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

George Pavlou, Antonio Liotta
University College London, Department of Computer Science
Paola Abbi, Stefano Ceri
Politecnico di Milano, Department of Electronic Engineering and Information Sciences

ABSTRACT: CMIS/P is the OSI System Management service and protocol, used as the base technology for the Telecommunication 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 as information models for emerging broadband technologies (SDH/SONET, ATM) exhibit various other relationships. In this paper 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 to traverse any object relationship and to filter 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 adding gradually sophisticated features. The applicability and importance of the proposed concepts is demonstrated through an example from SDH management while we also discuss implementation considerations.

Keywords: Management, Information Retrieval, CMIS/P, OSI-SM, TMN, SDH

Introduction

The advent of broadband network technologies (SDH/SONET transmission, ATM switching) which will form the basis of future telecommunication infrastructures poses complex management requirements. ITU-T has developed the Telecommunication Management Network (TMN) [1] as the framework for their management. The latter uses the object-oriented information architecture of OSI System 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 that are 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 eliminates further this selection through a boolean expression containing assertions on attribute values. When an operation is performed on multiple objects, a series of linked results is passed back to the manager application in a "back-to-back" fashion. The advantage of scoping and filtering is both expressive power and minimization of management traffic, the latter being one of the TMN fundamental architectural requirements.

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

i. scoping allows only for containment relationships to be navigated; and
ii. filtering is only applied to the selected objects after the selection process through scoping 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 a managed object 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. 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 to traverse any relationship and to combine 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 need also 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 paper has the following structure. We first look briefly at the OSI-SM model, concentrating particularly in 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 discussing also implementation issues. We finally close with a summary and our conclusions, while we briefly discuss potential alternative approaches.

0-7803-4383-2/98/$10.00 © 1998 IEEE.

The OSI System Management Model

OSI System Management [2] projects an object-oriented model, with applications in agent roles "exporting" Managed Objects (MOs) that encapsulate managed resources at various levels of abstraction. Applications in manager roles access these objects in order to realize management policies. Managed objects 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 allows also operations on multiple objects. The OSI manager-agent model is shown in Figure 1.

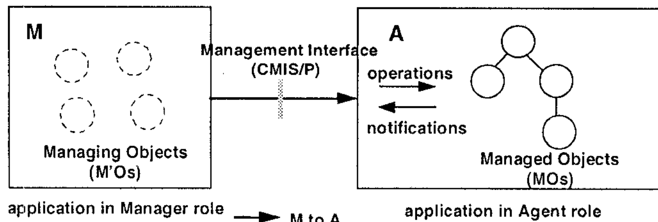

Fig. 1. The OSI Manager - Agent Model

Managed objects typically exist in managed network elements, systems and distributed applications. They also exist in management applications that act in agent roles. In this case managed objects are higher level 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 the one projected by the TMN [1], in which management applications exist in element, network and service management layers.

Each managed element or management application in agent role exports a cluster of managed objects, also referred to as its Management Information Base (MIB). Those managed objects have typically many relationships but containment is treated as a primary one that yields unique hierarchical names; hence, the managed objects are organized in a Management Information Tree (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 Distinguished Name (DN) of an object in the MIT is the concatenation of all the relative names after the root object, e.g. *{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 and the access management service and supporting protocol stack.

The containment relationships of managed objects manifest themselves implicitly through their names i.e. 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 managed object [8]. Managed objects in a 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 that 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 back-to-back, one for every accessed object.

OSI Management is a communications concept and, as such, it is object-oriented only in terms of information specification and access. Managed objects 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 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 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++ though, this means the OSI-SM MIM should be slightly modified.

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]. Therefore, no extension is required with respect to the data definition language provided, in this context, through the Guidelines for the Definition of Managed

Objects (GDMO) [7]. In the following sub-sections we propose to extend CMIS with path expressions in order to enhance its expressive power for retrieving interconnected objects. Each path expression consists basically of a sequence of attribute names, represented through a "dot notation", that gives the way to follow in order to reach objects pointed at by other ones. A simplified syntax for a path expression is the following:

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

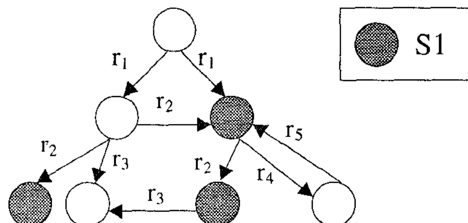

Fig. 2. Generic Relationship Graph.

We show the properties and capabilities of CMIS++ by examples. The full syntax of a CMIS++ M_GET operation is described in [13].

Simple Path Expressions

A simple path expression has the typical form expressed in the following 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` defined among objects (Fig. 2).

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

- attributes storing a single object value name, the Distinguished Name (DN);
- attributes of type "SET OF" objects names;
- attributes of type "SEQUENCE OF" object 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. It cannot store generic attribute values. In the example above, `r1` and `r2` are attribute names that reference different objects using their identifiers. In CMIS the DNs are used to uniquely identify objects. Thus, we assume that a variable can only store DNs.

In CMIS++, like in CMIS, a starting point for the graph traversal has to be chosen. BASE - that is, the clause already used in CMIS to refer to the base object - can be used as a variable name storing the base instances.

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 the matching of attribute values. A CMIS++ predicate is built as a CMIS filter. The predicates refer to attributes of the objects retrieved using path expressions. If the attribute is not applicable then the object is not selected (as in 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`. Notice that we use path expressions only for the graph traversal.

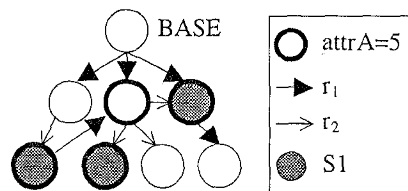

Fig. 3. LET S1 = BASE.(r1.r2) where (attrA = 5)

Path Expressions with Level-based Restrictions

In CMIS++, like in CMIS, it is possible to restrict scope operations to a specified sub-tree. In CMIS++ this can be done by means of a qualifier which indicates the starting and final levels at which objects are extracted by the path expression. In CMIS++ this mechanism is also extended with *recursion* and retrieval of *fringe objects*.

Single-level Restriction

The simplest form of level-based restriction is a single-level restriction where only one level is specified in the qualifier, like in the following examples:

```
LET S1 = BASE.(r1) [2]
   (extracts the objects at 2nd level
   reached following r1. Fig. 4).
```

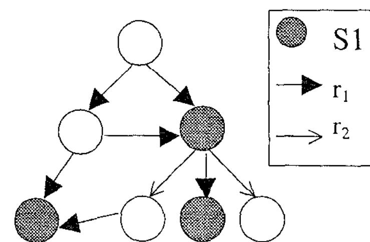

Fig. 4. S1 = BASE.(r1) [2]

```
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 whole
   sequence of relationships (r1, r2) 2
   times. The objects reached at the last
   step are extracted. The "where" option
   is applied at each step of navigation.
   Fig. 5).
```

Notice that the following expression

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

must be interpreted as

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

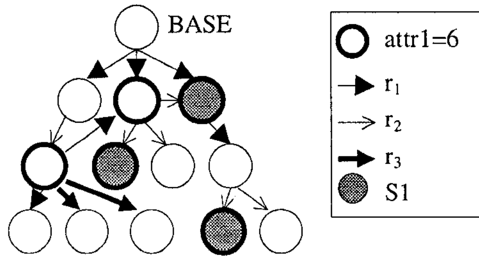

Fig. 5. S1 = BASE.(r1.r2) [2] where (attr1 = 6)

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)
```

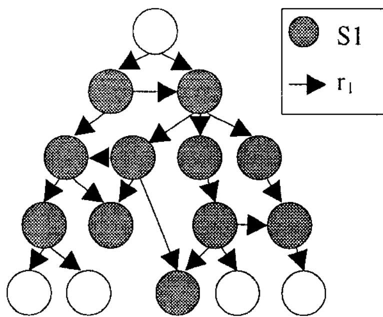

Fig. 6. S1 = BASE.(r1) [1..3]

```
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 each step of navigation).
```

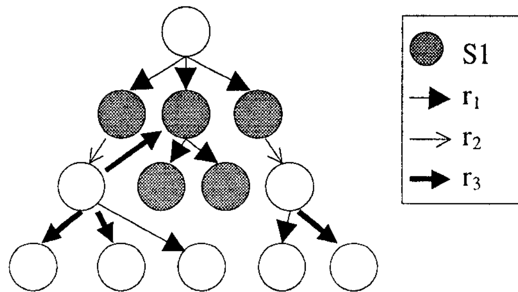

Fig. 7. S1 = BASE.(r1) [1..n]

The previous examples show that a "where" clause is a filter applied at each step of the graph traversal. This capability can significantly enhance the expressive power of CMIS in which filtering can only be applied after the selection process (through scoping) has been completed. In order to apply the "where" clause at the last step only (like in CMIS), a query can be split as in the example below:

```
LET S1 = BASE.(r1.r2) [3..5] where (attr1
   = 6)  (filter applied at each step)
LET S1 = BASE.(r1.r2) [3..5]  (no filter
   applied)
    S2 = S1 where (attr1 = 6)  (filter
   applied to the final set)
```

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 no new objects are
   found. Only the objects having "attr1=3"
   at each step of the graph traversal are
   retrieved and stored in S1 (Fig. 8).
```

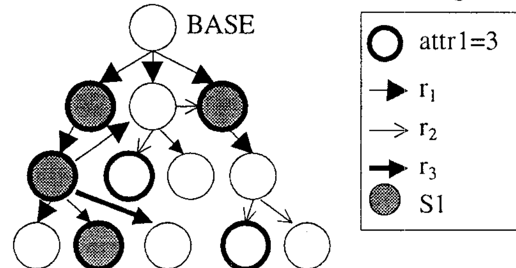

Fig. 8. S1 = BASE.(r1) [1..n] where (attr1 = 3)

```
LET S1 = BASE.(r1.r2) [1..n] where (attr1
   = 3)  (the sequence of relationships
   "r1.r2" is followed. The search
   continues until the sequence cannot be
   followed any more. The target objects
   are the ones having "attr1 = 3",
   retrieved via r2 from any level (Fig.
   9).
```

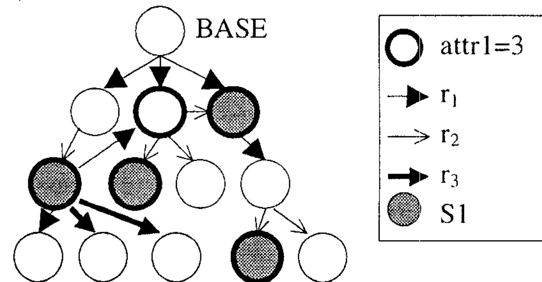

Fig. 9. S1 = BASE.(r1) [1..n] where (attr1 = 3)

Fringe Objects

Using recursion, *fringe* objects in a graph or sub-graph can be extracted. Objects are termed "fringe objects" either when a relationship - or sequence of relationships - can be no longer followed or when an object belongs to the extremes of the chosen sub-graph. 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)
```

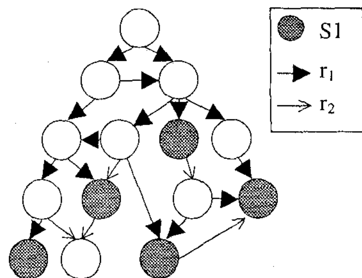

Fig. 10. S1 = BASE.(r1)! [1..n]

```
LET S1 = BASE.(r1)! [1..n] where (attr1 =
   6)  (like in the previous example. The
   "where" option is evaluated at each step
   of navigation)
LET S1 = BASE.(r1)! [1..3]  (objects at 1st
   and 2nd level missing the "r1" attribute;
   by definition all objects reachable at
   3rd level are fringe ones. Fig. 11)
```

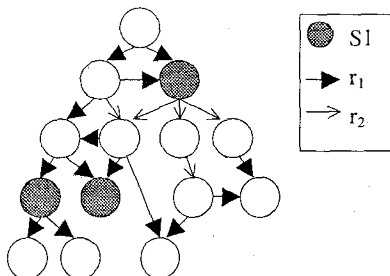

Fig. 11. S1 = BASE.(r1)! [1..3]

Scope and Filter in CMIS++

In the context of CMIS++, as we decided to traverse the object graph using qualified path expressions, the division between scope and filter selection mechanisms is no longer useful. We could choose to apply a final predicate (the filter) upon all the objects extracted through the graph traversal. However, a simpler way to work out this final selection is the use of a "where" clause, just like for the graph traversal.

In CMIS++ a single "**SELECTION**" clause is used. It is structured as follows:

- 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, difference among extents bound to the different variables.
- A final clause of the selection section ("RETURN") determines which objects, among the ones deposited in the set variables, have to be produced as output.

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

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

Containment in CMIS++

In CMIS, "containment" is the only relationship that can be used for object access. When a Managed Object is created its identifier is inserted in the Management Information Tree (MIT). This tree can be traversed using the scoping mechanism in order to select instances. Once an object is selected there is no way to retrieve either its "father" or its "sons" in the MIT. This feature represents a semantic limitation. In fact, although containment is useful to "order" object instances, the object semantic content itself is lost.

In CMIS++, the navigation of the containment relationship is implemented as follows: we add each object a couple of attributes: *contains* and *containedIn*. These attributes are implicitly defined for each object; they 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 father object in the tree. Accordingly, the *contains* attribute of the object father (or fathers) is updated too.

The *containedIn* attribute is written at the moment of the object creation; then, it can be accessed in read-only mode (likewise the case of "GET" access mode used in the GDMO attribute definitions). The *contains* attribute is a set-valued one with a special access clause: its value can be changed only when a new child object is inserted in the containment tree.

The use of the *contains* attribute allows for 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.

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 management action to find out the related relevant information. Such information can be obtained by querying the MIT. We show some CMIS queries suiting this purpose and their corresponding CMIS++ expressions.

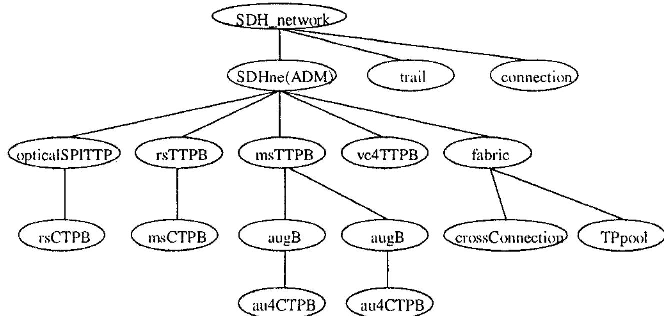

Fig. 12. SDH Containment Tree.

In our scenarios a network is built up with different paths, each one having 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, while Fig. 13 depicts the scenario. The object classes, packages and attributes used are defined in [11][12]. The fundamental ones are presented in [13].

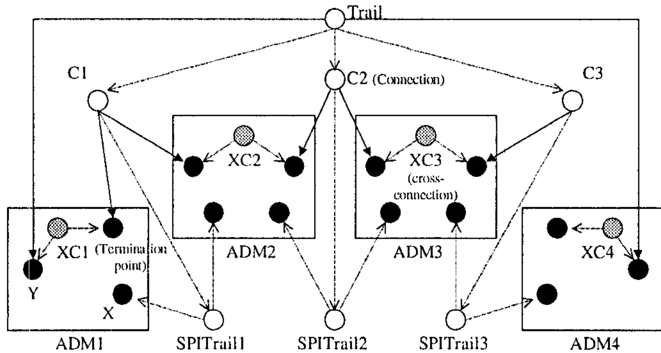

Fig. 13. SDH Scenario.

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 an M_EVENT_NOTIFICATION, as it usually occurs in the CMIS standard environment. An *event-report* 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; we suppose
  that C is a set of records, each one
  storing the trailID and its
  clientConnection. */
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
fiber 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 connections is 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 a single M_GET only. All the object DNs 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 event notification of the error comes from a *vc4TTPBidirectional*, marked with "Y" in Fig. 13. Two possible queries - in CMIS and CMIS++ respectively - aiming at detecting the *crossConnection* interested 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
crossConnection 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 inter-object 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 sub-optimal mapping is trivial while an optimal mapping that minimizes the CMIS/P interactions presents a difficult research problem. We do not present in this paper an algorithm for the mapping of CMIS++ to CMIS as we are rather concerned 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 managed objects within an agent by navigating their relationships. The query is evaluated by the agent and a number of replies are sent "back-to-back", in the form of a consolidated result. The bandwidth required to access this information is minimized while the overall latency is only slightly bigger that the latency incurred for 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 "proxy" unit 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 to 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. The structure of this parameter supports the CMIS++ features in two stages: *objectSelection* supports the core CMIS++ features while *objectSelection2* supports in addition 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 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, then a CMIS/P++ request with set expressions could be 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 but supports federation. The second, supported by the *objectSelection2* parameter does support set expressions but does *not* support federation if a request contains set expressions.

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 as overly complex and unimplementable due 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 reported in appendix 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 in Bristol 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 paper 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 is the case 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 be gradually 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 paper). 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++ can 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" managed objects. 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 [15], the relevant advantages being compatibility with CMIS/P and the disadvantages being performance due to the interpreted approach, 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 we will report our findings in the future.

Acknowledgements

This work described in this paper has been sponsored by Hewlett Packard and involved a collaboration among the HP Research Laboratory in Bristol, UK, University College London, UK, and Politecnico di Milano, Italy. We would like to acknowledge in particular Keith Harrison and Michele Campriani, both of HP research labs. Keith contributed the idea of set expressions (which add power and expressiveness at the expense of federation!) and Michele the SDH example.

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, Version 2, 1991.

[4] ITU-T Rec. X.710, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2, 1991.

[5] J.Case, M.Fedor, M.Schoffstall, 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.732, Information Technology - Open Systems Interconnection - Structure of Management Information - Guidelines for the Definition of Managed Objects, 1992.

[8] ITU-T Rec. X.720, Information Technology - Open Systems Interconnection - Structure of Management Information - General Relationship Model, 1992.

[9] G.Pavlou, K.McCarthy, S.Bhatti, G.Knight, *The OSIMIS Platform: Making OSI Management Simple*, Integrated Network Management IV, ed. A.Sethi, Y.Raynaud, F.Faure-Vincent, pp. 480-493, Chapman & Hall, 1995.

[10] G.Pavlou, Implementing OSI Management, Tutorial presented in the 3rd IFIP/IEEE International Symposium on Integrated Network Management, San Francisco, 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, S.Ceri, CMIS++: An Extension to CMIS for Accessing Telecommunication Databases, Proceedings of the IFIP/IEEE Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, 1996.

[14] ITU-T Rec. X.208, Open Systems Interconnection - Model and Notation -Specification of Abstract Syntax Notation One (ASN.1), 1988.

[15] A.Vassila, G.Pavlou, G.Knight, *Active Objects in TMN*, Integrated Network Management V, ed. A.Lazar, R.Saracco, R.Stadler, pp. 139-150, Chapman & Hall, 1997.

Appendix: 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 signifies "all subsequent levels"
   }

END

ObjectSelection2 DEFINITIONS ::=
BEGIN
IMPORTS PathExpression FROM ObjectSelection

ObjectSelection2 ::= SEQUENCE OF
                     PathOrSetExpression
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
```