A Formal Specification of DSM-CC Interfaces

Yoonsik Cheon, Soosun Cho and Heung-Nam Kim
Real-Time Computing Department
ETRI Computer & Software Technology Laboratory
{cheon, scho, hnkim}@etri.re.kr.

Abstract—This paper explores an application of formal methods to the standard description in the context of DSM-CC (Digital Storage Media - Command and Control). DSM-CC is a recent ISO/IEC standard developed for the delivery of multimedia broadband services such as video on demand and home shopping. The standard specifies the user-to-network (U-N) protocols and the user-to-user (U-U) interfaces. We suggest a technique for formally specifying the U-U interfaces, provide a concrete example and show the benefits of applying formal techniques to the standard descriptions. In particular, we demonstrate that a declarative interface specification method tailored to CORBA-IDL is well suited to specifying the behavioral semantics of DSM-CC interfaces. Abstract models for interface types are defined by a family of mathematical functions and the interface operations are specified by a pair of pre and postconditions written in terms of the abstract models. An explicit connection is established between the abstract world and the interface world by an abstraction function that maps interface values to abstract values.

Keywords—DSM-CC, U-U interface, abstraction, formal methods, interface specification, stream

I. INTRODUCTION

There is a widespread belief in the formal methods community that formal techniques are ideally suited for specifying standards. In particular, formal notations are advocated for specifying communication protocols and services being standardized by ISO [1]. In fact, formal methods have been used and are being used in varying degrees to aid the development and description of standards [2][3]. Formal methods can be used to provide precise, abstract models for standards and to provide analytical techniques based on these models. It can be used to provide notations for describing specific design made and it is also useful for simulating behavior or rigorously testing conformance [4].

It is reasonable to expect that formal methods will have a wide variety of uses in the area of MPEG standards as well. In this paper, we are concerned with formally specifying DSM-CC (Digital Storage Media — Command and Control), a recent addition to ISO/IEC MPEG standard series [5]. Indeed, we can enumerate several different things that might be formalized: core concepts, communication protocols between client and server, and application portability interfaces and their behaviors. It seems that there is no universal formal specification language or approach best suited for the work on formalizing DSM-CC. The best approach is through a composite approach which uses several techniques suggested so far. For example, the set theory-based Z notation [6] is known to be excellent for specifying mathematical concepts and defining underlying semantic models [7][8]. Protocols and services can be best specified by using process algebra-based languages such as CCS [9], CSP [10] and LOTOS [11], which are equipped with the theory of concurrent communicating processes. There are also formal specification languages specifically designed for specifying the interface of program modules, sometimes called (behavioral) interface specification languages (BISL) [12][13][14].

In this paper we show that we need a formalization of interfaces and behaviors of DSM-CC standards. Without such a formalization of behaviors, it is in question whether the promised portability of multimedia applications can be achieved through the use of standard such as DSM-CC. We establish a foundation for applying formal methods to standard specification, provide a concrete example of such an application, and show its benefits. In particular, we demonstrate that the declarative interface specification style tailored to CORBA-IDL is ideally suited for specifying the behavioral semantics for DSM-CC interfaces. We naturally hope to reveal a number of problems or inconsistencies in the standard specification, if any. One of the most important contributions of our work is that we provide a concrete example of formal interface specifications in the context of MPEG standards. We hope that the example clearly shows the benefits of formal methods applied to the standard development and specification. There are also several novel features in our method of interface specifications — e.g., making abstraction functions explicit and the notion of hidden interface operations.

In the rest of this section we give a quick overview of DSM-CC. Section II informally introduces the notion of interface specifications and its general approach. In section III and IV, the heart of this paper, we formalize the stream interface of DSM-CC standard. For this, we first define an abstract model for the stream interface and then formally specify its interface operations based on the formal model. We conclude in section V with a discussion of lessons we learned and further research issues.

A. An Overview of DSM-CC

DSM-CC is a very recent ISO/IEC standard developed for the delivery of multimedia broadband services [5][15]. It defines the protocols needed to deliver a complete application such as video on demand or home shopping. Fig. 1 shows the functional reference model of DSM-CC. It is based upon a very general...
model of client, server and network entities for selecting, accessing and controlling distributed video sources. In the DSM-CC model, a stream is delivered from a server to a client through a logical entity called the session and resource manager (SRM). The SRM provides a (logically) centralized management of the DSM-CC sessions and resources. The standard does not dictate how each entity must be realized, rather it focuses on the user-to-network (U-N) protocols and the user-to-user (U-U) interfaces. The U-U information flow is used between the client and the server and the interfaces provide a generic set of multimedia interfaces, i.e., a set of modular building blocks that can be used to enable a wide range of multimedia applications. For example, defined are such interfaces as for navigating services provided by a server, attaching to (or detaching from) a particular service, reading or writing to files stored in the server, and manipulating MPEG continuous media streams. The U-N information flows between the SRM and the client or the server and its primary purpose is to control sessions and network resources. For example, defined are such protocols as for creating sessions, allocating resources to a session, de-allocating resources from a session, and destroying a session.

In this paper we are only concerned with the U-U interfaces, in particular the stream interface, the core of U-U interfaces, but our approach is equally applicable to other interfaces or standard descriptions as well. The stream interface primitives are to control the delivery of a media stream through commands such as pause, resume, play, jump, etc. The server is represented as a stream object that provides this interface and can transport MPEG over the network.

II. INTERFACE SPECIFICATIONS

A program consists of several modules declaring types, variables, and procedures with their implementation. The interface of a module describes the syntactic and semantic properties of the module. The syntactic interface of a module comprises the type, variable, and procedure identifiers as well as the types of variables, procedure parameters, and procedure results. The semantic interface of a module captures the behavior of the procedures and data structures of the module. Currently software modules are mostly documented by their syntactic interface, an informal description of their behavior, and a couple of examples to illustrate their application. For example, the DSM-CC interface are defined in the Object Management Group (OMG) Interface Definition Language (IDL) [16], which is standardized by ISO/IEC. The OMG IDL can describe the syntactic interfaces of modules in language and protocol independent way with a strong typing system. However, there is no way to specify the semantic interfaces. An informal description may be used but it is imprecise and incomplete and can not be used for formal development, automatic analysis and verification. This has led to the development of interface specification languages, i.e., languages that allow to formulate interface properties in a formal notation [4] [12] [14] [13].

An interface specification language has to bridge the gap between the operational, state-based world of programs and the declarative, state-less properties specifying program behavior. There are two extreme approaches: operational approach and declarative approach. In the operational approach the behavioral annotations are given as boolean expressions of underlying programming language [17] [18]. In the declarative approach the annotations are formula that may refer to program variables, but not to procedures of the underlying programming language. The operational interface specifications are in general easy to learn and they are executable; annotations can be checked at runtime. On the other hand, the expressive power of such annotations is rather limited. They do not support free variables or quantification, and there is no guarantee of termination and defined-ness of annotations. Even worse, they are implementation-dependent and unsuitable for formal verification. Thus, we explore the declarative approach. In the declarative approach, each specification consists of two components: an abstract model and an interface specification [4] [12] [14]. The abstract model defines the underlying mathematical models of the modules being specified, and the interface specification specifies the interface properties of modules in terms of the abstract model. The interface specification part is typically composed of an invariant property of the module and a set of operation specifications for the operations exported by the module.

III. A FORMAL MODEL OF DSM-CC STREAM INTERFACE

A. DSM-CC Stream Interface

The core of DSM-CC U-U interfaces is the stream interface that defines operations to control the delivery of a MPEG media stream stored in the server [5]. The standard specifies such operations as pause, resume, play, jump, status and reset. These stream operations make use of a temporal addressing scheme called normal play time (NPT) to support random positioning and a variety of play rates. The NPT can be thought of as a virtual clock that a viewer associates with his program. It is similar to the one digitally displayed on a VCR. The NPT clock advances normally in the normal play mode, advances at a faster rate in the fast forward mode and decrements in the reverse play mode. Using NPT, it is possible to request a position relative to a specific application program and to control the positioning of the stream (e.g., jump to a specific position). The temporal status of stream objects consists of current NPT, start NPT, stop NPT, viewing mode (either forward or backward) and rate (relative to the normal playing speed).

A stream object can be abstractly viewed as a state machine that models the delivery of MPEG over the network through

![Fig. 2. A state transition diagram of stream objects](image-url)
commands such as pause, resume, etc (See Fig. 2). Upon receipt of a command, the stream object will modify its state machine, and perform the indicated function. The machine consists of six states such as P, ST, T, etc1. The state transition is represented by two actions: pause and resume. The resume action specifies the temporal position at which to begin transport and the pause action specifies the stream position at which to suspend the transport. Informally the meaning of each state is as follows.

- **P (Pause)**: The media stream is not being transported.
- **ST (Search Transport)**: The server is searching for the start NPT on which to transport the media stream.
- **T (Transport)**: The server is transporting the media stream and will pause at the end of stream.
- **TP (Transport Pause)**: The server is transporting the media and will pause at the stop NPT.
- **STP (Search Transport Pause)**: The server is searching for the start NPT. Once at the start NPT, it will transport the stream until the stop NPT.
- **PST (Pause Search Transport)**: The server is searching the media stream. It will transport the stream until the stop NPT is reached, then it will search for the start NPT to start a new transport.

### B. A Formal Model of Stream

Fig. 3 shows our formalization of an abstract model for stream objects, i.e., their abstract values and functions. A collection of abstract (mathematical) sets are defined such as Npt, Mode, Rate, State, Mpeg and Stream. We call them sorts, a jargon in the algebraic specification languages. The values of sorts are either enumerated or specified by injection functions. For example, the values of sort Npt is either NOW or given by an injection function npt, which maps Nat values to Npt values. Technically, Npt is the union of the singleton set {NOW} and the range of the injection npt (i.e., Npt = {NOW} \cup ran(npt)). A stream is defined to be a tuple of state, temporal status and Mpeg media. The temporal status of a stream consists of current NPT, start NPT, stop NPT, viewing mode and viewing rate. The viewing mode describes the direction of play, either forward (FWD) or reverse (REV). The viewing rate describes the display speed and a pair of Nat is used for that. The value (1,1) denotes the normal play rate, (2,1) denotes the play rate of two times faster than the normal play rate, and (1,2) denotes the play rate of two times slower than the normal rate.

\[
\begin{align*}
Npt &= NOW \mid npt(Nat) \\
Mode &= FWD \mid REV \\
Rate &= rate(Nat, Nat) \\
State &= P \mid ST \mid T \mid TP \mid STP \mid PST \\
Mpeg &= mpeg(Npt, ...) \\
Stream &= stream(State, Npt, Npt, Npt, Npt, Mode, Rate, Mpeg)
\end{align*}
\]

\[\text{pause: } Stream \times Npt \rightarrow Stream\]

\[\text{resume: } Stream \times Npt \times Mode \times Rate \rightarrow Stream\]

\[\text{advance: } Stream \rightarrow Stream\]

Fig. 3. An abstract model of stream

1 States such as Open, Error, etc, are ignored in this paper.

Three abstract functions are defined over the sorts of the stream model: pause, resume and advance. We also need several auxiliary functions to observe Stream values (see the signature below). Their definitions are trivial, so omitted in this paper. For example, the function mode is defined as:

\[\text{mode}(\text{stream}(s, t_1, t_2, t_3, m, r, p)) \equiv m.\]

\[\text{curNpt, startNpt, stopNpt, maxNpt: } Stream \rightarrow Npt\]

mode: Stream \rightarrow Mode
rate: Stream \rightarrow Rate

The abstract functions pause, resume and advance define the state transition of stream objects. The Npt argument to pause function specifies the stream position at which to suspend the media transport, and the Npt value to resume function specifies the stream position at which to begin the transport. The precise meaning of transition function pause is defined below. In the definition we use "[ ]" as a short-hand notation for the injection function stream and we also abbreviate unchanged fields with "...".

\[\begin{align*}
\text{pause}([P, \ldots], t) &= [P, \ldots] \\
\text{pause}([ST, t_1, t_2, \ldots], t) &= [ST, t_1, t_2, \ldots] \\
\text{pause}([T, t_1, t_2, \ldots], t) &= [T, t_1, t_2, \ldots] \\
\text{pause}([TP, t_1, t_2, \ldots], t) &= [TP, t_1, t_2, \ldots] \\
\text{pause}([PST, t_1, t_2, \ldots], t) &= [PST, t_1, t_2, \ldots] \\
\text{pause}([PST, t_1, t_2, \ldots], t) &= [TP, t_1, t_2, \ldots]
\end{align*}\]

The pause function sets the stop NPT of a stream to the specified Npt and makes a proper state transition, i.e., sets the state field of the stream according to the state transition machine shown in Fig. 2. If the stream is in the states ST, T, TP, STP and PST, it will transition to the states STP, TP, TP, STP and TP respectively. If it is in the state PST (i.e., a resume is queued), it will transition to the state TP indicating the stream delivery will cease when the stop NPT is reached, and no resume will occur. In the definition, we did not worry about boundary conditions or validity of argument values. For example, the NPT argument (t) to the function pause is expected to satisfy the condition \(\text{startNpt}(s) \leq t \leq \text{maxNpt}(s)\). We postpone this to interface specifications. It is an interface-level issue and leaving this to interface specifications makes the abstract model simple and understandable.

The abstract function resume causes the server to resume sending the stream at the specified NPT.

\[\begin{align*}
\text{resume}([P, t_1, t_2, \ldots], t) &= [ST, 0, t, \ldots] \\
\text{resume}([ST, t_1, t_2, \ldots], t) &= [ST, 0, t, \ldots] \\
\text{resume}([T, t_1, t_2, \ldots], t) &= [ST, 0, t, \ldots] \\
\text{resume}([TP, t_1, t_2, \ldots], t) &= [PST, t_1, t, \ldots] \\
\text{resume}([STP, t_1, t_2, \ldots], t) &= [ST, 0, t, \ldots] \\
\text{resume}([PST, t_1, t_2, \ldots], t) &= [PST, t_1, t, \ldots]
\end{align*}\]

If the state machine is in the states ST, T, P, or STP, it will immediately transition to the state ST. The server will then commence sending the stream from the start NPT position at the earliest possible time. When the server begins to send the MPEG stream indicated by the start NPT, the state machine will transition to the state T (refer to the definition of advance function below). If the state machine is in PST, only the start NPT field is updated; i.e., no state transition occurs. If the state machine is
in the state \( TP \), it will transition to the state \( PST \), followed by a proper update of the start NPT. The state \( PST \) means that when the state machine reaches the stop NPT, it will transition to \( ST \) and then \( T \), and MPEG stream delivery will commence from the new start NPT.

How do we model the delivery of MPEG streams? We model the progressing of media transport by the advance of current NPT. We model the virtual clock tick by an abstract function \( \text{advance} \). Let \( \delta \) be a unit of Npt, say 1 Npt. The function \( \text{tick} \) moves the current NPT pointer forward or backward depending on the current play mode.

\[
\begin{align*}
\text{tick}: Pnt \times Mode &\rightarrow Npt \\
\text{tick}(t, \text{FWD}) &= t + \delta \\
\text{tick}(t, \text{REV}) &= t - \delta
\end{align*}
\]

\[
\begin{align*}
\text{advance}([P, \ldots]) &= [P, \ldots] \\
\text{advance}([ST, t_1, t_2, t_3, t_4, m, \ldots]) &= \\
\text{if } t_1 = t_2 \lor t_2 = \text{NOW} \\
\text{then } [T, \text{tick}(t_1, m), \ldots] \text{ else } [ST, \text{tick}(t_1, m), \ldots] \\
\text{advance}([T, t_1, t_2, t_3, t_4, m, \ldots]) &= \\
\text{if } t_1 = t_4 \\
\text{then } [P, 0, 0, t_4, \text{FWD}, \ldots] \text{ else } [T, \text{tick}(t_1, m), \ldots] \\
\text{advance}([TP, t_1, t_2, t_3, t_4, m, \ldots]) &= \\
\text{if } t_1 = t_3 \lor t_3 = \text{NOW} \\
\text{then } [P, 0, 0, t_4, \text{FWD}, \ldots] \text{ else } [TP, \text{tick}(t_1, m), \ldots] \\
\end{align*}
\]

The \( \text{advance} \) function corresponds to the dotted transitions in Fig. 2. The first equation states that \( \text{advance} \) has no effect if the stream is in the state \( P \) — its current NPT remains the same meaning no further delivery of media stream. The second equation says that \( \text{advance} \) will transition the state machine to the state \( T \) if the current state is \( ST \) and the current NPT is equal to the start NPT or the start NPT is \( \text{NOW} \). The current NPT advances by \( \delta \) NPT (i.e., the virtual clock ticks). We use an arrow symbol \( \rightarrow \) to define a conditional equation. For example, the fifth equation states that if the current state is \( ST \), \( \text{advance} \) transitions the machine to \( TP \) if the current NPT is equal to the start NPT, that is to say, if the stream position reaches the start NPT. We can interpret other equations in a similar manner.

IV. INTERFACE SPECIFICATION OF STREAMS

Once we have a formal model for stream objects we are now in the position of specifying their interface properties. But to do that we first have to answer a couple of questions. Are the interface operations exported by the stream module atomic? Is the operation invocation synchronous or asynchronous? Even though there is no explicit mention in the standard on the atomic-ness of operations, they do not seem to be atomic. As an example, an MPEG stream delivery initiated by a play operation can be paused (i.e., interrupted) by invoking a pause operation. The semantics of operation invocations is synchronous, but simulates asynchronous invocation style. DSM-CC introduces the notion of request handle, a closure that encapsulates the return value of an operation invocation. The request handle allows an application process to overlap requests without blocking while waiting for replies from the remote service. A DSM-CC IDL compiler is expected to generate a request handle as the return value of an operation invocation.

What is the semantic model of interface operations? We take a very simple approach here. We think an interface operation consists of two part: a state transition part and a stream pumping part. The state transition part is atomic and executed concurrently with other operations. The stream pumping part is shared among interface operations. The idea is that when we invoke an interface operation, its state transition part is executed atomically and returns immediately with a request handle. The actual delivery of MPEG stream is modeled by the shared pumping part. For this, we assume there is a hidden (or internal) operation, say delivery, that models the stream pumping part (i.e., the pumping of MPEG streams). With respect to the formal model defined in the previous section, the hidden operation corresponds to function \( \text{advance} \). Thus, a snapshot of the system will consist of a set of operations that are invoked explicitly and one hidden operation, all running concurrently.

In the next section we are only concerned with specifying the behavior of the operation focusing on the state transition parts. We ignore the stream pumping part and just assume that there is such a hidden operation.

A. Interface Type Specifications

The user of stream interface is essentially interested in what service is provided by the interface and not how this service is achieved by the actual implementation. The basic idea of declarative interface specifications is to explain the service provided by an interface on an abstract, implementation-independent level. Then, the vocabulary of the abstract level can be used to specify the interface behavior. As an example, at the application level, NPT is represented by a pair of numbers, i.e., seconds and microseconds. In DSM-CC, it is defined as a structure type \( \text{AppNPT} \). The negative infinity (0x8000) seconds represents "now". For example, when the server receives a pause command with a time value 0x8000, it is expected to pause immediately. This representation of NPT is too much implementation-oriented and inappropriate for the specification purpose. Therefore, in our formal model, it is abstracted into an infinite set consisting of all the natural numbers plus a special element \( \text{NOW} \), called sort \( \text{Npt} \). The distinction, however, makes it necessary to establish an explicit connection between representation values like \( \text{AppNPT} \) and abstract values like \( \text{Npt} \) by an abstraction function. The abstract function maps the concrete data types of the programming language to the sorts of the interface types. For example, the abstraction function for \( \text{AppNPT} \), \( \alpha: \text{AppNPT} \rightarrow \text{Npt} \), can be defined as:

\[
\alpha(x, y) = \begin{cases} 
\text{NOW} & \text{if } x = 0 \times 8000 \\
|x| \times 10^6 + y & \text{otherwise}
\end{cases}
\]
In the following we assume that for each interface type there exists a corresponding abstraction function. It is not shown in this paper to save pages. Sometimes the abstraction functions are trivial (e.g., bool, int); so we may adopt a default mapping.

Objects of a type usually maintain certain properties during their lifetime. Many of these properties are well-formedness conditions on data representations. The properties may concern the representation of one abstract value or may relate two or more representations. Such properties are specified as interface invariants. An interface invariant is a formula with one free variable ranging over the type of the interface.

interface Stream : Base, Access {
  invariant s: Stream
    0 <= curNpt(s) <= maxNpt(s) /\ valid_rate(s)
  // ...
}

The function valid_rate asserts legality of rate values. For example, a Rate value of (0,0) is invalid. Its definition is trivial, so omitted in this paper. We first came up with a stronger form of invariant: \( \text{mode}(s) = \text{FWD} \Rightarrow (0 <= \text{startNpt}(s) <= \text{stopNpt}(s) <= \text{maxNpt}(s)) \). But we soon realized while formalizing interface operations that the invariant is too strong because operations like pause can take a stop NPT less than the start NPT to denote an immediate pause. This shows one benefit of writing formal specifications.

B. Operation Specifications

The standard specifies six operations for the Stream interface. They are reset, pause, resume, status, play, and jump. Two of the most important and fundamental operations are resume and pause. The operations play and jump can be defined in terms of resume and pause. We specify both pause and resume operations in somewhat detail and quickly go over the play and jump operations.

B.1 Resume Operation

The resume operation lets the server to start sending the MPEG stream at the given start NPT position with the given mode and rate. It makes a resume transition in the state machine formalized in the previous section.

```c
void resume(in AppNPT n, in Scale s)
  cases MPEG_DELIVERY,BAD_START,BAD_SCALE {
    requires true;
    modifies self;
    ensures
      self' = resume(self,n,toMode(s),toRate(self,s))
      except when !validNpt(n) then raised(BAD_START)
      when !validScale(s) then raised(BAD_SCALE);
  }
```

The behavior of an operation is formally specified by using a requires clause (precondition), a modifies clause (frame axiom), and an ensures clause (postcondition). The pre and postconditions are assertions about the abstract values of the operation’s parameters, including an implicit parameter self. The implicit parameter self denotes the receiving object of the operation invocation. The modifies clause asserts that only those object listed in the clause can change their values as the result of operation invocation. In the postcondition we use a primed variable (e.g., self’) to denote its post value, the value just after operation evaluation. An unprimed variable (e.g., self) denotes its pre value, the value just before operation evaluation. Mathematically, an operation denotes a partial relation on the space of input and output values of the parameters [19].

The pre and postconditions are composed of using abstract functions defined in the previous section. In particular, the core behavior of the operation relies on the interpretation of the abstract function resume. In addition, several auxiliary functions are used. The functions toMode and toRate convert the value of interface type Scale to the abstract value Mode and Rate respectively. In DSM-CC, a stream’s playing mode and rate is represented by a pair of signed and unsigned numbers (as struct Scale) [5]. A zero or positive enumerator represent a forward playing mode and a negative one represents a backward mode. A scale value represents a relative speed of MPEG playing with respect to the NPT rate (1,1) (see Section III-A).

A scale value of (0,n) (where n is not 0) indicates that the NPT should remain the same.

- **toMode**: Scale \( \rightarrow \) Mode
- **toRate**: Stream \( \times \) Scale \( \rightarrow \) Rate

The operation resume can raise several exceptions. An exception represents an abnormal termination of operation invocation. To specify exceptions we use an except clause in the postcondition. The meaning of except clause is that if the condition specified in the when clause is satisfied the evaluation of the specified operation should terminate in an abnormal state in which the predicate in the corresponding then clause is satisfied; otherwise the evaluation of the operation should terminate in a normal state in which the predicate preceding the except clause is satisfied [14]. A special predicate raised is used to assert that an exception has occurred. The abstract functions validNpt and validScale used in the when clauses assert whether NPT and Scale values respectively are valid or not.

```c
validNpt: Stream \( \times \) Npt \( \rightarrow \) Bool
validNpt(s,t) == \( (t = \text{NOW}) \lor (0 \leq t \leq \text{maxNpt}(s)) \)
validScale: Scale \( \rightarrow \) Bool
validScale(i) == i \( \neq 0 \)
```

A start NPT \( (n) \) which equals or exceeds the stream duration will result in a BAD_START exception. A scale of \( (0,0) \) is indeterminate and will result in a BAD_SCALE exception.

In the standard, the behavior of interface operations such as resume is not well explained and sometimes it is not clear how they behave. As a instance, let \( s \) be a stream object in state PST with a forward play mode (i.e., \( \text{mode}(s) = \text{FWD} \)). For a given start NPT \( n \), what if \( n \) is less than \( \text{curNpt}(s) \) or \( \text{stopNpt}(s) \)? Is this a valid input? In the standard there is no explicit mention on this situation. We have to infer or guess the intention of the standard developers. In our formalization, we allow such a start NPT value and interpret it as normal; the stream object will transition to ST when it reaches the stop NPT and it will start delivering...
MPEG stream from the new start NPT when it reaches at that NPT position. This partially shows the benefit of formal specifications — in that hidden questions and issues are popped up and come to the specifier’s attention.

B.2 Pause Operation

The pause operation is called to cause the server to stop sending the stream when it reaches a given NPT, called the stop NPT. In the specification given blow, the precondition is omitted, which defaults to requires true.

```plaintext
void pause(in AppNPT n) raises(MPEG_DELIVERY, BAD_STOP) { modifies self; ensures self' = pause(self, toNow(self,n)) except when !validNpt(self,n) then raised(BAD_STOP); }
```

As expected, the postcondition is specified in terms of the abstract function pause. If the stream is in the forward transport mode, either a start NPT (n) of negative infinity (i.e., NOW) or a start NPT less than the current NPT indicates an immediate pause. Refer to the definition of functions toNow below and advance in previous section. The standard does not define the value of exception BAD_STOP when it is raised. Therefore we leave it unspecified; this is an intentional under-specification. The actual presentation of video frames (e.g., freeze frame versus blanked or alternative display) is considered implementation-specific and is therefore not specified by the standard.

```plaintext
toNow: Stream \times Npt \rightarrow Npt
toNow(n) = if isNow(n) then NOW else n
```

```plaintext
isNow: Stream \times Npt \rightarrow Bool
isNow(n) = (n = NOW) \lor
  ((mode(n) = FWD \land n < curNpt(s)) \lor
  (mode(n) = REV \land n > curNpt(s))
```

The function toNow converts an NPT to NOW relative to the stream being played. If the stream is playing in the forward (FWD) mode and the given NPT is less than or equal to the current NPT, the NPT is converted into NOW. A similar rule is applied to the backward (REV) play mode.

B.3 Play and Jump Operations

The play operation lets the server to play the stream from a given start NPT (n1) to a stop NPT (n2). The DSM-CC standard mandates that the play operation behaves exactly the same as if the operation resume then pause have been called in quick succession. Thus, the postcondition is specified by using the abstract functions pause and resume.

```plaintext
void play(in AppNPT n1, in AppNPT n2, in Scale s) raises(MPEG_DELIVERY, BAD_START, BAD_STOP, BAD_SCALE) { requires interval(self,n1,n2,toMode(s)); modifies self; ensures self' = play(self,n1,n2,s) except when !validNpt(n1) then raised(BAD_START) when !validNpt(n2) then raised(BAD_STOP) when !validScale(s) then raised(BAD_SCALE); }
```

The precondition asserts that the two argument NPTs together denote a valid interval in temporal ordering. The function interval defines what we mean by an temporal interval. For example, if the stream is in the forward playing mode, the two values must be within 0 and the maximum NPT of the stream, and the start NPT must be smaller than the stop NPT.

```plaintext
interval: Stream \times Npt \times Npt \times Mode \rightarrow Bool
interval(t_1, t_2, FWD) = 0 \leq abs(s, t_1) \leq t_2 \leq maxNpt(s)
interval(t_1, t_2, REV) = 0 \leq t_2 \leq abs(s, t_1) \leq maxNpt(s)
```

abs: Stream \times Npt \rightarrow Npt

```plaintext
abs(s, t) = if t = NOW then curNpt(s) else t
```

The jump operation is called for the server to resume at a given start NPT when it reaches a given stop NPT. The operation must behave exactly the same as if the operations pause and resume have been called in quick succession. As expected, the specification is the same as that of the operation play except for the order of resume and pause function applications.

```plaintext
void play(in AppNPT n1, in AppNPT n2, in Scale s) raises(MPEG_DELIVERY, BAD_START, BAD_STOP, BAD_SCALE) { requires interval(self,n1,n2,toMode(s)); modifies self; ensures self' = resume(pause(self,n2),toNow(n1),toMode(s)) except when !validNpt(n1) then raised(BAD_START) when !validNpt(n2) then raised(BAD_STOP) when !validScale(s) then raised(BAD_SCALE); }
```

V. DISCUSSION

A major contribution of this paper is the application of formal methods — in particular, interface specifications — to the description of DSM-CC standard. Of course, our intention was not to completely replace the current informal standard description with a formal specification, but to provide a formal interpretation as a supplementary material. We think that both formal and informal specifications have their own roles in the context of standard description with their relative strengths, and can be used complementary. For the formalization of DSM-CC standard, we took a declarative approach, where the specifications of underlying abstract models and interface properties are separated. An abstract model is defined as a family of mathematical sets, called sorts, and a collection of total functions on them. Interface types and operations are specified in the Hoare-style pre- and postconditions using the vocabulary of the abstract model. One of the most distinguished features of our approach is that we make abstraction functions explicit. An abstraction function establishes a connection between the abstract world of models and the concrete world of interfaces and implementations. It converts a concrete value of interface types to an abstract value of the abstract model. This mapping is implicit in other interface specification languages and thus a specifier is sometimes enforced to use the concrete values for specifications or the interface properties can not be specified accurately. In our approach, a specifier can choose his own interpretation of abstraction functions. It provides separation of concerns and makes specifications very flexible, modular and convenient for formal
reasoning and verification because the specifier can choose the abstraction level that suits his purpose. We have shown an example of user-defined abstraction functions, $\text{AppNpt} \rightarrow \text{Npt}$. Without such an abstraction function, we would have to reason about interface properties in terms of $\text{AppNpt}$, which is too much implementation-biased, and thus cumbersome for formal verification.

The formalization process helped us a lot to deepen our understanding of the DSM-CC standard. It brought up many issues to the surface that, otherwise, had been buried down deep under the sea and never came to our attention. We kept asking the "what if" questions. One of the most frequent questions was about boundary conditions and the temporal ordering of current NPT, start NPT, stop NPT and maximum NPT of a stream. We did not find any serious technical faults in the standard description but we found several vague, imprecise, unclear things and missing explanation. As a side product we also gained some experience in writing and composing interface specifications. We used non-determinism, loose specification and some forms of under-specifications for things that are inappropriate for formalization or too much implementation-detail. However, we had to be very cautious about their use because it would be often unclear from the specification to determine whether they were intentional or the specifier’s mistakes. An informal comment may be of help. Ideally, a specification language should provide a systematic way to specify the intention of the specifier. Larch, for example, provides special facilities such as implies clauses to specify redundancy in specifications [20].

There are several topics that we would like to study in the future. In this paper we only considered the stream interface in isolation. In the standard, however, the interface is defined to be a subclass of both Base and Access interfaces. The Base interface defines common operations for DSM-CC objects and the Access interface provides common description and access control attributes needed by most objects. This way of classifying and hierarchically organizing DSM-CC interfaces modularizes the standard description. In the formal specifications, however, we have to worry about how to specify the inheritance of interface specifications, and how to give formal semantics for such a specification inheritance. One possible approach that can be adapted is the homomorphic coercion and restore functions suggested in [19]. The second issue is about specification of exceptions and providing a formal ground for hidden operations. In the specification of stream interface operations we did not worry about the specification of exception MPEG DELIVERY. When is it raised and who raises it? Informally the server raises the exception when it cannot deliver the MPEG stream for some reasons. In our framework the delivery of stream is modeled by a hidden operation delivery. The notion of hidden interface operations is new in this paper. It seems reasonable for the hidden operation to specify the occurrence of exceptions. However, it is declared in the normal operations, say resume! Even worse, when the exception is raised by the hidden operation, the evaluation of the operation resume have been already completed returning a request handle. Without any formal foundation, we just assumed the existence of the hidden operation. What is the formal semantics for hidden operations? How do they interact with normal operations? Can we define a sound framework for the hidden operations? We need to further investigate on this in the future. Lastly our long-term goal is to formally verify and reason about properties of the DSM-CC standard. For example, the standard demands the play operation to behave exactly the same as to invoke the operations resume and pause consecutively. It would be a good exercise to formally prove this property. For this we may need a modular object-oriented verification method that fits with our specification framework.

References


