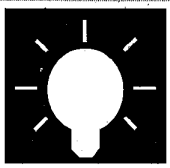


CMIS/P++: Extensions to CMIS/P for Increased Expressiveness and Efficiency in the Manipulation of Management Information

George Pavlou, University of Surrey
Antonio Liotta, University College London
Paola Abbi, Hewlett-Packard Italiana
Stefano Ceri, Politecnico di Milano

Abstract

CMIS/P is the OSI systems management service and protocol used as the base technology for the telecommunications management network. It is a generic object-oriented protocol that provides multiple object access capabilities to managed object clusters administered by agent applications. Its navigation and object selection capabilities rely on traversing containment relationships. This is restrictive because information models for emerging broadband technologies (SDH/SONET, ATM) exhibit various other relationships. In this article we present extensions to the CMIS service that provide a richer access language and show how these extensions can be supported by corresponding extensions to the CMIP protocol. These extensions allow traversal of any object relationship and filtering out objects at any stage of the selection process. CMIS++ provides much greater expressive power than CMIS, while CMIP++ supports the remote evaluation of the corresponding expressions, minimizing the management traffic required for complex management information retrieval. These extensions follow an incremental approach, starting from a version compatible with the current standard and gradually adding sophisticated features. The applicability and importance of the proposed concepts is demonstrated through an example from SDH management, while we also discuss implementation considerations.

The advent of broadband network technologies — synchronous digital hierarchy/synchronous optical network (SDH/SONET) transmission, asynchronous transfer mode (ATM) switching — which will form the basis of future telecommunications infrastructures poses complex management requirements. The International Telecommunication Union — Telecommunication Standardization Sector (ITU-T) has developed the telecommunications management network (TMN) [1] as the framework for their management. The latter uses the object-oriented information architecture of open systems interconnection systems management (OSI-SM) [2] as the means to model manageable resources and the associated access service and protocol to standardize interactions across management interfaces.

The OSI-SM access service and protocol are the common management information service (CMIS) [3] and Common Management Information Protocol (CMIP) [4], respectively. Managed elements and management applications acting in *agent* roles contain clusters of managed objects (MOs) organized in a management information tree (MIT) according to containment relationships. MOs exhibit hierarchical names based on the containment relationships. Management applications acting in *manager* roles access these objects by using CMIS/P in order to realize management policies. Access could be either to a single object, by using its name, or to multiple objects that are selected through *scoping* and *filtering* parameters. Scoping selects objects based on containment relationships starting from a particular position in the MIT (the *base*). Filtering further eliminates this selection through a Boolean expression containing assertions on attribute values. When an operation is performed on multiple objects, a series of linked replies is passed back to the manager application. The advantage of scoping and filtering is both expressive power of remote management requests and minimization of management traffic, the latter being one of the fundamental TMN architectural requirements [1].

The CMIS/P expressive power and capabilities, though much better than that of the Simple Network Management

0890-8044/98/$10.00 © 1998 IEEE

Protocol [5] for broadband network management, it is still not rich enough to address the complex management needs of emerging broadband environments. We have identified two main limitations:

- Scoping allows only for containment relationships to be navigated.
- When filtering is used in conjunction with scoping, it is only applied to selected objects after the selection process has been completed.

In order to address these limitations, we have extended CMIS/P with additional features. We have first extended CMIS to CMIS++, and we present these extensions as an MO manipulation language, similar to object-oriented database languages. This can be supported within manager applications and be mapped onto CMIS/P in order to retain interoperability with existing TMN applications in agent roles. We then present CMIP++ extensions that support the remote evaluation of CMIS++ expressions *within* agent applications. CMIS++ provides greater expressive power than CMIS by allowing traversal of any relationship and combining object selection and filtering at every stage of that process. CMIP++ supports the remote evaluation of the corresponding expressions, minimizing the traffic required for complex management information retrieval. Such powerful features also need to be implementable, and we examine relevant implementation issues. Finally, in order to show the applicability and importance of the proposed concepts, we present an example from SDH management.

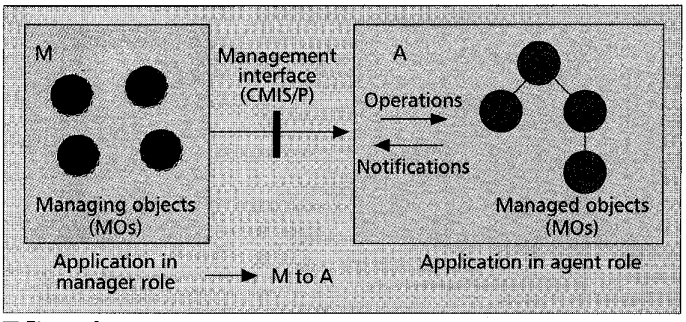
The rest of this article has the following structure. We first look briefly at the OSI-SM model, concentrating particularly on the navigation issues. We then present the CMIS++ extensions and present examples from SDH management to show its applicability. We then present the CMIP++ extensions, addressing federation and also discussing implementation issues. We finally close with a summary and conclusion, while we briefly discuss potential alternative approaches.

The OSI System Management Model

OSI-SM [2] projects an object-oriented model, with applications in agent roles "exporting" MOs that encapsulate managed resources at various levels of abstraction. Applications in manager roles access these objects in order to realize management policies. MOs conform to the management information model [6] and are formally specified using the Guidelines for the Definition of Managed Objects (GDMO) [7]; the latter is an object-oriented information specification language. The CMIS access service [3] has "remote method call" semantics and also allows operations on multiple objects. The OSI manager-agent model is shown in Fig. 1.

Managed objects typically exist in managed network elements, end systems, and distributed applications (i.e., in agents at the lowest level of a management hierarchy). They may also exist in higher-level management applications. In this case, MOs are abstractions of managed element resources. The distinction between manager and agent roles for a management application serves only the purpose of the model and is not strong in engineering terms: a management application may be in both roles. This is in fact the norm in a hierarchical layered architecture such as that projected by the TMN [1], in which management applications exist in element, network, and service management layers.

Each managed element or management application in an agent role exports a cluster of managed objects, also referred to as its management information base (MIB). Those MOs typically have many relationships, but containment is treated as a primary one that yields unique hierarchical names; hence,



■ Figure 1. *The OSI manager-agent model.*

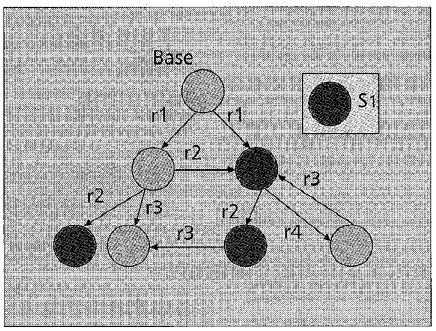
the MOs are organized in an MIT. Every object has a relative distinguished name (RDN) which is a tuple consisting of a naming attribute and its value (e.g., *connectionId=123*). The local distinguished name (LDN) of an object in the MIT is the concatenation of all the relative names after the root object; for example, {*subsystemId=nw, protocolEntityId=x25, connectionId=123*} identifies an X.25 virtual circuit. Interoperable communication between applications in manager and agent roles is achieved by the formal specification of management information in the agent, the access management service (CMIS), and the supporting protocol stack.

The containment relationships of MOs manifest themselves implicitly through their names; that is, there is no explicit information in an object's attributes denoting the containing and contained objects [6]. Other relationships manifest themselves as "pointer" attributes that contain the name of a related MO [8]. MOs in an MIT may be accessed either individually or collectively, through an object-oriented database query facility. Many objects may be selected by traversing containment relationships through *scoping*. The selection may be further eliminated by specifying a *filter* expression to be evaluated, containing assertions on attribute values linked by Boolean operators. An example using this facility could be *"retrieve all the X.25 VCs from that switch which start or terminate at address X."* This may be expressed by scoping all the objects contained immediately below the X.25 protocol entity object and using the filter *(objectClass=x25VC and (srcAddr=X or destAddr=X))*. One CMIS/P request is sent and a number of replies are returned, one for every accessed object. This reduces both the amount of traffic required to access a number of objects and also the overall retrieval time.

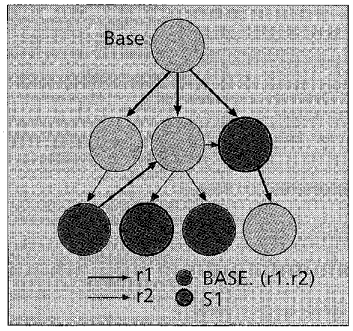
OSI management is a communications concept and, as such, is object-oriented only in terms of information specification and access. MOs are accessible across management interfaces, but the internal structure of the communicating applications is not dictated and may not be object-oriented. Research infrastructures such as OSIMIS [9] have shown early how such an object-oriented information specification may be mapped onto a fully object-oriented engineering framework, providing high-level application programming interfaces (APIs) and various transparencies. It has also shown that providing scoping and filtering facilities is easy in engineering terms and not expensive in memory and processing requirements [10]. Implementation considerations for CMIS/P++ draw on our relevant experience with CMIS/P in OSIMIS.

Extensions to CMIS

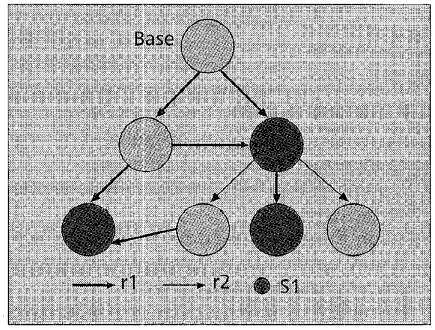
Given the limitations of CMIS/P identified above, the first step is to define more expressive access facilities. The intention is to avoid altering CMIS/P [2, 3] in order to maintain compatibility and interoperability with the increasing existing base of TMN implementations. These access facilities will be



■ Figure 2. *Generic relationship graph.*



■ Figure 3. *Let S1 = BASE ($r_1.r_2$) where (attA = 5).*



■ Figure 4. *S1 = BASE(r_1).*

implemented within manager applications and will have to be mapped onto CMIS/P. As such, we refer to them as CMIS++ to mean higher-level access facilities in the form of a language and associated APIs, but still conforming to the CMIS/P standards for interoperability. In this case, a well-defined mapping from CMIS++ to CMIS is also necessary.

A key limitation of CMIS/P is that it allows object traversal based only on containment relationships. Since it is our intention to support the arbitrary traversal of any relationship, the first step is to modify the management information model [6] to allow for this. In essence, we would like to treat containment as any other relationship which implies pointer attributes to express *contains* and *containedIn* relationships. These attributes should be added to the *top* class which is the root of the OSI-SM inheritance hierarchy [6]. For the purpose of CMIS++, this is only a "virtual" extension: In the case of CMIP++, this can still remain a virtual extension i.e. agents will treat the contains and containedIn "virtual attributes" in a special fashion when included in object selection expressions. An alternative approach with far reaching implications would be to actually modify the OSI-SM management information model.

In general, MOs are linked through generic relationships, rather than through the simple containment relationship, as shown in Fig. 2. Object references are supported through pointer attributes [8]. As such, no extension is required to the data definition language, which in this context is provided by the GDMO [7]. In the following subsections we propose to extend CMIS with path expressions in order to enhance its expressive power for retrieving interconnected objects with arbitrary relationships. Each path expression consists of a sequence of attribute names, represented through a "dot notation." A path expression denotes the *path* to follow in order to reach objects in the MIB graph . A simplified syntax for a path expression is the following ([] denotes optionality and {} potential repetition):

```
variable_name "=" simple_path_expression [
       "where" filter ]
simple_path_expression -> variable_name["."
       path]
path -> attribute_label{"."attribute_label}
```

We show the properties and capabilities of CMIS++ by examples. The full syntax for the CMIS++ M_GET operation is provided in Appendix A.

Simple Path Expressions

A simple path expression has the typical form expressed through the example

```
S1 = BASE.(r1.r2)
```

where S1 is a variable storing the set of all the objects retrieved starting from the `BASE` object and traversing the graph following the relationships r1 and r2, as shown in Fig. 2. The shaded objects are those that constitute the set S1.

The relevant attribute types, from a "navigational" point of view, are:

- Attributes storing a single object name, the LDN
- Attributes of type SET OF object names (nonordered list of names)
- Attributes of type SEQUENCE OF object names (ordered list of names)

These attributes represent relationships and are candidates as building elements for path expressions.

At each step of the graph traversal, a path expression retrieves a set of objects. These objects are progressively extracted from the MIB and assigned to variables. A variable can only be used to store object names. In the example above, r1 and r2 are attribute names that reference different objects using their identifiers (i.e., names). In CMIS the LDNs are used to uniquely identify objects. Thus, we assume that a variable can only store LDNs.

In CMIS++, as in CMIS, a starting point for the graph traversal has to be chosen. BASE — that is, the parameter already used in CMIS to refer to the base object — can be used as a variable name storing the base instance where the evaluation starts.

Qualified Path Expressions

Qualified path expressions are navigational clauses followed by predicates, composed using Boolean expressions. Predicates — syntactically, "where clauses" — reduce the number of objects retrieved via path expressions through matching rules on attribute values. A CMIS++ predicate is actually a CMIS filter. The predicates refer to attributes of the objects retrieved using path expressions. If the attribute is not present in the object, the relevant predicate simply evaluates to false (as in CMIS filters).

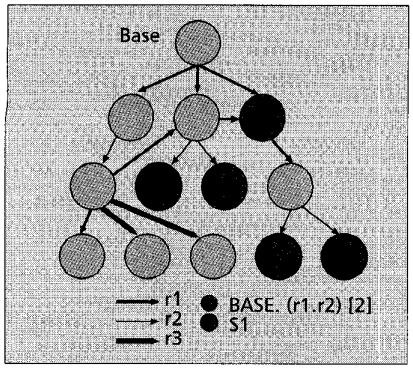
A qualified path expression has the typical form expressed in the following example (Fig. 3).

```
LET S1 = BASE.(r1.r2) where (attrA = 5)
```

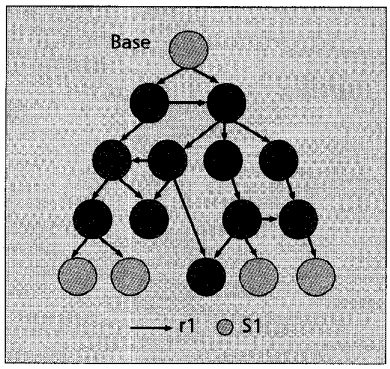
where only the objects retrieved starting from BASE, following the relationships `r1` and `r2` and having `attrA = 5`, are selected and stored in S1. Note that path expressions are used only for graph traversal. Note also that a CMIS++ query may include a sequence of qualified path expressions, the first starting always from the base object and subsequent ones typically taking as input the objects selected from the previous one. This means that predicates can be applied at different steps of the selection process, providing more flexibility than CMIS.

Path Expressions with Level-Based Restrictions

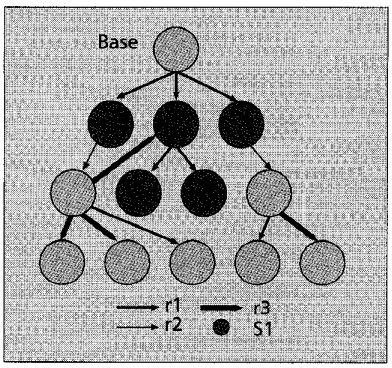
In CMIS++, as in CMIS, it is possible to restrict scope operations to a specified subtree. In CMIS++ this can be done by



■ Figure 5. *S1 = BASE.(r1.r2) [2] where (attr1 = 6).*



■ Figure 6. *S1 = BASE.(r1) [1..3].*



■ Figure 7. *S1 = BASE.(r1)[1..n].*

means of a qualifier which indicates the starting and final levels at which objects are extracted by the path expression. This mechanism is also extended with capabilities for infinite recursion and the retrieval of *fringe objects*.

Single-Level Restriction — The simplest form of level-based restriction is an individual-level restriction where that level is specified in the qualifier, as shown in the following examples. Note also that the absence of any level implies level [1], which is what was used in the previous examples.

```
LET S1 = BASE.(r1) [2]
   (extracts the objects at 2nd level reached
   following r1. Fig. 4).
LET S1 = BASE.(r1) [2] where (attr1 = "blue")
   (extracts the objects at 2nd level reached
   following r1 and having attr1 = "blue")
LET S1 = BASE.(r1.r2) [2] where (attr1 = 6)
   (goes down, following the sequence of rela-
   tionships (r1, r2) twice (level 2). The
   objects reached at the last step are extracted.
   The "where" option is applied at the end of
   navigation. Fig. 5).
```

Note that the expression

```
LET S1 = BASE.(r1.r2) [1]
```

is interpreted as

```
LET S1 = BASE.(r1.r2)
```

Multiple-Level Restrictions — Multiple-level restrictions can be specified as in the following examples:

```
LET S1 = BASE.(r1) [1..3]
   (extracts objects between levels 1 and 3,
   reached following r1. Fig. 6)
LET S1 = BASE.(r1.r2) [3..5] where (attr1 = 6)
   (goes down following the whole sequence of
   relationships (r1, r2) 5 times. The objects
   belonging to the 3rd, 4th and 5th level of
   navigation are extracted. The "where" option
   is applied at the end of navigation).
```

The previous examples show that a where clause is a filter that may be applied at the end of evaluating a path expression. Given the fact that a CMIS++ query may comprise a series of path expressions with where clauses ("qualified path expressions"), it enhances the expressive power of CMIS in which filtering can only be applied only at the end of the selection process (through scoping). The following example shows first a query with a single path expression in which the where clause is evaluated at the end. It is subsequently shown how the query should be formulated for the where clause to be evaluated at each step (i.e., level) of navigation.

```
LET S1 = BASE.(r1.r2) [3..5] where (attr1 = 6)
   (filter applied at the end of navigation as
   in CMIS)
LET S1 = BASE.(r1.r2) [3] where (attr1 = 6)
   S2 = S1.(r1.r2) where (attr1 = 6)
   S3 = S2.(r1.r2) where (attr1 = 6)
   (similar to the one above but with the filter
   applied at each level)
```

Recursion — Recursive path expressions denote the transitive closure of a path expression, interpreted as a binary relationship. The tree traversal through a certain direction is stopped when the given path cannot be followed any more.

Recursion can be specified as in the following example:

```
LET S1 = BASE.(r1) [1..n]
   (extracts all the objects, at any level,
   starting from BASE and following r1. The
   BASE object is not stored in S1. Fig. 7)
```

The starting level can be changed. In the following example the BASE object is included in S1.

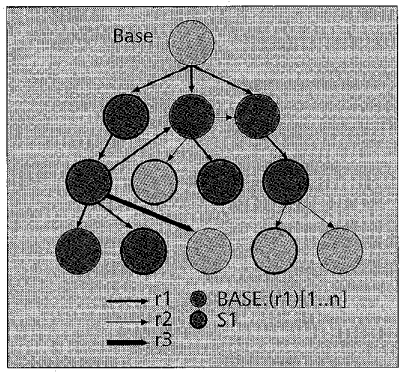
```
LET S1 = BASE.(r1) [0..n]
```

The following examples show the use of predicates in conjunction with recursion.

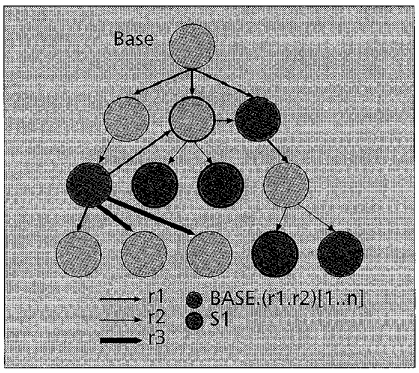
```
LET S1 = BASE.(r1) [1..n] where (attr1 = 3)
   (the relationship r1 is followed recursively
   until new objects are found. Only the objects
   having "attr1=3" are finally stored in S1.
   Fig. 8).
LET S1 = BASE.(r1.r2) [1..n] where (attr1 = 3)
   (the sequence of relationships "r1.r2" is
   followed. The search continues until the
   sequence can be entirely followed. The target
   objects are the ones having "attr1 = 3"
   Fig. 9)
```

Fringe Objects — Using recursion, *fringe* objects in a graph or subgraph can be extracted. Objects are termed fringe objects when either a relationship — or sequence of relationships — can no longer be followed or an object belongs to the extremes of the chosen subgraph. The following examples show the notation adopted in CMIS++ to retrieve fringe objects.

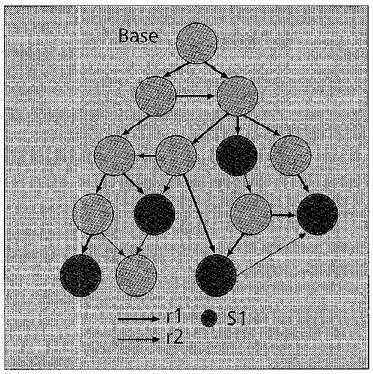
```
LET S1 = BASE.(r1)! [1..n]
   (extracts the last level: all the objects
   in the graph starting from which it is impossible
   to follow r1. Fig. 10)
LET S1 = BASE.(r1)! [1..n] where (attr1 = 6)
   (like in the previous example. The "where"
   option is evaluated at the end of navigation)
```



■ Figure 8. *S1 = BASE.(r1) [1..n] where (attr1 = 3).*



■ Figure 9. *S1= BASE.(r1.rs) [1..n] where (attr1 = 3).*



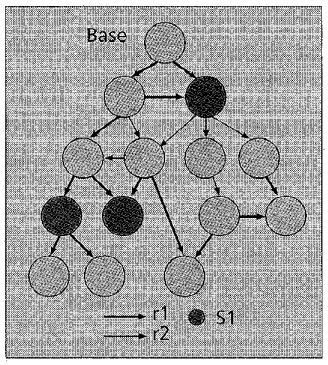
■ Figure 10. *S1 = BASE.(r1)! [1..n].*

```
LET S1 = BASE.(r1)! [1..3]
(objects at 1st and 2nd level from which it
is impossible to follow r1; by definition,
all objects reachable at 3rd level are fringe
ones. Fig. 11)
```

Set Expressions

From the previous introduction to CMIS++ through examples, it is clear that the functionality of CMIS scoping and filtering is provided in a more general fashion. Arbitrary relationships or sequences of relationships can be followed to arbitrary levels, with predicates (filters) applied at the end of each path expression evaluation. Sequences of such qualified path expressions can be specified in a single CMIS++ request, providing enormous flexibility in accessing management information.

In the typical case, the set of objects selected from a qualified path expression become the input to the next one, as demonstrated in the last example. An additional facility of CMIS++ is that it also allows specifying set expressions (union, intersection, difference) which operate on object sets which have been the output of previous expressions. The set features of CMIS++ are summarized below, followed by an example. The full formal definition of the CMIS++ SELECTION clause which replaces CMIS scoping and filtering is provided in Appendix A, being part of the M_GET query; its formal definition in ASN.1 is provided in Appendix B.



■ Figure 11. *S1 = BASE.(r1)! [1..3].*

- Arbitrary set definitions are possible: each one is a statement assigning the result of a graph traversal to a variable.
- Arbitrary set operations are possible between variables which denote sets of objects already extracted by the scope definition.
- Set operators are used to evaluate union, intersection, and difference among extents bound to the different variables.
- A final clause of the selection section (RETURN) determines which object, deposited within sets, has to be produced as output.

The typical structure of a selection clause can be seen in the example below.

```
SELECTION:
LET S1 = BASE.(r1) where (attrA = 5), # qualified
     path expression.
     S2 = S1.(r2) where (attrB > 34), # qualified
     path expression.
     S3 = S2.(r3.r4), # path expression.
     S4 = S2 INTERSECT S3, # set operation.
RETURN S4 where (attrC = 0) # output production.
```

The problem with set expressions, though, is that they make it difficult to federate CMIS/P++ queries across different agent applications; this aspect is discussed in more detail later.

Containment in CMIS++

In CMIS, containment is the only relationship that can be used for object navigation through scoping. When an MO is created, its identifier is inserted in the MIT. The latter can be traversed using the scoping mechanism in order to access particular instances. Once an object is selected there is no way to retrieve its "parent." This feature represents a semantic limitation. In fact, although containment is useful to order object instances, some semantic content is lost.

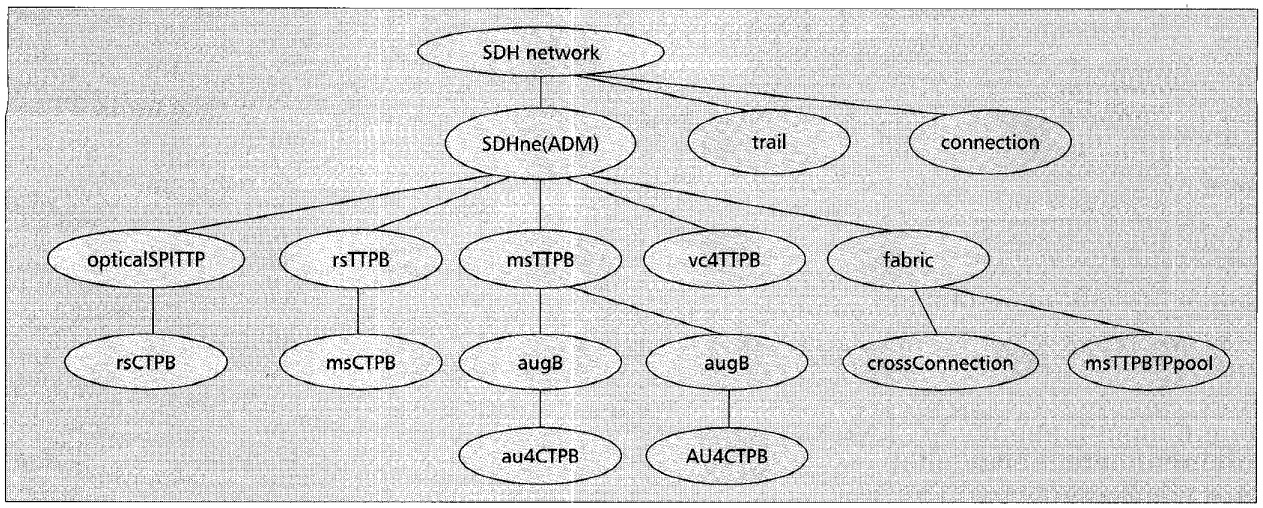
In CMIS++, the navigation of the containment relationship is implemented as follows: we add in each object the *contains* and *containedIn* attributes. These are implicitly defined for each object and do not need to be included in the object's declaration. When an object is created, its *containedIn* attribute is set to the value of its parent object in the tree. Accordingly, the *contains* attribute of the parent object is updated too.

The *containedIn* attribute is updated at object creation. The *contains* attribute is set-valued and is only updated when a new child object is created. Both attributes only have GET properties.

The use of the *contains* attribute allows backward compatibility with CMIS. CMIS scope and filter capabilities are still supported, as shown in the examples below.

```
LET S1 = BASE.(contains) [0..n] (whole sub-tree
  is retrieved)
LET S1 = BASE.(contains) [0..5] (objects from
  base to 5th level are extracted)
LET S1 = BASE.(contains) [3] (objects at 3rd
  level only are extracted)
LET S1 = BASE.(contains) [1..n] where (attr1 = 1)
  (objects at any tree-level having attr1 = 1)
LET S1 = BASE.(contains) ! [1..n] where (attr1
  = 1)
  (objects at the last tree-level having attr1 = 1)
```

Therefore, the new navigational capabilities affect the navigation of the containment relationship. The scope clause no longer exists, and a different way to scope objects is adopted. Path expressions can now be used, and the *contains* and *containedIn* attributes are the ones to be used to build the paths.



■ Figure 12. *The SDH containment tree.*

Example Use of CMIS++

We present here two scenarios in the context of the SDH technology, showing the expressive power of CMIS++. Both scenarios depict an error condition requiring a sequence of management operations to find out the required information. The latter can be obtained by accessing MOs in the MIT. We show some CMIS queries serving this purpose and their corresponding CMIS++ expressions.

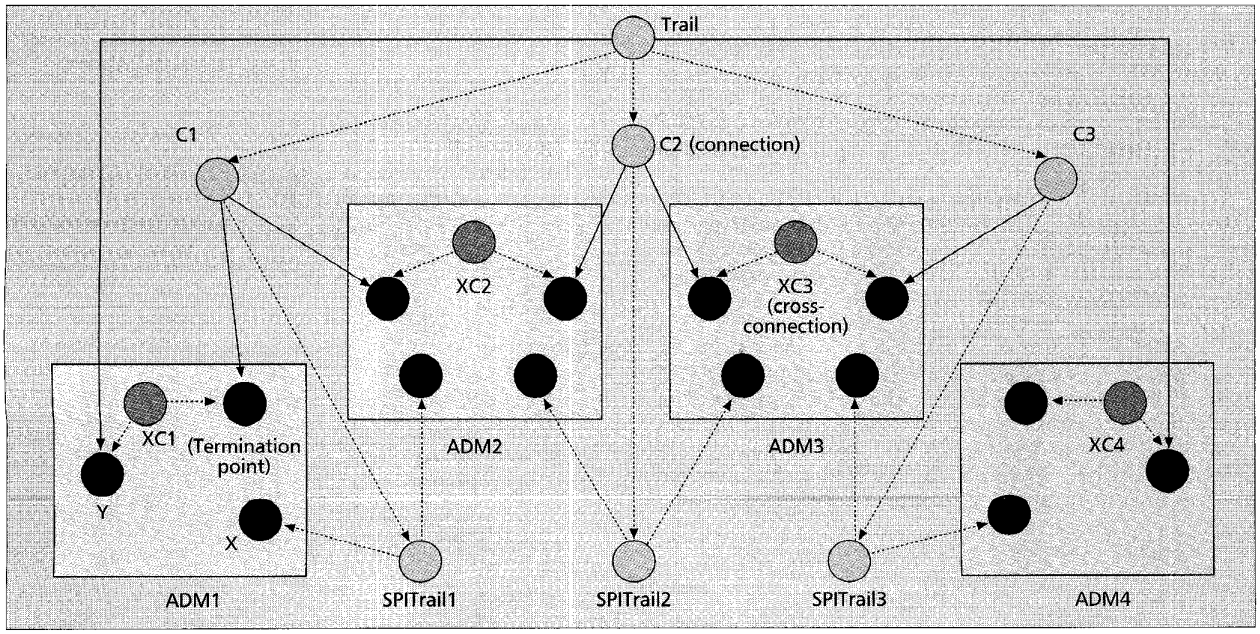
In our scenarios a network is built up from different multiplexing paths, each one crossing a number of add drop multiplexers (ADMs). ADMs allow a signal of specified bandwidth to be carried on the same physical link with other signals. ADMs are connected through *connection objects*, whereas different ports within a single ADM are connected through a *cross-connection object*. Fig. 12 shows the containment hierarchy of this environment; Fig. 13 depicts the scenario. The object classes, packages, and attributes used are defined in [11, 12]. The fundamental ones for this example are presented in Appendix C.

Fiber Break Scenario

In this scenario there has been a fiber break between two ADMs. A query is required in order to find out all the vc4-trails (end-to-end paths) affected by this fault. An error has been notified to a certain manager through a CMIS M_EVENT_REPORT, as usually happens in a TMN environment. A notification is generated by an opticalTTPBidirectional object, the termination point of an end-to-end path. We term X the identifier of the instance notifying the error occurrence (Fig. 13). Two possible queries — in CMIS and CMIS++, respectively — are described below. Another possible solution, highlighting further aspects of this scenario, is presented in [13].

CMIS Query —

```
/*Search for all the spiTrails having "X" as
"a" or "z" tp-instance;
```



■ Figure 13. *SDH scenario.*

```
we suppose that C is a set of records, each
one storing the trailID and its clientConnec-
tion. */

BEGIN

C= M_GET
   {
   BASE_INSTANCE: sdh_network,
   SCOPE: 1,
   FILTER: (a_tp_instance = X OR z_tp_instance = X),
   ATTRLIST: trailID, clientConnection
   }

/* For each connection in C the "clientTrail"
attribute will find the associated trail, which
is involved in the fibre break. */
   FOR EACH ( c in C.clientConnection ) DO
      M_GET
         {
         BASE_INSTANCE: c,
         SCOPE: no,
         FILTER: no,
         ATTRLIST: clientTrail
         }
   END
END
```

CMIS++ Query —

```
M_GET
{
   BASE: sdh_network,
   SELECTION:{
      LET S1 = BASE.contains where (a_tp_instance=X
            OR z_tp_instance=X),
         S2 = S1.clientConnection;
      RETURN S2
      }
   ATTRLIST: clientTrail
}
```

We can underline the difference between the two approaches. The CMIS query works in two steps. First, a number of trails and connections are retrieved through an M_GET. Then, for each of these connections another CMIS M_GET is required. Therefore, many M_GETs might be required to carry out the task.

In contrast, in CMIS++ the same query can be expressed with only a single M_GET. All the object LDNs retrieved following the containment relationship at the first step are stored in S1. Then, S1 can be used to reach all the client connections whose client trails are relevant to the fault.

Configuration Mismatch Scenario

In this scenario a crossConnection is changed in one ADM while a vc4 path is using it. Thus, the path is broken. The notification of the error is generated from a vc4TTPBidirectional object, marked Y in Fig. 13. Two possible queries — in CMIS and CMIS++, respectively — aimed at detecting the crossConnection affected by the change are described below.

CMIS Query —

```
T = M_GET
     {
     BASE_INSTANCE : sdh_network,
     SCOPE: 1,
     FILTER:(a_tp_instance = Y OR z_tp_instance = Y)
     ATTRLIST: trailID,
     }

/*This query finds out all the impacted
trails(T).
For each trail serving connections are searched
for */
  FOR EACH (t in T)
     {
     C = M_GET
        {
        BASE_INSTANCE : T,
        SCOPE: no,
        FILTER:no,
        ATTRLIST: serverConnectionList,
           a_tp_instance, z_tp_instance
        }
     }
  END

/*Let us suppose that, for a particular trail,
we find out 3 connections:
C1, C2, C3 (Ordered List); let us call
vc4TTPB(A) and vc4TTPB(Z) the two end points of
the trail.
We have to check whether there is a crossCon-
nection between each pair of
terminationPoints: */

a)
M_GET
     {
     BASE_INSTANCE : sdh_network,
     SCOPE: 3,
     FILTER: ( from = vc4TTPB(A) AND to =
          a_tp_instance(C1))
        OR (from = a_tp_instance(C1) AND to =
          vc4TTPB(A)),
     ATTRLIST: crossConnectionID
     }
b)
M_GET
     {
     BASE_INSTANCE : sdh_network,
     SCOPE: 3,
     FILTER: ( from = z_tp_instance(C1) AND to
          = a_tp_instance(C2))
        OR (from = a_tp_instance(C2) AND to =
          z_tp_instance(C1)),
     ATTRLIST: crossConnectionID
     }
c)
M_GET
     {
     BASE_INSTANCE : sdh_network,
     SCOPE: 3,
     FILTER: ( from = z_tp_instance(C2) AND to
          = a_tp_instance(C3))
        OR (from = a_tp_instance(C3) AND to =
          z_tp_instance(C2)),
     ATTRLIST: crossConnectionID
     }
d)
M_GET
     {
```

```
    BASE_INSTANCE : sdh_network,
    SCOPE: 3,
    FILTER: ( from = vc4TTPB(Z) AND to =
        a_tp_instance(C3))
      OR (from = a_tp_instance(C3) AND to =
        vc4TTPB(Z)),
    ATTRLIST: crossConnectionID
    }

/* We find the changed crossConnection when we
get no identifiers from one of
the queries. */
```

CMIS++ Query — In CMIS++ a single query is sufficient to find out each connection, coupled with its client trail and termination points, as shown below.

```
M_GET
{
  BASE: sdh_network,
  SELECTION: {
    LET S1 = BASE.contains where
        (a_tp_instance = Y OR z_tp_instance = Y),
      S2 = S1.serverConnectionList
    RETURN S2
    }
ATTRLIST: connectionID, trailID, a_tp_instance,
z_tp_instance }
```

Extensions to CMIP

In the two previous sections we presented the CMIS++ extensions to CMIS and demonstrated its use through examples that prove the increased expressiveness in manipulating management information that exhibits complex interobject relationships. CMIS++ can be implemented as an access language within manager applications and mapped onto CMIS/P for the exchange of interoperable messages across TMN interfaces. We have actually studied the mapping of CMIS++ to CMIS and concluded that a suboptimal mapping is trivial, while an optimal mapping that minimizes the CMIS/P interactions presents a difficult research problem. We do not present in this article an algorithm for the mapping of CMIS++ to CMIS because we are concerned rather with the direct mapping of CMIS++ to CMIP++, the latter being an extended protocol that supports the remote evaluation of CMIS++ queries within agent applications.

The advantage of CMIP++ is that it combines the expressiveness of CMIS++ with increased efficiency in manipulating management information, minimizing the amount of management traffic required to retrieve complex information. In principle, a single operation is adequate to access any number of MOs within an agent by navigating their relationships. The query is evaluated by the agent, and a number of replies are sent back in the form of a "consolidated result." The bandwidth required to access this information is minimized, while the overall latency is only fractionally bigger than that incurred accessing a single object, increasing the timeliness of the retrieved information. The obvious disadvantage is that CMIP++ is not directly interoperable with CMIP. Despite that, a generic "adaptor" application could be used to adapt between the two by deploying the same algorithm as for the mapping between CMIS++ and CMIS.

As already explained, the CMIS/P multiple object access capabilities are supported through the *scope* and *filter* parameters which are evaluated by the agent, starting from the base object specified in the request. In fact, we have already commented on the undesirable aspects of separating the scoping and filtering procedures; and, as such, CMIS++ supports a combination of the two, adding also multiple-step evaluation within a single request. Naturally, CMIP++ is only different from CMIP in the sense that scope and filter have been replaced by the objectSelection parameter. The CMIP protocol data units [4] are specified in the Abstract Syntax Notation One (ASN.1) language [11]. CMIP++ retains that specification but replaces scope and filter with the new objectSelection parameter, which is also specified in ASN.1. The exact object selection syntax is presented in Appendix B. The structure of this parameter supports the CMIS++ features in two stages: objectSelection supports the core CMIS++ features, while objectSelection2 supports the more sophisticated set expressions. The reason for this separation is explained below.

One important aspect of CMIS/P is that it can support federation. Whole agents can be organized in a hierarchical fashion, with MITs in subordinate agents hanging through logical links from the MIT of superior agents. The link objects in the superior agent are virtual in the sense that they map onto the root object of a subordinate MIT. A request to any agent can also address objects in subordinate agents through scoping. While scoping is evaluated, if a virtual link object is encountered, the same CMIS/P request is sent to the subordinate agent with the scope parameter modified accordingly to reflect the already scoped levels. The results are returned to the superior agent, which forwards them to the manager in a transparent fashion. The whole procedure can be recursive, with many levels of agents involved in the cascading of requests.

One fundamental aspect of the object selection procedure in order to support federation is that it should not require knowledge of any previous results in the evaluation process. The CMIS/P++ set expressions violate this principle since previous results are necessary in order to perform the set operations. If the target agent does not contain link objects pointing to subordinate agents, a CMIS/P++ request with set expressions could be fully evaluated. If, though, a link object is encountered, the whole procedure should be aborted with a special processingFailure error returned to the manager. This is the reason we have presented two versions of CMIP++ of increasing complexity: the first, supported by the objectSelection parameter, does not support set expressions and can deal with federation in the same manner as CMIP. The second, supported by the objectSelection2 parameter, does support set expressions but cannot support federation.

Finally, another important consideration regarding CMIS/P++ is its implementability since it is more complex than CMIS/P. In fact, CMIS/P was originally thought to be overly complex and unimplementable due to the scoping, filtering, and linked reply features. Early platform implementations such as OSIMIS [9, 10] have shown this not to be true and have been used widely for early TMN prototypes. Given our experience with CMIS/P in OSIMIS, we have attempted a CMIS/P++ implementation as a proof of concept, using the same environment. We have implemented the objectSelection parameter as specified in Appendix B and subsequent support in agents without great difficulty. In fact, the key differences from CMIS/P are the multiple evaluation steps and the fact that arbitrary relationships should be followed. In addition to our implementation, a research team in the Hewlett Packard Research Laboratories, Bristol, United Kingdom, have implemented CMIS++ in their prototype called *Society*. In summary, CMIS/P++ is more complex than CMIS/P but still perfectly implementable. Its additional complexity is a one-off problem, since CMIS/P++ will be "hidden" in generic agent and manager support infrastructures that will provide the relevant power, expressiveness, and efficiency to management applications.

Summary and Conclusions

In this article we have identified a number of problems in CMIS/P which stem from the fact that scoping allows only containment relationships to be navigated, and filtering is only applied at the end of the selection process through scoping. These problems become apparent when management information with complex relationships is accessed, as in SDH/SONET and ATM network elements and relevant network/element management applications. The current features of CMIS/P result in many complex queries that increase the complexity of management applications, increase the traffic incurred on the managed network, and reduce the timeliness of the accessed information. These problems can gradually be overcome using two approaches in an incremental fashion.

In the first approach, a CMIS++ higher-level access language has been specified which can be mapped generically onto CMIS/P (we have not addressed details of this mapping in this article). This results in increased expressiveness and reduced complexity of management applications while it maintains interoperability with the increasing installed base of TMN-capable network elements and applications. In the second approach, a modified CMIP++ protocol supports the remote evaluation of CMIS++ requests within agents. This reduces management traffic and increases the timeliness of accessed information, but breaks the compatibility with CMIP. CMIS/P++ could be thought as the new generation of TMN management service and protocol. Interoperability with existing CMIP-capable elements and applications could be supported through generic adaptation units. Finally, CMIS/P++ is only modestly complex compared to CMIS/P and, as such, perfectly implementable.

An alternative approach to CMIS/P++ would be to try and achieve the same effect through "intelligent" or "active" MOs. These contain interpreted logic that can be evaluated in a CMIS/P capable agent and return the results to the initiating manager. We have also been pursuing this line of research [14], the relevant advantages being compatibility with CMIS/P and the disadvantages being performance due to the interpreted approach, and security and federation issues. In general, intelligent mobile agent technology may support a lot of the OSI-SM/TMN functionality in a different fashion. We are working in this direction and will report our findings in the future.

Finally, the use of distributed object technologies in TMN environments is increasingly being considered, with the Object Management Group (OMG) common object request broker architecture (CORBA) [15] being the representative technology. The Network Management Forum (NMF)/Open Group Joint Inter-Domain Management (JIDM) task force has considered mappings of both GDMO and CMIS/P to the CORBA Interface Definition Language (IDL). In this case, MOs are mapped to CORBA objects, with each MO acting as a CMIS-like agent for its subtree. An alternative approach would be to separate CORBA-based MOs from CMIS-like access aspects, with special management broker (MB) objects acting as CMIS-like agents [16]. In this case, CMIS++ behavior can be supported by special MBs that exhibit a CMIS++-like IDL interface. We have been experimenting with CORBA-based CMIS and CMIS++-based MBs in the European-funded ACTS REFORM project and also intend to report our findings in the future.

Acknowledgments

This work described in this article was initially sponsored by Hewlett Packard and involved a collaboration among the HP Research Laboratory, Bristol, and University College London, United Kingdom, and Politecnico di Milano, Italy. We would like to acknowledge in particular Keith Harrison and Michele Campriani, both of HP Research Laboratories.

This work was continued in the ACTS REFORM project (Resource and Fault Restoration and Management for ATM), which is partially funded by the Commission of the European Union.

We would also like to thank the anonymous reviewers of this article for their helpful comments.

References

[1] ITU-T Rec. M.3010, "Principles for a Telecommunications Management Network (TMN)," Study Group IV, Report 28, 1991.
[2] ITU-T Rec. X.701, "Information Technology — Open Systems Interconnection — Systems Management Overview," 1991.
[3] ITU-T Rec. X.710, "Information Technology — Open Systems Interconnection — Common Management Information Service Definition," v. 2, 1991.
[4] ITU-T Rec. X.711, "Information Technology — Open Systems Interconnection — Common Management Information Protocol Specification," v. 2, 1991.
[5] J. Case, M. Fedor, M. Schoffstall and J. Davin, "A Simple Network Management Protocol," RFC 1157, 1990.
[6] ITU-T Rec. X.720,"Information Technology — Open Systems Interconnection — Structure of Management Information -Management Information Model," 1992.
[7] ITU-T Rec. X.722, "Information Technology — Open Systems Interconnection — Structure of Management Information — Guidelines for the Definition of Managed Objects," 1992.
[8] ITU-T Rec. X.725, "Information Technology — Open Systems Interconnection — Structure of Management Information — General Relationship Model," 1992.
[9] G. Pavlou *et al.*, "The OSIMIS Platform: Making OSI Management Simple," *Integrated Network Management IV*, ed. A. Sethi, Y. Raynaud, F. Faure-Vincent, Chapman & Hall, 1995, pp. 480–493.
[10] G. Pavlou, "Implementing OSI Management," Tutorial presented at the 3rd IFIP/IEEE Int'l. Symp. Integrated Network Mgmt., San Francisco, CA, 1993; ftp://cs.ucl.ac.uk/osimis/tutorial-isinm93.ps.Z
[11] ITU-T Rec. M.3100, "Telecommunications Management Network: Generic Network Information Model," 1992.
[12] ITU-T Rec. G.774, "Transmission Systems — Synchronous Digital Hierarchy (SDH) — Management Information Model for the Network Element View," 1992.
[13] P. Abbi and S. Ceri, "CMIS++: An Extension to CMIS for Accessing Telecommunication Databases," *Proc. 7th IFIP/IEEE Wksp. Dist. Sys.: Ops. and Mgmt.*, L'Aquila, Italy, 1996.
[14] A. Vassila, G. Pavlou, and G. Knight, "Active Objects in TMN," *Integrated Network Management V*, A. Lazar, R. Saracco, and R. Stadler, Eds., Chapman & Hall, 1997, pp. 139–50.
[15] OMG, "The Common Object Request Broker: Architecture and Specification (CORBA)," v. 2.0, 1995.
[16] G. Pavlou, "From Protocol-based to Distributed Object based Management Architectures," *Proc. DSOM '97*, A. Seneviratne, V. Varadarajan, and P. Ray, Eds., 1997, pp. 25–40.

Additional Reading

[1] ITU-T Rec. X.208, "Open Systems Interconnection —Model and Notation — Specification of Abstract Syntax Notation One (ASN.1)," 1988.
[2] NMF/Open Group, JIDM Specs., "Static Mapping of GDMO/ASN.1 to CORBA IDL," 1995; also "Interaction Translation," 1998

Biographies

GEORGE PAVLOU (G.Pavlou@ee.surrey.ac.uk) received his Diploma in electrical and mechanical engineering from the National Technical University of Athens, and his MSc. and Ph.D. in computer science, both from University College London. Over the last 10 years he has undertaken and directed research in the areas of communications protocols, performance evaluation, distributed systems, broadband network technologies, network management, and service engineering. He has been involved in a number of European collaborative research projects, addressing the management of broadband networks and next-generation telecommunication services. He has contributed to ISO, ITU-T, NMF, OMG, and TINA, and is the author of about 40 papers in international refereed conferences and journals. He has contributed parts of two books on network management. Since the beginning of 1998 he is a full professor at the University of Surrey, School of Electrical Engineering and Information Technology, where he leads the activities of the networks research group.

ANTONIO LIOTTA received his Diploma in electronic engineering from the University of Pavia, Italy, in 1994 and his MSc. in information technology from the Politecnico di Milano in 1995. Over the last five years he has undertaken research, first in the area of real-time control of autonomous mobile robots and then in the field of computer-supported cooperative work over ATM and ISDN networks. He is currently a Ph.D. candidate in computer science at University College London, where he is working on the application of mobile agent technologies to distributed network management. He is collaborating with the Hewlett-Packard Laboratories in

Bristol on management by delegation and code migration, and with the ESPRIT European project INSERT on Web-based management.

PAOLA ABBI received her Diploma in electronic engineering from the Politecnico di Milano in 1995. Her final year dissertation was undertaken under the auspices of a European research project and concerned the study of advanced systems for database management. She subsequently obtained a scholarship from Hewlett-Packard and was involved in a project related to telecommunications network management. After that, she worked at IBM for two years as a technical consultant and is currently working at Hewlett Packard Italy in the European technical support team.

STEFANO CERI is full professor of database systems at the Dipartimento di Elettronica e Informazione at the Politecnico di Milano; he was a visiting professor at the Computer Science Department of Stanford University between 1983 and 1990. His research interests focus on extending database technology to incorporate data distribution, deductive and active rules, and object orientation, and on design methods for data-intensive Web sites. He is responsible for several European ESPRIT projects at the Politecnico di Milano, including "Web-Based Intelligent Information Infrastructures." He was associate editor of *ACM Transactions on Database Systems* (1989–1992) and is currently associate editor of several international journals, including *IEEE Transactions on Software Engineering*. He is the author of several papers in international conferences and journals and co-author of several books, including *Active Database Systems, Advanced Database Systems, The Art and Science of Computing,* and *Designing Database Applications with Objects and Rules: The IDEA Methodology.*

Appendix A: Syntax of a CMIS++ M_GET

```
get -> "M_GET {"
         get_argument
         "};"

get_argument -> argument_body
         [",ATTRLIST:" attribute_list ]

argument_body -> base
         [ "," selection ]

base ->"( BASE:"
         attribute_label "=" base_managed_object_
           instance" )"

selection -> "SELECTION" ":{"
              "LET" selection_statement
              {"," selection_statement }
              "RETURN" selection_statement
              "}"

selection_statement->
       variable_name "=" path_expression [
         "where" filter ] | variable_name "="
       variable_name set_operator variable_name

path_expression -> simple_path_expression |
         structured_path_expression

simple_path_expression -> variable_name["." path]

structured_path_expression ->
       variable_name"."attribute_label naviga-
         tional_clause
     |variable_name".("attribute_label"."path")"
         navigational_clause

path -> "("attribute_label{"."attribute_label}")"

navigational_clause -> level

level -> "[" single_level "]" |
       "[" level_group "]" |
       "!"

single_level -> positive_integer

level_group -> integer ".." integer

variable_name -> [A-Z] [A-Za-z_0-9]*

set_operator -> UNION | INTERSECTION | DIFFERENCE
```

Appendix B: Syntax of the CMIP++ ObjectSelection Parameter

```
ObjectSelection DEFINITIONS ::=
BEGIN
IMPORTS AttributeId, CMISFilter FROM CMIP;

ObjectSelection ::= SEQUENCE OF PathExpression

PathExpression ::= SEQUENCE
  {
  relationships SEQUENCE OF Relationship OPTIONAL,
         -- non-existing or empty signifies
            "contains"
  relationshipNegation BOOLEAN DEFAULT FALSE,
  searchLevels SearchLevels OPTIONAL,
  filter CMISFilter    OPTIONAL
  }

Relationship ::= AttributeId
-- only ObjectInstance, SET OF ObjectInstance,
     SEQUENCE OF ObjectInstance
-- attribute values are meaningful

SearchLevels ::= CHOICE
  {
  singleLevel INTEGER, -- >= 0
  multipleLevels MultipleLevels
  }

MultipleLevels ::= SEQUENCE
  {
  firstLevel  INTEGER,
  lastLevel INTEGER OPTIONAL -- absence signi-
    fies "all subsequent levels"
  }

END

ObjectSelection2 DEFINITIONS ::=
BEGIN
IMPORTS PathExpression FROM ObjectSelection

ObjectSelection2 ::= SEQUENCE OF PathOrSetEx-
    pression

PathOrSetExpression ::= CHOICE
  {
  pathExpression[0] PathExpression,
  setExpression [1] SetExpression
  }

SetExpression ::= CHOICE
  {
  union   [0] IndexList,
  intersection[1] IndicesAB,
  difference  [2] IndicesAB
```

```
  }

Index ::= INTEGER
-- references (the product of) a position in the
     PathOrSetExpression
-- the first position is 1, hence an index
     should always be >= 1

IndexList ::= SET OF Index

IndicesAB ::= SEQUENCE
  {
  a Index,
  b Index
  }

END
```

Appendix C: Definition of the Fundamental Managed Objects Classes, Packages, and Attributes Used in the SDH Scenario Example

```
trail MANAGED OBJECT CLASS
  CHARACTERIZED BY trailPack;
  REGISTERED AS …;

connection MANAGED OBJECT CLASS
  CHARACTERIZED BY connectionPack;
  REGISTERED AS ...;

terminationPoint MANAGED OBJECT CLASS
  CHARACTERIZED BY tpointPack
  REGISTERED AS ...;

crossConnection MANAGED OBJECT CLASS
  CHARACTERIZED BY crossConnectionPack;
  REGISTERED AS ...;

trailPack PACKAGE
  ATTRIBUTES
     ATT trailID,
     ATT a_tp_instance,
     ATT z_tp_instance,
     ATT serverConnectionList;
  REGISTERED AS ...;

connectionPack PACKAGE
  ATTRIBUTES
     ATT connectionID,
     ATT clientTrail;
  REGISTERED AS ...;

crossConnectionPack PACKAGE
  ATTRIBUTES
     ATT crossConnectionID,
     ATT from,
     ATT to;
  REGISTERED AS ...;

tpointPack PACKAGE
  ATTRIBUTES
     ATT tpointID;
  REGISTERED AS ...;

trailID ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ....(string);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

a_tp_instance ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ....(termination
       Point);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

z_tp_instance ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ....(termination
       Point);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

serverConnectionList ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ... (SEQUENCE OF
       connection);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

clientTrail ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ... (trail);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

from ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ....(termination
       Point);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;

to ATTRIBUTE
     WITH ATTRIBUTE SYNTAX ....(termination
       Point);
     MATCHES FOR EQUALITY;
REGISTERED AS ...;
```