time I/OFSM [21].

We show below an algorithm to generate all stable states, which form the starting, and the ending states of interoperability test cases. Actual test cases generation is realized by decorating the algorithm with the step to store the applicable sequence of events between stable states [19]. In our approach to interoperability testing, the number of stable states is small enough that it can be used as a suitable basis for manual calculation as well as for automatic generation of interoperability test suites.

In Fig. 7, $s_{1.0}$ and $s_{2.0}$ denote the initial stable states of real-time I/OFSM S_1, S_2 respectively and (s_1, s_2) represents the global system state which s1 and s_2 are stable states of S_1 and S_2 respectively. Delete-one-element (set) is a function that chooses an arbitrary element from the set and removes it. Interaction-sequence $((s_1, s_2))$ generates all possible sequences of interactions that are initiated by the real-time I/OFSM that is in the state S_1. s' denotes the final stable state that became to the results of applying the (s_1, s_2).

We applied the proposed algorithm to the reference I/OFSM given in Fig. 5. The resulting application of the algorithm is shown in Fig. 8. The global states in boxes are stable states and those in parentheses are transient states. Interoperability test cases generation is realized by decorating the sequence of events between stable states.

In each stable state, the dotted vertical lines means that left and right edges of those lines denote interactions which are initiated by the left and right IUTs, respectively. The nodes of tree marked up (&) would be not fully expanded as the full expansion has been done in other stages. Finally, the sequence of events between stable states is an interoperability test cases for the reference real-time I/OFSM of flow monitoring shown in Fig. 5 and for the test architecture given in Fig. 6.

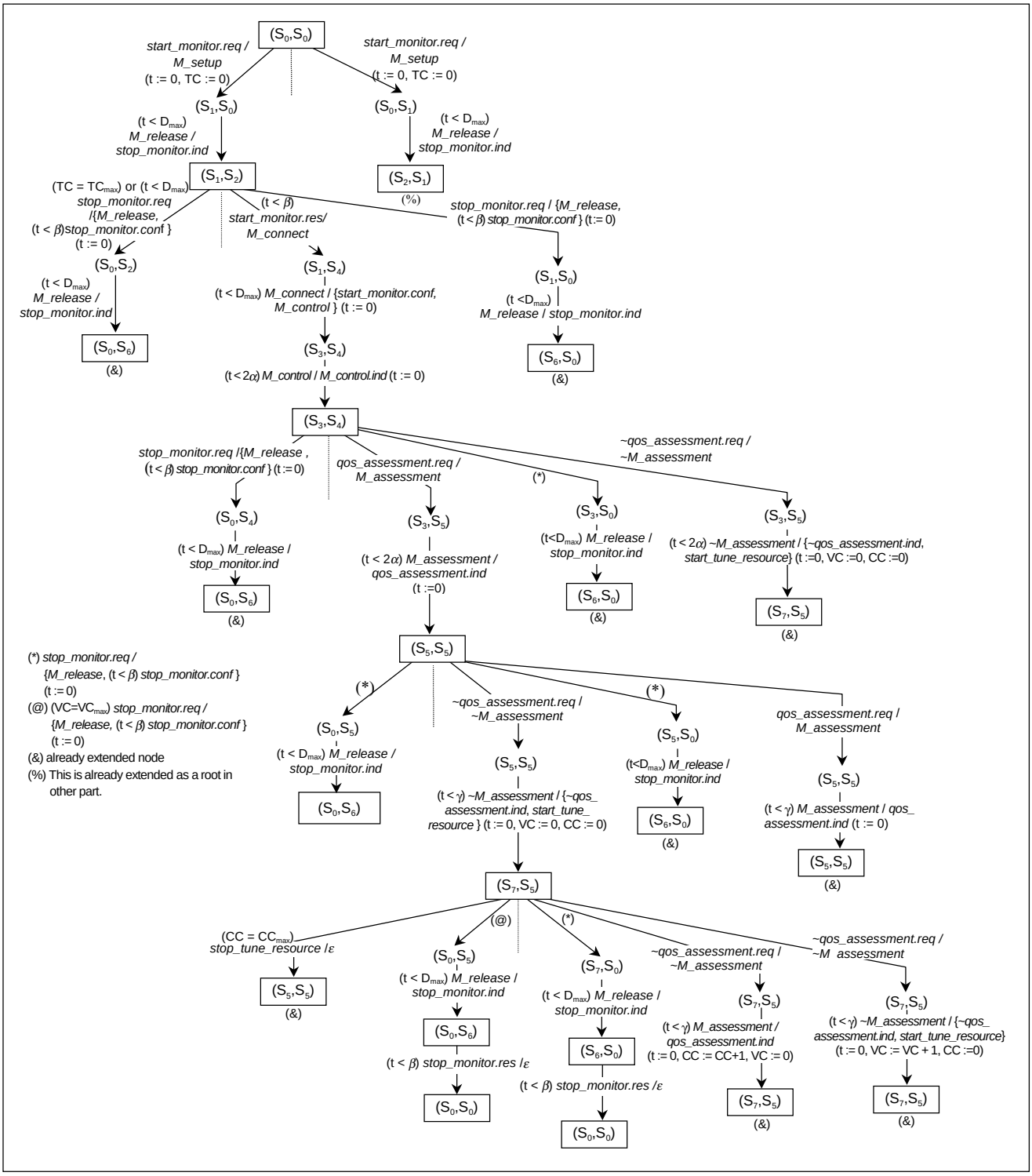

Fig. 8. Application of algorithm.

V. REAL-TIME TTCN NOTATION

In this section the real-time TTCN notation of interoperability test cases described above are presented.

ISO 9646-3 define TTCN that denotes ATS specifications by standard methods [22]. The goal of TTCN notation is to provide simplicity in general analysis for ATS and to provide the bases to do automatic execution of tests. ATS consist of sets of test cases.

The current TTCN is not appropriate test for testing real-time

Test Case Dynamic Behavior							
Test Case Name	: $(S_1,S_2)\_(S_0,S_6)$						
Purpose	: From starting state set (S_1,S_2), we test that Lta receives stop_monitor.conf from IUTa within ($t<\beta$) after Lta sends stop_monitor.req to IUTa, and also we test that LTb receives stop_monitor.ind from IUTb and then we confirm whether the arrived state set is (S_0,S_6).						
Nr	La	Time	Time Options	Behavior Description	CRef	Verdict	Comment
1	L1			+preamble_S_1_S_2			
2				(T1:=β)			
3	L2	L1, L1+T1		LTa ! stop_monitor.req			
4				(T2:=D_{max})			
5		L2, L2+T2	M	LTa ? stop_monitor.conf		(P)	
6		L2, L2+T1	M	LTb ? stop_monitor.ind		(P)	
7				+state_verify(S_0,S_6) (T1:=0, T2:=0)			
8				LTb ? OTHERWISE		(F)	
9				+postamble			
10				LTa ? OTHERWISE		(F)	
11				+postamble			
12		L2, L2+T2	M	LTb ? stop_monitor.ind		(P)	
13		L2, L2+T1	M	LTa ? stop_monitor.conf		(P)	
14				+state_verify(S_0,S_6) (T1:=0, T2:=0)			
15				LTa ? OTHERWISE		(F)	
16				+postamble			
17				LTb ? OTHERWISE		(F)	
18				+postamble			

Fig. 9. Real-time TTCN.

and multimedia systems. Because test events in TTCN are for message-based system and not for stream-based systems and in TTCN real-time can only be approximated. In this paper we use the notation of real-time ATS by means of real-time TTCN. In the extension of TTCN to real-time TTCN the syntactical extension is that statements are annotated with time labels for earliest execution time (EET) and latest execution time (LET) [23].

The EET and LET are time constraints that a transition can be performed in this time and transition must be performed within that time, respectively. In real-time I/OFSM <L, S, s_0, C, Tr >, an $EET_t \in \mathbf{R}$ and a $LET_t \in \mathbf{R} \cup \{\infty\}$ where **R** is set of real value defined for each transition $t \in Tr$, where we can assume that $EET_t \leq LET_t$. EET_t and LET_t define timing constraints which ensure that transitions can not be performed neither to early (EET_t) nor too late (LET_t). We suppose the default values are zero for EET_t and ∞ for LET_t.

A distinction between Fig. 9 and traditional TTCN is Time and Time Options column are added in Test Case Dynamic Bahavior table. An entry in the Time column specifies EET and LET for the corresponding TTCN statement.

In real-time TTCN, besides Label can be used in a GOTO, Label can be used in Time column, so that EET and LET values are computed relative to the execution time of the alternative identified by the Label. In Fig. 9 on line 3 the time labels (L1, L1+T1) are referencing to the execution time of the alternative in line 1 (for which label L1 is defined). If time option M (mandatory) is specified and the corresponding alternative can be successfully evaluated before it has been enabled for EET units, then this results in a Fail verdict.

VI. CONCLUSION

In this paper, based on QoS architecture model which includes ATM network in lower layer, we have specified the real-time behavior of flow monitoring of transport layer QoS protocol which is proposed to address QoS from an end-to-end's point of view. We have used a real-time I/OFSM to model the behavior of real-time flow monitoring over time. From the modeled real-time I/OFSM, we have generated interoperability test cases to check the correctness of METS protocol's flow monitoring behaviors for two end systems. We have experimented that formal specification (specified in real-time I/OFSM) approach gives the automatic or machine-assisted analysis of the correctness of specifications with respect to requirements.

This paper has also presented the translation of the test cases obtained by our generation algorithm to real-time TTCN. We have defined syntax and semantics of real-time TTCN. On a

syntactical level TTCN statements can be annotated by time labels. Time labels are interpreted as earliest and latest execution times of TTCN statements relative to the enabling time of the TTCN notation. The operational semantics of real-time TTCN is based on timed transition system. This approach gives the advantages that only a few syntactical changes are necessary, and TTCN and real-time TTCN are compatible.

In the future, we have plan to specify all functions of METS protocol in real-time I/OFSM and build a tool for the proposed framework with special attention to real protocol in order to generate the executable test cases in an automatic way.

ACKNOWLEDGMENTS

This research was supported partly by the Institute of Information Technology Assessment in Korea.

REFERENCES

[1] D. Hutchison, G. Coulson, A. Campbell, G. Blair, "Quality of Service Management in Distributed Systems," M. Sloman Ed., *Network and Distributed Systems Management*, Addison Wesley, 1994, Chapter 11.

[2] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of QoS Architectures," *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, Vol. 6 No. 3, May 1998, pp. 138–151.

[3] Byoung-Moon Chin *et al.*, "Automated Test Generation from Specification Based on Formal Description Techniques," *ETRI Journal*, Vol. 19, No. 4, December 1997, pp. 363–388.

[4] A. Cavalli, Byoung-Moon Chin, and Kilnam Chon, "Testing Methods for SDL Systems," *Computer Networks and ISDN Systems*, Vol. 28, No. 12, 1996, pp. 1669–1683.

[4] G. Bonnes, "IBM OSI Interoperability Verification Services," *The 3rd International Workshop on Protocol Test Systems*, IFIP TC6/WG6.1, 1990.

[6] J. Gadre, C. Rohre, C. Summers, and S. Symington, "A COS Study of OSI Interoperability," *Computer Standards and Interfaces,* Vol. 9, No. 3, 1990, pp. 217–237.

[7] G. S. Vermeer and H. Blik, "Interoperability Testing: Basis for the Acceptance of Communication Systems," *Protocol Test Systems, VI (C-19),* Elsevier Science Publishers B. V. (North-Holland), 1994.

[8] O. Rafiq and R. Castanet, "From Conformance Testing to Interoperability Testing," *The 3rd International Workshop on Protocol Test Systems*, IFIP TC6/WG6.1, 1990.

[9] N. Arakawa and T. Soneoka, "A Test Case Generation Method for Concurrent Programs," *Protocol Test Systems, IV*, J. Kroon, R. J. Heijink and E. Brinksma (Eds.), Elsevier Science Publishers B. V. (North-Holland), 1992.

[10] N. Arakawa, M. Phalippou, N. Risser, and T. Soneoka, "Combination of Conformance and Interoperability Testing," *Formal Description Techniques V* (C-10), M. Diaz and R. Groz (Eds.) Elsevier Science Publishers B. V. (North-Holland), 1993.

[11] R. Castanet and O. Kone, "Deriving Coordinated Testers for Interoperability," *Protocol Test Systems*, *VI*, O. Rafiq (Ed.), Elsevier Science B. V. (North-Holland), 1994.

[12] G. Luo, G. Bochmann, and A. Petrenko, "Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method," *IEEE Transactions on S. E.*, Vol. 20, No. 2, February 1994.

[13] S. Kang and M. Kim, "Test Sequence Generation for Adaptive Interoperability Testing," *The 8th International Workshop on Protocol Test Systems*, IFIP TC6/WG6.1, Evry, France, September 1995.

[14] A. Campbell, G. Coulson, and D. Hutchison, "A Quality of Service Architecture," *ACM Computer Communications Review*, April 1994.

[15] A. Campbell, G. Coulson, and D. Hutchison, "A Multimedia Enhanced Transport Service in a Quality of Service Architecture," *Proc. of Fourth International Workshop on Network and Operating Systems Support for Digital and Audio and Video*, Lancaster, UK, October 1993.

[16] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, 126: 183–235, 1994.

[17] Y.-H. Choe, D.-I. Lee, and S. Kumagai, "A Basic Theorem for Modular Synthesis of State Machine Allocatable Nets," *IEICE Trans. on Fundamentals*, Vol. E-81-A, No. 4, 1998.

[18] S. Kang and M. Kim, "Test Sequence Generation for Adaptive Interoperability Testing," *The 8th International Workshop on Protocol Test Systems*, IFIP TC6/WG6.1, Evry, France, September 1995.

[19] S. Kang and M. Kim, "Interoperability Test Suite Derivation for Symmetric Communication Protocols," IFIP *FORTE/PSTV '97*, Osaka, Japan, November 1997.

[20] Ana R. Cavalli, J. P. Favreau, and M. Phalippou, "Formal Methods for Conformance Testing: Results and Perspectives," *IWTCS VI*, North Holland: Elsevier Science Publishers B. V., September 1992, pp. 3–19.

[21] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Trans. on S.E.*, Vol. SE-4, No. 3, May 1978.

[22] ISO/IEC Information Technology—OSI Conformance Testing Methodology and Framework—Part3: Tree and Tabular Combined Notation (TTCN*), ISO/IEC IS 9646-3*, 1996.

[23] Q. Thomas and G. Jens, "Real-Time TTCN for Testing Real-Time and Multimedia Systems," *10th International Workshop on Testing of Communicating System*, IFIP TC6, Cheju, Korea, September 1997.

Byoung-Moon Chin was born in Seoul, South Korea in 1953. He received his B.S. degree in electrical engineering in 1976 and M.S. degree in computer engineering in 1983, both from Seoul National University. He received his Ph.D. Degree in computer science in 1996 from Korea Advanced Institute of Science and Technology (KAIST). He joined ETRI in 1980, where he is currently working as Director at the Protocol Engineering Center, and a Special Rapporteur of ITU-T SG7 Q.23 on data communication protocol testing. His research interests include protocol testing, test methodology, protocol engineering and computer networks.

Sung-Un Kim received his B.S. degree in electronics engineering from Kyungpook National University in 1982. He joined ETRI in 1982 and then Korea Telecom Research Labs (KTRL) in 1985. While working for the KTRL, he received his M.S. and Ph.D. degrees in computer science from the University of Paris 7 in 1990 and 1993, respectively. He is currently a Professor in the Department of Telematics Engineering, Pukyung National University, Pusan, Korea. His interests include protocol engineering on telecommunications, protocol testing, computer networks, and distributed systems.

Sung-Won Kang received B.A. from Seoul National University in 1982 and received M.S. and Ph.D. degrees in computer science from the University of Iowa in USA in 1989 and 1992, respectively. Since 1993, he has been a senior researcher at Korea Telecom. In 1995–1996, he was a guest researcher at National Institute of Standards and Technology of USA. In 1997, he was co-chair of the 10th International Workshop on Testing of Communicating Systems. Currently he is the head of the Protocol Engineering Team at Korea Telecom R&D Group. His research interests include communication protocol testing, program optimization and programming.

Chee-Hang Park received the B.S. degree in applied physics from Seoul National University, Korea in 1974, the M.S. degree from Korea Advanced Institute of Science and Technology, Korea in 1980, and the Ph.D. degree in computer science from University of Paris 6, France in 1987. During the last 10 years he has been involved as project leader in several large projects such as Multimedia Computer System Development and High Speed Parallel Computer System Development. His research interests include multimedia systems, distributed system, middleware, groupware, network virtual computing, and mobile agent architecture. He is currently Executive Director of Information Support Division, ETRI.