Appendices

Appendix A: Work Based on the Proposed TMN Development Environment

The object-oriented software architecture for the TMN development environment proposed in this thesis was validated through the design and implementation of the OSIMIS platform. Since the latter is a generic development environment, the validation of the proposed architecture and the software platform itself was ultimately achieved through the design, implementation and deployment of a number of TMN systems in various research projects. In fact, the existence of such a development environment both stimulated and enabled additional research in this area. In this appendix we present briefly research and development work based on the proposed development environment.

A.1 Research and Development Work Involving the Author

The very first application developed using OSIMIS was an agent for managing the ISO/ITU-T Transport Protocol [TPMIB]. This was developed by the author and was a re-engineering of the same application that had been previously developed by S. Walton of UCL. This particular application validated the early version of the OSIMIS Generic Managed System. It used to be part of OSIMIS releases until version 3.3 and served as an example of relatively sophisticated use of the agent part of the infrastructure.

The next application developed with OSIMIS was the generic MIB browser [Pav92a], which has since become part of OSIMIS. This was an important manager application that highlighted necessary support aspects for generic manager applications. It also highlighted the need for high-level manager infrastructure and led to the specification and embryonic implementation of the OSIMIS RMIB. The MIB browser was developed by J. Cowan of UCL.

During the summer of 1991, a MSc student group project produced a first version of an agent for managing the OSI version of the Internet SNMP MIB-II [OIM]. This was taken further after one of the students stayed at UCL as a research associate after his MSc. This application stretched

aspects of the agent infrastructure since it involved extensive interaction with the UNIX kernel. It was developed by S. Bhatti and was included in OSIMIS versions prior to 4.0.

During the summer of 1992, an MSc student group project produced a "proxy" implementation of the OSI version of the Internet SNMP MIB-II [OIM]. This was based on a non-generic model but highlighted the possibilities for a generic CMIS/P-SNMP proxy agent. One of the students, K. McCarthy, stayed at UCL as a research associate after his MSc and was involved in the research, design and implementation of a generic proxy agent [McCar95]. This is known as the Internet Q-Adapter (IQA) and is part of the OSIMIS 4.0 distribution. J. Reilly of VTT designed and implemented the supporting SNMP SMI to GDMO MIB translator.

During the summer of 1994, an MSc student group project produced a comprehensive local area network monitoring tool. This reproduced most of the functionality of the SNMP Remote Monitoring MIB [RMON] by re-using the OSI SMFs supported by OSIMIS. It demonstrated the use of generic infrastructure to simplify the task of producing complex applications such as a RMON agent.

A PhD student, N. Vassila, investigated the applicability of the "management by delegation" paradigm [Yemi91] in TMN environments. She used the Tcl scripting language to introduce Active Managed Objects (AMOs) in the OSIMIS agent framework. Such MOs can be dynamically downloaded to agent applications [Vass95][Vass97].

The first hybrid manager-agent TMN OSs were developed in the RACE NEMESYS project as part of a TMN system for ATM Quality of Service management. An Element Manager OS and a Service Manager OS were developed, managing a simulated ATM network through a relevant Q-Adapter. The architecture of this system is described in [Pav91b] while its modelling, design and the implementation experiences are described in [Pav92b]. This was the first hierarchical TMN system with complete Q_3 interfaces. The Q-Adapter verified the scalability of the agent infrastructure since it contained more than 10000 managed objects for the simulated ATM network that was used in the experiments.

OSIMIS was used in a large scale in the RACE ICM project. The project's TMN system is described in detail in [Gri95][Gri96b][ICM] and provided ATM Virtual Path Connection and Routing Management (VPCRM) services and also ATM-based Virtual Private Network (VPN) services. This was the most complex TMN system built at the time, comprising twelve different types of TMN Operations Systems (OSs). These could be instantiated in different domains, providing end-to-end VPN services and intra-domain VPCRM services. The system operated over both real and simulated networks. It was developed mostly by researchers in different

companies and research institutions around Europe who had little or no exposure to network programming. It demonstrated the suitability of the proposed framework for the rapid development of sophisticated TMN systems.

Various research topics were addressed by the ICM project as part of its TMN activities. [Georg95] proposes an approach to realising management services for performance verification in multi-service ATM networks. These have a minimal impact on the managed network by delegating performance management activities to the network elements. This is achieved through the use of the OSI Metric Monitoring [X739], Summarisation [X738] and the Intelligent Monitoring Function (IMF) [Pav96c]. The latter was devised, designed and developed in the ICM project. It combines the power and expressiveness of the metric monitoring and summarisation functions and allows a manager application to form complex expressions of monitored attributes and delegate them to the agent. The IMF support objects were implemented by G. Mykoniatis of the National Technical University of Athens (NTUA).

An important aspect of the ICM project was the demonstration of TMN security services, mainly inter-domain across the X interface but also intra-domain. A lightweight approach was adopted for the authentication, stream-integrity and confidentiality services, based on symmetric, secret-key based encryption and signing functions. The security architecture, transformations and programmatic access aspects are described in [Bhat96]. S. Bhatti and G. Knight of UCL were the major architects of the solution while K. McCarthy implemented a major part of it. In addition, the author designed and implemented the object-based access control function together with T. Tin of UCL. The lightweight security and access control functions are part of the OSIMIS 4.1 version.

TINA applications based on the CORBA-based version of OSIMIS were used in the ACTS VITAL and REFORM projects for Resource Configuration Management (RCM), as described in [Pav97b]. These were TMN-like applications holding network topology information through managed objects with various relationships, according to the TINA Network Resource Information Model (NRIM) [NRIM]. Access to those was provided through the CORBA-based "Q_3" management broker interface that was described in Chapter 4. These applications were developed by the author and T. Tin of UCL.

A.2 Research and Development Work by Others

OSIMIS was used in a number of other RACE, ACTS and ESPRIT projects, in addition to the NEMESYS, ICM, VITAL and REFORM projects mentioned above.

In the RACE PREPARE project it was used in addition to a number of commercial platforms for developing TMN applications, as described in [Lewis95]. Parts of it were also used to support the PREPARE Inter-Domain Management Information Service (IDMIS) [Bjer94] platform. It is also currently being used in the ACTS PROSPECT project, which is the continuation of PREPARE. PROSPECT uses the Tcl-RMIB infrastructure [Tirop97]. OSIMIS is also being used in the ACTS MISA project.

In the RACE TOMQAT project it was used for developing quality of service management applications [Lioup94]. In the RACE BAF project it was used for developing Q-Adapter TMN applications. It is also currently being used in the ACTS BONAPARTE project, which is the continuation of BAF. There it is used for developing TMN NE agents for ATM switching equipment [I751].

In the ESPRIT PROOF project it was used to develop an OSI-based management system for a primary rate ISDN to Internet IP gateway. In the ESPRIT MIDAS project it was used to develop applications for managing OSI X.500 directory systems and X.400 mail systems. It was also used as a vehicle to do research in and develop asymmetric public-key based security services as described in [Bhat95]. In the ESPRIT IDSM project it was used to implement intelligent monitoring policies.

OSIMIS has also been used by researchers all around the world in various research projects. Related research work is presented below, based mainly on publications in the IFIP/IEEE Integrated Management (IM) Symposium, the IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM), the Intelligence in Services & Networks (IS&N) conference and the Journal of Network and Systems Management (JNSM).

[Filip93] describes an experimental agent structure for migrating IBM's SNA-based network management aspects to OSI-SM environments. The agent translates between SNA management information and protocols to GDMO and CMIS/P and was based on IBM's adaptation of an early version of OSIMIS.

[Jord93] describes a model and associated prototype for alarm correlation, which tries to identify the faulty resource with high probability. The prototype was developed at IBM ENC and was based on IBM's adaptation of an early version of OSIMIS.

[Ditt95] describes the ANDROMEDA platform (Analysis, Development and Provision of a Management Environment for Distributed Applications), which was developed by GMD Fokus as part of their BERKOM project. It is a collection of tools and environments which reuses the OSIMIS agent infrastructure and MSAP manager API together with the ISODE QUIPU DSA [QUIPU]. It also provides a uniform access interface to both management and directory information known as IDMIS - Inter-Domain Management Information Service [Bjer94].

[Sartz95] describes the design and implementation of a meta-management system for managing the TMN itself. This includes a system for placing the various TMN applications in particular network nodes, monitoring their activity and potentially relocating them for performance of fault reasons. The relevant prototype system was based on OSIMIS.

[Perr95] proposes a model and an associated prototype tool for automating the development of management agents based on different frameworks e.g. OSI-SM and SNMP. The architectural decomposition of the generic agent is largely based on the OSIMIS GMS which also served as the basis for the relevant prototype.

[Katch95] describes work towards the enhancement of GDMO with behavioural specifications for automating the detection of pre-conditions that will result in the generation of a notification. GDMO augmentations have been proposed and the OSIMIS GDMO compiler and generic agent infrastructure have been modified to support those.

[Sidou95] describes an information model simulator that supports the testing of a newly developed GDMO model. This simulates the behaviour of managed objects and uses the OSIMIS agent infrastructure and support tools.

[Mazz96] describes extensions both to the agent and manager parts of OSIMIS that allow the realisation of interpreted managed and managing objects in the Scheme language. The goal is to achieve rapid prototyping of TMN applications by avoiding the compile and link cycle.

[Aner96] describes an architecture and experimental platform for the management of ATM Virtual Path Connections (VPCs). The relevant development was based on OSIMIS, while the author developed his own generic MIB browser using X-Windows Motif technology.

[Korm96] describes issues behind the design and implementation of the OSI Usage Metering SMF, using the relevant tools and facilities provided by OSIMIS. It also describes an associated case study for accounting CPU usage in a distributed workstation environment.

[Park96] describes the development of a Shared Management Knowledge (SMK) system using two different approaches, one based on OSIMIS and one based on CORBA. It then presents a comparative performance evaluation of the two approaches, concluding that the CORBA implementation is relatively faster.

[Nataf97] investigates the issues behind GRM-based relationship representation [X725] and proposes a prototype as a first step towards a complete support tool for relationship management. The relevant prototype is based on OSIMIS.

[Ramo97] proposes a novel approach towards providing access control services [X741] in OSI-SM/TMN environments by exploiting the use of allomorphism. The relevant experimentation was based on OSIMIS.

[Seitz97] investigates the use of a meta managed object specification language that can be mapped onto specific languages and access protocols. The OSIMIS uxObj example class is used to demonstrate the features of the meta-language while the OSIMIS environment has been used for the relevant prototype.

It should be finally added that OSIMIS has been used by more than one hundred companies and research institutions around the world, whose research and development work may have not resulted in publications.

Appendix B: The Object Modelling Technique Notation

The Object Modelling Technique (OMT) notation [Rumb91] has been used throughout this thesis for describing object class relationships. Figure B-1 depicts the OMT notation for the relationships used in this thesis.

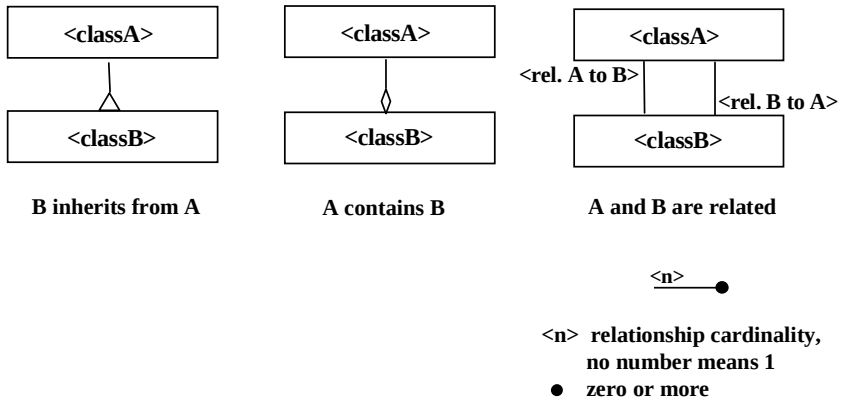

Figure B-1 OMT Notation for Inheritance, Containment and Other Relationships

Appendix C: Specification of the Managed Object Classes Used in the Examples

In Chapters 3 and 4, the *uxObj* and *simpleStats* object classes were used in the examples to demonstrate aspects the proposed infrastructure. Their complete specification is included in this appendix, first in OSI-SM GDMO/ASN.1and then in CORBA IDL. The IDL specifications have been derived from the GDMO/ASN.1 ones, following the JIDM guidelines [JIDM95]. It should be noted an OSI-SM implementation of those classes is part of the OSIMIS-4.0 distribution [Pav95b].

C.1 Specification in GDMO/ASN.1

In GDMO/ASN.1 specifications, a set of GDMO templates [X722] specify first the object-oriented aspects of the managed object class. An ASN.1 [X208] module follows, specifying the types of the attributes, actions, notifications and the object identifiers used when registering the various entities in the GDMO specification.

The root of the GDMO inheritance hierarchy is the *top* class [X721], which is not included in this specification. Both the uxObj and simpleStats classes are positioned in the MIT under instances of the *system* class [X721], which is the MIT root. The system class is not included in this specification.

Note also that the *gauge* attribute and the *objectCreation*, *objectDeletion* and *attributeValueChange* notifications and their syntaxes are imported from the Definition of Management Information recommendation [X721], so they are not included in the specification.

```
-- THE uxObj CLASS


-- Class and Package Templates

uxObj MANAGED OBJECT CLASS
DERIVED FROM        top;
CHARACTERIZED BY    uxObjPackage;
REGISTERED AS       { uclManagedObjectClass 50 };

uxObjPackage PACKAGE
ATTRIBUTES
    uxObjId                 GET,
    sysTime                 GET,
    wiseSaying              GET-REPLACE REPLACE-WITH-DEFAULT
                            DEFAULT VALUE
                            UCL-ASN1Module.defaultUxObjWiseSaying,
    nUsers                  GET;
ACTIONS
    echo;
NOTIFICATIONS
    objectCreation,
    objectDeletion,
    attributeValueChange;    -- when setting the wiseSaying attribute
REGISTERED AS       { uclPackage 50 };


-- Name Binding Template, positions uxObj instances in the MIT

uxObj-system NAME BINDING
SUBORDINATE OBJECT CLASS    uxObj AND SUBCLASSES;
NAMED BY
SUPERIOR OBJECT CLASS       system AND SUBCLASSES;
WITH ATTRIBUTE              uxObjId;
CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
DELETE                      ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS               { uclNameBinding 50 };


-- Attribute Templates

uxObjId ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.SimpleNameType;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 501 };

sysTime ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.Time;
MATCHES FOR EQUALITY, ORDERING;
REGISTERED AS       { uclAttributeID 502 };

wiseSaying ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.String;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 503 };

nUsers ATTRIBUTE
DERIVED FROM X721:gauge;
REGISTERED AS       { uclAttributeID 504 };


-- Action Templates

echo ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.String;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.String;
REGISTERED AS       { uclAction 501 };
```

```
-- THE simpleStats CLASS


-- Class and Package Templates

simpleStats MANAGED OBJECT CLASS
DERIVED FROM        top;
CHARACTERIZED BY    simpleStatsPackage;
REGISTERED AS       { uclManagedObjectClass 70 };

simpleStatsPackage PACKAGE
ATTRIBUTES
    simpleStatsId   GET;
ACTIONS
    calcSqrt,
    calcMeanStdDev;
REGISTERED AS       { uclPackage 70 };


-- Name Binding Template, positions uxObj instances in the MIT

simpleStats-system NAME BINDING
SUBORDINATE OBJECT CLASS    simpleStats AND SUBCLASSES;
NAMED BY
SUPERIOR OBJECT CLASS       system AND SUBCLASSES;
WITH ATTRIBUTE              simpleStatsId;
CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
DELETE;
REGISTERED AS               { uclNameBinding 70 };


-- Attribute Templates

simpleStatsId ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.SimpleNameType;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 701 };


-- Action Templates

calcSqrt ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.Real;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.Real;
REGISTERED AS       { uclAction 701 };

calcMeanStdDev ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.RealList;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.MeanStdDev;
REGISTERED AS       { uclAction 701 };
```

```
-- THE ASN.1 MODULE

UCL-ASN1Module DEFINITIONS ::=

BEGIN


uclManagedObjectClass OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 3}
uclNameBinding OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 6}
uclPackage OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 4}
uclAttributeID OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 7}
uclAction OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 9}
uclNotification OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 10}


-- SimpleNameType could have been imported from the X.721 ASN.1 module
-- but is included here to demonstrate use of the CHOICE ASN.1 type and
-- its translation to CORBA IDL in the next section

SimpleNameType ::= CHOICE
{
    num     INTEGER,
    str     GraphicString
}


-- GraphicString and UTCTime are base ASN.1 types

String ::= GraphicString

Time ::= UTCTime


-- REAL is a base ASN.1 type

RealList ::= SET OF REAL

MeanStdDev ::= SEQUENCE
{
    mean    REAL,
    stdDev  REAL
}

-- default values

defaultUxObjWiseSaying String ::= “Hello World”
END
```

C.2 Specification in CORBA IDL

The same specifications are included here in CORBA IDL, based on the JIDM translation rules. The translation follows the spirit rather than the letter of the rules. For example, no exceptions have been associated with attribute access methods (they are unnecessary).

The nUsers attribute is of gauge type which has associated ObservedValue ASN.1 syntax [X721]. This maps to the ObservedType_t IDL type which is not explicitly included in this specification. Note also that the notification interface for the uxObj class is not included.

The *i_uxObj* and *i_simpleStats* interfaces inherit from *i_top* one, which results from translating the GDMO *top* class [X721]. The i_top interface has been depicted in the Code 4-1 caption in Chapter 4 and inherits subsequently from the*i_ManagedObject* interface.

```
// THE i_uxObj INTERFACE

// the argument types for operations are required first in IDL,
// in a similar fashion to programming languages

// SimpleNameType: integer or string

enum SimpleNameTypeChoice {
    numberChoice,
    stringChoice
};

union SimpleNameType_t switch (SimpleNameTypeChoice) {
    case numberChoice: long   num;
    case stringChoice: string str;
};

typedef string UTCTime_t; // string with well-defined format

// note there is no name binding in CORBA

// the invalidArgumentValue exception is part of the
// _ManagedObject interface and is defined as follows:
//
// exception invalidArgumentValue {
//     any argumentValue
// };


interface i_uxObj : i_top
{
    SimpleNameType_t      uxObjId_get ();
    UTCTime_t             sysTime_get ();
    string                wiseSaying_get ();
    void                  wiseSaying_set (in string);
    string                wiseSaying_setDefault ();
    ObservedValue_t       nUsers_get ();
    void                  echo (in string, out string)
                              raises (invalidArgumentValue);
                              // never raised
};


// the notifications are part of a separate interface
```

```
// THE i_simpleStats INTERFACE

// SimpleNameType_t has been previously defined

typedef sequence<real> RealList_t;

typedef struct MeanStdDev_t
{
    real     mean;
    real     stdDev;
}


interface i_simpleStats : i_top
{
    // invalidArgumentValue is a standard CMIS action error which
    // is mapped to an IDL exception - it should be normally defined
    // as part of the i_ManagedObject interface


    SimpleNameType_t      simpleStatsId_get ();
    void                  calcSqrt (
                                in real, out real
                          ) raises (invalidArgumentValue);
    void                  calcMeanStdDev (
                                in RealList_t,
                                out MeanStdDev_t
                          ) raises (invalidArgumentValue);
                          // never raised
};
```

According to the JIDM rules, every action method with an *in* argument may raise an invalidArgumentValue exception. While this is the case for the calcSqrt method, e.g. for a negative number, this exception is never raised for the calcMeanStdDev method. The same is true for the i_uxObj echo method. In short, automatic translation cannot exploit semantics associated with the GDMO object class.

Appendix D: A String Language for CMIS Filters

A CMIS filter [X711] is a recursive ASN.1 type that supports tree-like expressions, with attribute value assertions in the leaves associated with boolean operators. The precise specification in ASN.1 is the following:

```
CMISFilter ::= CHOICE {
    item     [8]  EXPLICIT FilterItem,
    and      [9]  IMPLICIT SET OF CMISFilter,
    or       [10] IMPLICIT SET OF CMISFilter,
    not      [11] EXPLICIT CMISFilter
}

FilterItem ::= CHOICE {
    equality        [0] IMPLICIT Attribute
    substrings      [1] IMPLICT SEQUENCE OF CHOICE
      initialString       [0] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
      anyString           [1] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
      finalString         [2] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
    greaterOrEqual  [2] IMPLICIT Attribute,
    lessOrEqual     [3] IMPLICIT Attribute,
    present         [4] AttributeId,
    subsetOf        [5] IMPLICIT Attribute,
    supersetOf      [6] IMPLICIT Attribute,
    nonNullSetIntersection
                    [7] IMPLICIT Attribute
}
```

In the above specification, an attribute comprises an attribute ID and attribute value as defined in [X711]. When mapping this type to a C++ class through the O-O ASN.1 compiler, the user has to “fill it in” with the attributes, values and boolean operators. This is a both tedious and error-prone procedure, especially for complex filter expressions. A string-based user-friendly “language” can be used, which can be parsed to create the associated C++ object instance. The OSIMIS string-based language for CMIS filters is described semi-formally below.

A filter expression is always (<cmisFilter>) where <cmisFilter> is one of: <andFilter>, <orFilter>, <notFilter> or <filterItem>. The characters used to represent the logical operators are: & for AND, | for OR and ! for NOT. The <andFilter>, <orFilter>, <notFilter> and <filterItem> are as follows:

- < andFilter> has the form: ((<cmisFilter>) & (<cmisFilter>) …);
- < orFilter> has the form: ((<cmisFilter>) | (<cmisFilter>) …);
- <notFilter> has the form: (!(<cmisFilter>); and
- <filterItem> has one of two forms:
    1. <attributeName>) for creating a filter item with the "present" assertion; and
    2. (<attributeName><assertionType><attributeValue>) for all the other assertion types.

Table D-1 shows the lexemes used for the filter item assertions. With the substrings operator, the * character can be used as a wild card, in a similar fashion to the UNIX shell. The attribute values should follow the string representations according to the print method for that ASN.1 type.

Lexeme	Assertion Type
=	equality
:=	substrings (* is the wild card)
>=	greater or equal
<=	less of equal
:<	subset of
:>	superset of
><	non-null set intersection

Table D-1 Lexemes Used for CMIS Filter Assertions

Examples of filter expressions are:

```
((objectClass=log) & (administrativeState=unlocked))
((objectClass=routeEntry) & (nextHopAddr=X)
((wiseSaying=*ello*) | (!(wiseSaying=*ello), (nUsers >= 10)
```

In this way, sophisticated filter expressions can be easily constructed. This string notation was devised by the author together with S. Bhatti of UCL who implemented the parser.

Appendix E: Lightweight CMIS/P Specification

The concepts behind the Lightweight CMIP (LCMIP) were described in section 3.3.2.4 of Chapter 3. The specification of the GetArgument and GetResult LCMIP types is include below in order to demonstrate the simplifications compared to CMIP [X711]. These simplifications are briefly discussed after the specification.

```
GetArgument ::= SEQUENCE {
    baseManagedObjectClass       ObjectClass,
    baseManagedObjectInstance    ObjectInstance,
    accessControl                AccessControl,
    synchronization              CMISSync,
    scope                        Scope,
    filter                       CMISFilter,
    attributeIdList              AttributeIdList
}

GetResult ::= SEQUENCE {
    managedObjectClass           ObjectClass,
    managedObjectInstance        ObjectInstance,
    currentTime                  GeneralizedTime,
    getAttributeList             SET OF GetAttribute
}

GetAttribute ::= SEQUENCE {
    attributeId     AttributeId,
    attributeValue  AttributeValue,
    errorStatus     ENUMERATED {
                           noError (0),
                           accessDenied (2),
                           noSuchAttribute (5)
                    }
}

CMISSync ::= ENUMERATED {        -- same as in [X711]
    bestEffort (0),
    atomic (1)
}

Scope ::= SEQUENCE {             -- simplified version of [X711]
    type     ENUMERATED {
                 baseObject (0),
                 firstLevelOnly (1),
                 wholeSubtree (2),
                 individualLevel (3),
                 baseToNthLevel (4)
             },
    level    INTEGER
}

CMISFilter ::= GraphicString     -- the OSIMIS string notation

ObjectClass ::= GraphicString    -- textual name
ObjectInstance ::= LCMIPDN       -- global / local distinction
                                 -- through agreed prefixes
LCMIPDN ::= GraphicString        -- OSIMIS/ISODE convention
                                 -- e.g. logId=1@logRecordId=5
AccessControl ::= OCTET STRING
GeneralizedTime ::= GraphicString -- with internal structure
AttributeId ::= GraphicString    -- textual name
AttributeValue ::= GraphicString -- the pretty-printed string

GraphicString ::= OCTET STRING   -- for easy encoding/decoding
```

As it can be seen from the specification, LCMIP introduces a number of important simplifications. The ObjectClass and AttributeId types are string names instead of OIDs. The ObjectInstance is a string with internal structure in the form used in OSIMIS/ISODE. The same is the case with the CMISFilter which is a string according to the conventions described in Appendix D. The Scope is a simplified version of the CMIP Scope which avoids the ASN.1 CHOICE type. The AttributeValue is a well-agreed pretty-printed string representation for a particular type. Finally, the GraphicString type is encoded as ASN.1 OCTET STRING for the ease of encoding / decoding.

In summary, LCMIP is a lightweight version of CMIP that uses strings as much as possible and avoids tags, optional elements and other ASN.1 aspects which complicate encoding and decoding. It should be possible to implement LCMIP by hand, without the need for ASN.1 compilers which inevitably introduce inefficiencies.