International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.782

(05/2012)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

OSI management – Management functions and ODMA functions

Guidelines for defining web services for managed objects and management interfaces

Recommendation ITU-T X.782

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems management framework and architecture	X.700–X.709
Management communication service and protocol	X.710–X.719
Structure of management information	X.720–X.729
Management functions and ODMA functions	**X.730–X.799**
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, concurrency and recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	X.1000–X.1099
SECURE APPLICATIONS AND SERVICES	X.1100–X.1199
CYBERSPACE SECURITY	X.1200–X.1299
SECURE APPLICATIONS AND SERVICES	X.1300–X.1399
CYBERSECURITY INFORMATION EXCHANGE	X.1500–X.1599

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.782

Guidelines for defining web services for managed objects and management interfaces

Summary

Recommendation ITU-T X.782 defines a set of guidelines for managed object modelling and a management interface for web services-based network management. It composes a framework for web services-based network management interfaces along with Recommendation ITU-T Q.818. It specifies how service-oriented web service interfaces should be defined. It covers the suitable application scenario of web services in network management interfaces, generic accessing methods of XML-based managed objects, information modelling in web service WSDL and XML schema. Some WSDL definitions and XML schema are provided for defining some basic data types: generic managed object (MO) and generic MO accessing methods. This Recommendation and Recommendation ITU-T Q.818 together compose a framework for web services-based network management interfaces with a wide range of applications.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T X.782	2012-05-14	2

Keywords

Distributed processing, extensible markup language (XML), managed objects, network management interfaces, web service (WS), web services description language (WSDL), XML schema.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

		Page
1	Scope	1
2	References	1
3	Definitions	2
	3.1 Terms defined elsewhere	2
	3.2 Terms defined in this Recommendation	2
4	Abbreviations and acronyms	2
5	Conventions	3
6	Overview of a web services-based management framework	4
7	Principles for service-oriented WSDL-based interface design	4
8	Definition of a generic managed object using XML schema	5
	8.1 Web-service role in management interfaces	5
	8.2 Definition of managed objects using XML schema	6
9	Accessing methods for managed objects	9
10	Inheritance of managed objects and interface operations	11
	10.1 Attributes inheritance of managed objects	11
	10.2 Considerations for the inheritance of interface operation	12
11	Information modelling guidelines for web services-based interfaces	12
	11.1 namespace	12
	11.2 complexType	12
	11.3 attribute	12
	11.4 request	12
	11.5 response	13
	11.6 notification	13
	11.7 Name conventions for MOCs, packages, attributes and data types	13
12	Style idioms for XML schema specifications	14
	12.1 Data model using XML schema	14
	12.2 XML schema design considerations	14
	12.3 Recommendations for schema developers	16
	12.4 Guidelines for schema extensions	18
13	Compliance and conformance	18
	13.1 Standards document compliance	19
	13.2 System conformance	19
	13.3 Conformance statement guidelines	19
Annex A – Common WSDL and XML schema definitions		20
	A.1 XML schema definitions for common data types and a generic managed object	20

Page

A.2 WSDL and XML schema definition for common object accessing methods 28

Appendix I – Overview of web service technology and application scenarios in network management interfaces 34

I.1 Characteristics of web service technology 34

I.2 Suitable and unsuitable application scenarios of web services in network management 35

Bibliography 37

Recommendation ITU-T X.782

Guidelines for defining web services for managed objects and management interfaces

1 Scope

The network management architecture defined in [ITU-T M.3010] introduces the use of multiple management protocols. So far, the GDMO/CMIP and CORBA GIOP/IIOP are possible choices at the application layer. Based on the management interface specification methodology defined in [ITU-T M.3020], more technology-based paradigms can be introduced into network management interfaces and web service/XML is now an additional paradigm for network management.

This Recommendation, together with [ITU-T Q.818] sets out to define a framework for defining how interfaces supported by management systems and network elements should be modelled using web service/XML schema. It is within the scope of this Recommendation to provide the following guidelines or instructions:

– service-oriented WSDL-based interface design approach;

– suitable and unsuitable application scenarios for web services in network management interfaces;

– generic accessing methods for managed objects;

– inheritance of managed objects and interfaces;

– information modelling guidelines for web services-based interfaces;

– style conventions for web service WSDL and XML schema specifications.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T E.164] Recommendation ITU-T E.164 (2010), *The international public telecommunication numbering plan*.

[ITU-T M.3010] Recommendation ITU-T M.3010 (2000), *Principles for a telecommunications management network*.

[ITU-T M.3020] Recommendation ITU-T M.3020 (2011), *Management interface specification methodology*.

[ITU-T M.3701] Recommendation ITU-T M.3701 (2010), *Common management services – State management – Protocol neutral requirements and analysis*.

[ITU-T Q.818] Recommendation ITU-T Q.818 (2012), *Web service-based management services*.

[ITU-T X.701] Recommendation ITU-T X.701 (1997), *Information technology – Open Systems Interconnection – Systems management overview*.

[ITU-T X.703] Recommendation ITU-T X.703 (1997), *Information technology – Open Distributed Management Architecture*.

[ATIS-I-000002] ATIS Specification ATIS-I-000002 (2011), *ATIS XML Schema Development Guidelines*.

[OASIS WSN] OASIS Specification (2006), *Web Services Base Notification v1.3*.

[OASIS UDDI] OASIS Specification (2004), *Universal Description, Discovery and Integration (UDDI) v3.0.2*.

[W3C Primer] W3C Recommendation (2004), *XML Schema Part 0: Primer Second Edition.*

[W3C SOAP] W3C Recommendation (2007), *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).*

[W3C WSDL] W3C Recommendation (2001), *Web Services Description Language (WSDL) 1.1*.

[W3C XML] W3C Recommendation (2000), *Extensible Markup Language (XML) 1.0 Second Edition*.

[W3C XS-P1] W3C Recommendation (2004), *XML Schema Part 1: Structures Second Edition*.

[W3C XS-P2] W3C Recommendation (2004), *XML Schema Part 2: Datatypes Second Edition*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 agent [ITU-T M.3020]

3.1.2 managed object class [ITU-T X.701]

3.1.3 manager [ITU-T M.3020]

3.1.4 notification [ITU-T X.703]

3.2 Terms defined in this Recommendation

This Recommendation does not define any new terms.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

B2B Business to Business

BPEL Business Process Execution Language

C2B Customer to Business

CMIP Common Management Information Protocol

CORBA Common Object Request and Broker Architecture

DCOM Distribute Component Object Model

DN Distinguished Name

DTD Document Type Definition

EMS Element Management System

GDMO Guidelines for the Definition of Managed Objects

GIOP	General Inter-ORB Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IT	Information Technology
LAN	Local Area Network
MO	Managed Object
MOC	Managed Object Class
MOO	Multiple Object Operation
NMS	Network Management System
OOAD	Object-Oriented Analysis and Design
OS	Operating System
RDN	Relative Distinguished Name
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TMN	Telecommunications Management Network
UDDI	Universal Description Discovery and Integration
WS	Web Services
WSDL	Web Services Description Language
WSN	Web Services Notification
XML	extensible Markup Language
XSD	XML Schema Definition

5 Conventions

A few conventions are followed in this Recommendation to make the reader aware of the purpose of the text. While most of the Recommendation is normative, paragraphs succinctly stating mandatory requirements to be met by a management system (managing and/or managed) are preceded by a boldface "R" enclosed in parentheses, followed by a short name indicating the subject of the requirement, and a number. For example:

(R) EXAMPLE-1 An example mandatory requirement.

Requirements that may be optionally implemented by a management system are preceded by an "O" instead of an "R". For example:

(O) EXAMPLE-2 An example optional requirement.

The requirement statements are used to create compliance and conformance profiles.

Examples of WSDL and XML are included in this Recommendation and normative WSDL and XML specifying the data types, base classes and other service-oriented modelling constructs of the framework are included in Annex A. The WSDL and XML are written in a 10 point courier typeface:

```
<!-- Example XML schema -->
<xsd:complexType name="AType">
   <xsd:sequence>
      <xsd:element name="a" type="xsd:string"/>
      <xsd:element name="b" type="xsd:long"/>
   </xsd:sequence>
</xsd:complexType>
```

6 Overview of a web services-based management framework

Web services have been widely used in the IT industry. Appendix I provides more information on the features of web services technology. Web services similar to CORBA technology, can be used in network management interfaces.

This Recommendation together with [ITU-T Q.818] sets up a framework for defining how interfaces supported by management systems and network elements should be modelled using WSDL and XML schema.

The web services-based management framework includes the following aspects.

1) Managed object and interface definition guidelines:
   - principles for the service-oriented WSDL-based interface design;
   - definition of managed objects using XML schema;
   - accessing methods for MOs;
   - inheritance of MOs and interface operations;
   - information modelling guidelines for web services-based interface operations;
   - style idioms for WSDL and XML schema specifications.
2) Web services supporting services for network management:
   - definition of notification services using OASIS web services base notification [OASIS WSN];
   - usage of OASIS UDDI service registration [OASIS UDDI];
   - definition of a heartbeat service;
   - definition of a multiple object operations (MOO) service;
   - definition of a containment service.

This Recommendation mainly deals with the managed object and interface definition guidelines, and [ITU-T Q.818] mainly deals with the web services supporting services for network management. The two Recommendations together form a web services-based management framework.

7 Principles for service-oriented WSDL-based interface design

This clause identifies some interface design considerations that should be addressed by this framework through service-oriented interfaces. It provides the modelling principles for service-oriented managed objects and their accessing methods.

The service-oriented design considerations related to WSDL and XSD repertoire and modelling concerns super-classes, naming of managed objects and service-oriented interfaces, operations and notifications.

A web service is a service-oriented technology compared with the traditional CORBA/IDL and GDMO/CMIP management paradigms. An object-oriented interface analysis and design (OOAD) focuses on the class level, i.e., it encapsulates behaviour and related data in the same object which exposes one or more interfaces for access to their encapsulated states and attributes. The service-

oriented interface design separates the encapsulated data and state from the behaviour, thus loose coupling can be achieved at a higher level. In this approach, the behaviour of some (shared) objects is no longer controlled by themselves but by a few predefined interface operations.

This Recommendation, along with [ITU-T Q.818], defines a lightweight generic use of service-oriented interface design patterns. The management and controlling functions are defined using a service-grained approach, that is, interface operations are organized within the unit of a certain service (management function sets, such as configuration management, performance management, etc.), not an individual management object class. This service-orientation requires the flexibility of application-specific access granularity where well-defined sets of TMN entity types are accessed through predefined WSDL interfaces.

The framework has the following principles to define a service-oriented WSDL and XSD based management information model and interfaces.

– All interface interactions are defined as WSDL operations, each operation includes a request and an optional corresponding response when needed.
– Each MOC is defined as an XML complexType when exchanged through the management interface, and each attribute or state of the MOC is defined as an element in the complexType.
– The naming of MOC instances follows the DN concept, but it is a string containing all of the RDN list.
– A generic service is defined in this framework, which contains five generic object accessing methods: createMO, deleteMO, getMOAttribute, setMOAttribute, and getPackages. Also, the generic object accessing methods use "name-value" pairs to express the properties and their values of different types of MOC instances.
– Other interface control functions are defined as WSDL interface operations which are organized as a service with the granularity of management function sets.
– Common data types are defined as XML schemas which can be shared by application-specific interface definitions.
– Notifications sent from the agent to the manager should follow the format and behaviour defined in [OASIS WSN]. The control of notification management services are further defined in [ITU-T Q.818].

8 Definition of a generic managed object using XML schema

8.1 Web-service role in management interfaces

To support the software objects representing manageable resources, a base class is defined for use in modelling network resources. Other MOCs (managed object class) in information models must be derived from this base class in order to operate within this framework. Some generic accessing methods and some other extended services are defined to provide interfaces to manage MOs.

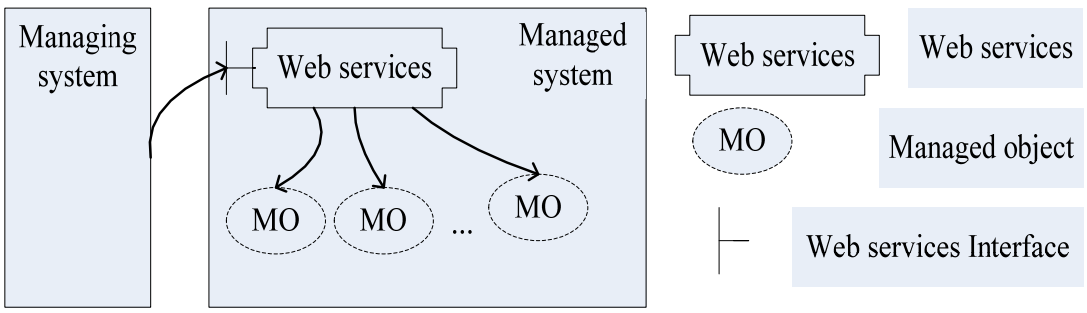

Figure 1 – Web services role

Figure 1 shows how a managing system accesses a managed system that supports a web services interface. A web services interface acts as an intermediate entity that enables a managing system to manage proper MOs in a managed system representing manageable resources.

8.2 Definition of managed objects using XML schema

An MO is the OSI management view of a resource that is subject to management, such as a connection or an item of physical equipment. Thus, an MO is the abstraction of such a resource that represents its properties for the purpose of management. An MO may include attributes that provide information used to characterize itself and operations that represent its behaviours. The purpose of the framework is to provide a collection of capabilities to manage these MOs. MOs need some approaches to describe their properties and behaviours. A service-oriented MO is a managed entity that represents a manageable service resource in terms of shared state and behaviour where state and behaviour are separated through outsourcing of the behaviour to an assigned so-called "managing entity" (e.g., a service and its interface) that takes a steward role with regard to the behaviours of its allocated managed entities. Since an MO's state and behaviour can be separated, state can be described by XML schema and behaviour by web services' interfaces in WSDL. One important benefit of using an XML document to store an MO's state is that web services use XML schema to describe the data type of its exchanged messages, and these XML-based MOs' information can be exchanged without any modification.

8.2.1 Definition of a generic managed object class

A managed object class is a further abstraction of managed objects. All network resources have some common attributes and all MOCs shall inherit, either directly or indirectly, from a super class, namely a ManagedObject. Using ManagedObject to define new MOCs will be easier and faster and provide better maintenance. As mentioned above, all MOCs are described in XML schema and the data type of ManagedObject is given in Table 1 and the attributes can be found in Table 2.

Table 1 – Data type of super class ManagedObject

```
<xsd:complexType name="ManagedObject_C">
  <xsd:sequence>
    <xsd:element name="objectClass" type="xsd:string"/>
    <xsd:element name="objectInstance" type="x782:NameType"/>
    <xsd:element name="packages" type="x782:PackageListType"/>
    <xsd:element name="creationSource" type="x782:SourceIndicatorType"/>
  </xsd:sequence>
</xsd:complexType>
```

Table 2 – Attributes of super class ManagedObject_C

Attribute name	Support qualifier	Read qualifier	Write qualifier
objectClass	Mandatory	Mandatory	–
objectInstance	Mandatory	Mandatory	–
packages	Optional	Mandatory	–
creationSource	Optional	Mandatory	–

As shown in Table 2, ManagedObject is made up of four attributes including objectClass, objectInstance, packages and creationSource. An attribute has an associated value, which may have a simple or a complex structure. Here, the attribute of MO is different from the attribute in the XML

schema specification, and it can be just mapped to the element in the XML schema. The attribute objectClass is used to identify the class type of this MO instance. The attribute objectInstance is used to uniquely identify an MO instance, and the data type is using NameType (also it has the same semantics of DN, distinguished name) as specified in (1). The attribute packages is a set of strings to indicate the capacities that the MO supports. The attribute creationSource indicates whether an MO is created automatically in a managed system, or by a managing system through a management operation, or unknown.

```
DN(list of xsd:string) ::= "<attriubte_name_1>=<attibute_value_1>",          (1)
             "<attribute_name_2>=<attribute_value_2>"
              …,
             " <attribute_name_n>=<attribute_value_n>"
```

The attributes in the above formula should be naming attributes of MOCs.

A complete XML schema definition for the generic ManagedObject_C is defined in clause A.1.

(R) OBJECT-1. All the classes used to model resources on a managed system shall inherit (directly or indirectly) from the *ManagedObject_C* described above and defined in the XML schema in clause A.1. The capabilities described above shall be supported.

8.2.2 Inheritance of managed objects

One MOC shall be defined as a specialization of another MOC by utilizing inheritance such as for all the rest of the MOs that must directly or indirectly inherit from ManagedObject. Specialization of an MOC implies that all attributes defined on the super class will be supported by the subclass too. As attributes of MOs are described in XML schema, the inheritance can be achieved by the extension of data types. For example, assume the equipment MOC inherits directly from the base ManagedObject class, and equipment can inherit all attributes from ManagedObject by extension and declare other attributes as shown in Table 3, such as userLabel, for itself.

Table 3 – Data type of Equipment_C by extension

```
<xsd:complexType name="Equipment_C">
 <xsd:complexContent>
   <xsd:extension base="x782:ManagedObject_C">
     <xsd:sequence>
       <xsd:element name="userLabel" type="xsd:string"/>
       …
     </xsd:sequence>
   </ xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Some MOs may need multiple inheritance, but the XML schema only supports single inheritance for data types and the use of multiple inheritance is not recommended. If the semantics of multiple inheritance are required, the derived MOC has to singly inherit from one of the superior MOCs and attributes from the other superior class(es) should be added manually.

8.2.3 Package feature

Packages can be used to group certain capabilities (for example, related attributes), or provide conditional support capabilities. A package can be defined as an XSD complexType, with a "_P" as its name suffix. When it is used, the element may have a type of the complexType representing this

package, with minOccurs="0" maxOccurs="1" as the qualifiers, which indicate this package can be conditional or optional. An example of package definition and usage is shown in Table 4.

Table 4 – Data type of equipment by extension

```
<!-- definition of a Package -->
<xsd:complexType name="StatePackage_P">
    <xsd:sequence>
       <xsd:element name="adminstrativeState"
type="x782:AdminstrativeStateType"/>
       <xsd:element name="operationalState" type="x782:OperationalStateype"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Equipment_C">
 <xsd:complexContent>
   <xsd:extension base="x782:ManagedObject_C">
    <xsd:sequence>
      <xsd:element name="equipmentId" type="xsd:string"/>
      <xsd:element name="userLabel" type="xsd:string"/>
      …
      <!-- usage of the Package -->
    <xsd:element name="statePackage" type="x782:StatePackage_P" minOccurs="0"
maxOccurs="1" /></xsd:sequence>
   </ xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

8.2.4 Common attributes and data types

The following table shows some common attributes as well as some common data types that can be shared by this framework.

Table 5 – Standard attributes and data types

Attribute name	Data type	Description
administrativeState	AdministrativeStateType	See [ITU-T M.3701] for more details
availabilityStatus	AvailabilityStatusSetType	See [ITU-T M.3701] for more details
backedUpStatus	BackedUpStatusType	See [ITU-T M.3701] for more details
controlStatus	ControlStatusSetType	See [ITU-T M.3701] for more details
creationSource (Note)	SourceIndicatorType	See [ITU-T M.3701] for more details
externalTime	ExternalTimeType	
objectClass (Note)	ObjectClassType	It indicates an MOC
objectInstance (Note)	NameType	It indicates an MO instance
operationalState	OperationalStateType	See [ITU-T M.3701] for more details
packages (Note)	StringSetType	It indicates the packages that are supported

Table 5 – Standard attributes and data types

Attribute name	Data type	Description
		by an MO instance.
proceduralStatus	ProceduralStatusSetType	See [ITU-T M.3701] for more details
standbyStatus	StandbyStatusType	See [ITU-T M.3701] for more details
systemLabel	SystemLabelType	It indicates a label for a system.
unknownStatus	UnknownStatusType	See [ITU-T M.3701] for more details
usageState	UsageStateType	See [ITU-T M.3701] for more details
NOTE – These attributes are inherited by all managed objects.		

The detailed XSD definitions for the above data types can be found in clause A.1.

9 Accessing methods for managed objects

This clause describes a web services-based network management framework and this framework shall provide a collection of methods to control network resources. These methods provide basic capabilities to manage MOs and so they are called generic accessing methods; see Table 6. Figure 1 gives the accessing procedure and the framework uses web services technology to exchange information of MOs. Web services separate an MOs' states and behaviours and exposes their behaviours through a web services interface. As a web service is a service-oriented technology, in this framework all MOs are designed to be accessed through a single interface, and the interface must know which MO is the actual target of an operation, and the unique identifier of the target MO should be provided in each accessing request. According to the above requirements, some necessary generic accessing methods are given in Table 1:

Table 6 – Generic accessing methods

Operation name	Input parameter	Output parameter
getMOAtrributes	– objectInstance : Name – attributeNameList : SEQUENCE OF String	– attributeNameAndValueList : SEQUENCE OF {attributeName, attributeType, attributeValue} – status : ENUMERATION
setMOAttributes	– objectInstance : DN – attributeNVMList : SEQUENCE OF {attributeName, attributeType, attributeValue, modifyOption}	– status
createMO	– objectclass – objectClassInstance – attributeNameAndValueList	– status
deleteMO	– objectInstance	– status
getPackages	– objectInstance	– packages : List of string – status

Where,

1) getMOAttributes – to retrieve all, or any subset, of an MO's attribute value in one operation. It uses the DN as the first parameter to uniquely identify the MO and a list of attribute names to be queried. The return result is made up of attribute values and operation

status. The attributeNameAndValueList is a list of triples including attributeName, attributeType and attributeValue. The attributeType indicates the original type of attributeValue and attribute values are returned through the "any" element of XML schema for arbitrary type values. The status parameter indicates whether the operation is performed successfully or failed. As "any" is defined for the data type of the return attribute value, when receiving such a request from the client, the server will return the requested attributes into the output parameter attributeNameAndValueList, where the attributeValue field will be encoded from a variable element to a piece of XML text, which can be decoded by the client application with the help of the attributeType parameter.

2) setMOAttributes – to modify attribute values of an MO in existence. Besides using objectInstance to indicate the target MO whose values are to be modified, the operation also uses a list of quadruples including attributeName, attributeType, attributeValue, and modifyOption to set MO attributes. The first three attributes are the same as above. The modifyOption indicates how to set corresponding MO attribute values. It is an enumeration type consisting of "REPLACE", "ADDValues", "REMOVEValues" and "SETToDefault". "REPLACE" means the attribute value(s) specified shall be used to replace the current values(s) of the attribute. "ADDValues" means the attribute values(s) specified shall be added to the current value(s) of the attribute. "REMOVEValues" means the attribute value(s) specified shall be removed from the current values(s). "SETToDefault" means the attribute shall be set to its default value. The modifyOption is optional, and if it is not specified, the "REPLACE" shall be assumed.

3) createMO – to create an MO in the managed system. It must specify the created MO's class and name. The attributeNameAndValueList parameter is used to provide attribute values, but it can be omitted, and if it is omitted the attributes are set to default values.

4) deleteMO – to release any resources associated with the MO and to delete it. It uses DN to identify the target MO and then return the operation status. If the target MO cannot be removed or any of its contained MOs cannot be removed, the operation will return an OperationFailed status.

5) getPackages – to return the capabilities of the target MO (a group of attributes and/or operations). An MO instance may or may not support all the capability groups defined in an MOC and this operation is used by the client to get the actual supported capacities of the MO instance.

List and tabular structures are used extensively to provide a compact and efficient representation of potentially complex data that is to be conveyed as a single unit. At the same time, it also introduces problems such as difficulties with data validation and comprehension. As web services are web-oriented and service inheritance will brake up their loose coupling, it is not recommended. A singleton design pattern can be considered in order to keep data consistency of MOs.

All the methods mentioned above just operate on one MO. With potentially millions of entities to manage, there is a need for the framework to support operations on multiple objects with a single method invocation or perhaps a small number of invocations. The multiple object operation (MOO) service provides this capability, which can be found in [ITU-T Q.818].

A complete XML schema and WSDL interface definition for the generic MO accessing methods can be found in clause A.2.

(R) OBJECT-2. An implementation of the MO accessing methods shall support all the operations described above and whose WSDL is defined in clause A.2.

10 Inheritance of managed objects and interface operations

10.1 Attributes inheritance of managed objects

One managed object class may be defined as a specialization of another managed object class by utilizing inheritance. Specialization of a managed object class implies that all methods and attributes defined on the super class will also be supported by the subclass. In case of service-oriented interfaces based on web services, only attributes of managed object classes are supported. Introducing operations inheritance will lead to the weakening of characteristics such as good interoperability and loose coupling.

While attributes of managed objects are described with XML schema, clause 4.2 'Deriving Types by Extension' of [W3C Primer] can be followed to create attributes inheritance. Since attributes data types of base managed objects are defined in complex type in XML schema, the subclass can extend the base managed objects data type by the value of the *base* attribute on the *extension* element in XML schema.

When a complex type is derived by extension, its effective content model is the content model of the base type plus the content model specified in the type derivation. Furthermore, the two content models are treated as two children of a sequential group. In the case of UKAddress, the content model of UKAddress is the content model of Address plus the declarations for a postcode element and an exportCode attribute.

```
<complexType name="Address">
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <element name="city"   type="string"/>
  </sequence>
 </complexType>
 <complexType name="USAddress">
  <complexContent>
   <extension base="ipo:Address">
    <sequence>
     <element name="state" type="ipo:USState"/>
     <element name="zip"   type="positiveInteger"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

USAddress equals to the following:

```
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <element name="city"   type="string"/>
   <element name="state" type="ipo:USState"/>
   <element name="zip"   type="positiveInteger"/>
```

```
  </sequence>
</complexType>
```

The above approach can be used for element inheritance, which supports the semantics for managed object attribute inheritance.

10.2 Considerations for the inheritance of interface operation

Although the later version of WSDL (version 2.0) supports the syntax of inheritance for operations, it is not widely supported by the industry. Earlier versions of WSDL (before version 1.1) do not support this syntax, but it is fully supported by the industry. Thus the expression of operation inheritance is not allowed in this Recommendation. This feature can be supported semantically by repeating the interface operation definitions inherited from super-interfaces.

11 Information modelling guidelines for web services-based interfaces

11.1 namespace

This framework uses the following URI for the target namespace:

Table 5 – Target namespace in this framework

Recommendation	Target namespace
ITU-T X.782	http://www.itu.int/xml-namespace/itu-t/x.782
ITU-T Q.818	http://www.itu.int/xml-namespace/itu-t/q.818
Other, such as X.nnnn	http://www.itu.int/xml-namespace/itu-t/x.nnnn

When developing other Recommendations, the actual Recommendation number should be used to replace "X.nnnn" in the last row of the above table.

11.2 complexType

When using XML schema to define the content of an MOC, the XSD complexType is used for modelling the MOC. An XSD complexType contains a sequence which can include one or more XSD element(s). Each complexType corresponding to an MOC should have a "_C" as the suffix to the name.

Other common attribute data types can also be defined as complexTypes but they should use the suffix "Type" in the type name.

11.3 attribute

Attributes and states of an MOC are defined as the elements in the complexType corresponding to an MOC. Note here the attribute is the conceptual property of an entity which is modelled as an MOC, it is not the same as the keyword attribute in XSD. In this framework, the XSD keywords attribute of element will not be used.

11.4 request

Request is used to define a message for interactions between a web service client and a web service server application. A request message is sent from the web service client to the server. A request contains all the input parameters of an operation in a web service. The input parameters can be of multiple parts, or a request can only define one part which combines all the input parameters as its internal elements, contained in its corresponding complexType.

11.5 response

Response is used to define a message sent from the web service server back to the client. A response contains all the output parameters as well as the return value of an operation in a web service. The output parameters can be multiple parts, or a response can define one part, which combines all the output parameters as its internal elements, contained in its corresponding complexType.

11.6 notification

The format of any notifications to be sent across the management interface should follow the format in the [OASIS WSN] specification. [ITU-T Q.818] provides a common header for all the notifications, and the notification contents for some commonly used notification types, including:

objectCreation, objectDeletion, attributeValueChange, stateChange, communicationAlarm, environmentalAlarm, equipmentAlarm, processingErrorAlarm, qualityOfServiceAlarm, Violation, integrityViolation, operationalViolation, physicalViolation, securityViolation, timeDomainViolation, relationshipChange, and heartbeat notifications.

Implementers of this framework should follow the notification definitions in [ITU-T Q.818] for the known notification types. Any new notification definitions should follow the same approach as well as the notification header.

11.7 Name conventions for MOCs, packages, attributes and data types

The following name conventions are applied for XML schema based modelling:

– All the attributes of an MOC are defined as an XSD complexType, the name of the MOC should have a "_C" as name suffix, with the first character capitalized, so that this complexType can be distinguished from other normal data type definitions. For example: ManagedObject_C, Equipment_C.

– An attribute of an MOC is defined as an element within the complexType presenting an MOC, and the first character of an attribute name should be in lower case.

– A package can be defined as an XSD complexType, with a "_P" as its name suffix, with the first character capitalized.

– A normal data type definition should have a "Type" as its name suffix, with the first character capitalized to make it more readable. For example: AdminstrativeStateType.

– Use lowerCamelCase, e.g., "personName" for elements.

– Use UpperCamelCase for defining simpleType and complexType names.

– A set-valued type (unordered set) should have SetType as its name suffix, and a list-valued type (ordered sequence) should have ListType as its name suffix.

12 Style idioms for XML schema specifications[1]

12.1 Data model using XML schema

12.1.1 Overview of XML schema use

XML allows arbitrary data definitions using ad-hoc tags. The XML document becomes practical for use when it is constrained by a well-defined structure. XML schema provides a means of defining rules, semantics and structure for XML documents. A schema provides programmatic validation of a structured XML document. An XML schema is defined using an XML format in a file with an .xsd extension.

12.1.2 Schema specification version

The namespace for XML schema 1.1 http://www.w3.org/2001/XMLSchema is the same as for XML schema for the 1.0 document and the namespace for the XML instance document remains http://www.w3.org/2001/XMLSchema-instance. This allows schema developers to develop XML schema 1.0 documents without worrying about updating namespaces when adding 1.1 features. XML schema 1.1 introduces a new namespace for version control (http://www.w3.org/2007/XMLSchemaversioning).

This Recommendation uses XML schema version 1.1 as specified in clause 7.2 of [ITU-T Q.818].

12.2 XML schema design considerations

12.2.1 Developing a single schema or a collection of related schemas

The schema definition language allows constructs to import schema from other documents. When developing a set of related schemas, namespace considerations become important in how the resulting XML instance document references other schemas.

– Reference using <xsd:import>: The import element allows references to schema components from schema documents with different target namespaces. This is the most common schema design approach and is also referred to as heterogeneous namespace design.

– Reference using <xsd:include>: The include element adds the schema components from other schema documents that have the same target namespace (or no specified target namespace) to the containing schema. The include element thus allows you to add all the components of an included schema to the containing schema. This gives rise to two different design approaches.

– Homogeneous namespace design approach: In this design approach all related schemas being developed are assigned the same target namespace.

– Chameleon namespace design approach: In chameleon namespace design approach, there is one or more supporting schema defined with no target namespace, the main schema includes the supporting schema(s). The supporting schema(s) take the namespace of the main schema.

The homogeneous or the chameleon design approach allows for the ability to redefine a type, group and attribute group definitions defined in a supporting schema. Type redefinition affects the elements of the including schema as well as for those in the included schema. Thus redefined types can interact with derived types and generate conflicts.

In this Recommendation and [ITU-T Q.818], only the [xsd:import] approach will be used.

---

[1] The text in this clause is originally described in [ATIS-I-000002], with some updates to fit this web services-based management framework.

12.2.2 Schema design patterns

The following four structural design patterns are commonly mentioned in relation to XML schema design:

– Russian doll: The Russian doll design provides a single global element in the schema file. All child elements are defined within this single element definition hierarchy. The design does not promote element reusability and requires the definition of elements in a single schema file.

– Salami slice: In this design approach, multiple root elements are defined but no complexType is defined. The design supports reusability of individual elements.

– Garden of Eden: A schema design approach in which all elements and types are exposed globally, complex types are defined through reference to reusable global elements. The characteristic of this design pattern is that there are many possible root elements.

– Venetian blinds: A schema design approach in which types are created first from which elements are built. The simpleTypes and complexTypes are created so as to maximize reuse. The characteristic of this design approach is that there is a single root element.

In this framework, the Garden of Eden design is used.

12.2.3 Designing for resilience

When developing a schema one of the goals is to make it resilient to changes and provide flexibility that anticipates future needs of the schema users. This clause discusses some of the mechanisms that are available for providing flexibility and extensibility in an XML schema model.

12.2.3.1 Using wildcards

The W3C schema allows constructs such as <xsd:any> and <xsd:anyAttribute> to allow an instance document to contain XML data not directly constrained by the content model of the defining XML schema. This allows the extensibility mechanism to have some degree of control that can be specified using namespace and processContents attributes (See [W3C XS-P1] and [W3C XS-P2]). The namespace attribute, for example, can be used to constrain the XML data to a set of predefined namespaces thus providing some data validation that can be performed on these extended XML data within an instance document. The "processContents" attribute controls how this extended XML data is validated by the XML validator.

The <xsd:any> wildcard can allow vendors to develop vendor-specific functionalities not defined in the defining schema. A careful use of <xsd:any> and <xsd:anyAttribute> can help build a schema content model more resilient to change and less prone to schema churn. However on a note of caution, with <xsd:any> it is possible to inadvertently allow the creation of non-deterministic content models which can cause issues with some XML parsers. A schema developer will have to consider this constraint when deciding to use <xsd:any>.

It is allowed to use <xsd:any> in this framework but explanations for the usage should also be provided whenever <xsd:any> is used.

12.2.3.2 Using substitutionGroup

The content model of substitution group members is related to each other by type derivation. The replaceable element is called the head element and has to be defined in the schema's global scope. In essence, the substitutionGroup construct helps build a collection of elements that can be specified using a generic element.

The substitutionGroup construct can be helpful in the following cases:

– Development of related class hierarchies: Creating class hierarchies can enable object oriented programming languages to take advantages of such inheritance.

– Customizing an external schema for a specific need: If an element in an imported schema is defined globally, it is possible to create a substitutionGroup for this element through construction of a derived type and allowing substitution of this derived type in a complex data structure.

Class hierarchies and substitutionGroup results in tight coupling between data structures and can lead to brittle and unmodifiable design. Constructs such as <xsd:choice> can be used instead to create a composite content model. The composite design can lead to simplicity and a decoupled design.

This Recommendation uses extension to complexType to present class hierarchy and substitutionGroup is not used in this framework.

12.3 Recommendations for schema developers

12.3.1 XML schema design recommendations

The author of an XML based interface should:

– create shared schemas whenever feasible.

The author shall:

– use the following namespace format for ITU-T generated schemas:

  http://www.itu.int/xml-namespace/itu-t/<ITU-T document number> or

  http://www.itu.int/xml-namespace/itu-t/<ITU-T document number>/<data model identifier>

– use only lowercase characters for namespace names.

– It is possible that the document could change with little or no change to the schema (e.g., updated references).

For this reason, the <ITU-T document number> does not include the document version number. Including the document version number would force a change in namespace with every document update.

The author should:

– Declare all simple and complex types globally.

– Declare elements and attributes locally. One main element encapsulates all others.

– Ensure the schema version attribute (schema minor version number) is present and increments with any change to the schema. The initial value is expected to be "0".

– Ensure all schemas have at least 2 namespaces: the W3C XML schema namespace and the namespace related to the companion standard.

– Provide example schema attributes for version 1.0 of the MO access service (moas) schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.782"
   xmlns:moas="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"
   version="1.0">
.....
</schema>
```

12.3.2 XML schema coding recommendations

The author should utilize the following guidelines for types, elements, and attribute TAG names:

– Utilize common industry or business names where possible to ensure consistency between industry or business documentation and schemas.

– Make names descriptive; avoid using unnecessary abbreviations.

– Do not use the same name twice in any one schema.

– Avoid acronyms but if they are used the capitalization remains.

– Avoid period and dash characters.

– Use only alpha-numeric characters to define elements and type names.

– Use singular names unless the concept itself is plural.

The author should consider element as the default model type and use the following guidelines:

– If the item can be considered an independent object, make it an element.

– If the item needs to be re-usable, make it a global type.

The author should utilize the following general guidelines:

– Declare elements as optional unless absolutely required.

– Use minOccurs="0" instead of nillable="True".

– Use maxOccurs. If a dimension is more than one, define it. Otherwise use maxOccurs="unbounded".

– Create simpleTypes as much as possible.

– Create a global type when the element is to be reusable. (Global types are defined directly under the schema element.)

– All types are to be defined globally.

– Use elementFormDefault = "qualified".

– Use attributeFormDefault = "unqualified".

– Use UTF-8 character encoding: <?xml version="1.0" encoding="UTF-8"?>

– The <documentation> elements are to be used wherever re-use is likely and more clarity is desired. (The xsd:lang attribute is to be used to specify language.)

– Use annotations to describe all types definitions; minimally include the type name.

– Use only xsd:dateTime element for date and time items.

– Use only <sequence> or <choice> where a compositor is required.

The author should utilize the following guidelines when adding constraints:

– When designing new schema, the new simple types are to be constrained and appropriate restrictions applied whenever possible.

– Identify facets that constrain the range of values.

– Choose the appropriate simple type. For example, when a number is required, use of the nonNegativeInteger or positiveInteger simple type is preferred to integer if possible.

– When defining a string type, if possible use maxLength and patterns to provide additional restrictions. For example, when defining a 15-digit international phone number, the pattern can be ([ITU-T E.164]):

```
<xsd:restriction base="xsd:string">
   <xsd:pattern           value="([1-9][0-9]{0,2})(-[1-9][0-9]{0,4})?(-[1-
9][0-9]{0,13})(-[0-9]+)?"/>
</xsd:restriction>
```

– When a simple type takes only a predefined set of values, use an enumeration facet to restrict the legal values of the simple type.
– Use union types if the data types take two or more different sets of values that can independently be constrained. For example, state code or zip code.

12.3.3 XML schema constructs to avoid

The author should use the following guidelines for XML schema constructs:

– Do not use <xsd:all> (use <sequence> or <choice>). The <xsd:sequence> and <xsd:choice> forces a fixed ordering of child elements in the instance document.
– Do not use processing instructions. Processing instructions do not form part of the document but are passed to the application to perform application specific functions. Processing instructions do not have to follow the internal structure and as such have little use in schema definitions.
– Do not use DTD or XML style comments. The annotation and documentation elements provide construct for comments and information. The XML style comments are not useful for the processing of an instance document.
– Do not use XML groups or group redefinition. The redefinition of a group can generate conflict between processing of redefined type and derived type instance data.
– Do not use substitution groups. The substitution group construct creates tight coupling and adds complexity, which can lead to brittle and unmodifiable designs.
– Do not use default/fixed values. Including such attribute uses will tend to mislead readers of the schema document because the attribute uses would have no effect.

12.4 Guidelines for schema extensions

It is understood that the schema authors cannot anticipate in advance all future needs of a schema. The author should utilize a schema definition that provides flexibility and allows implementers to carry private or custom data. In general, schema authors should anticipate the need for supporting private data by including constructs to support the inclusion of some data in a generic way (for example name-value pair).

However such inclusion might not be suffice for a more complex need, where for example, validation, rules assertions and policies might be required.

The schema author should utilize one of two possible ways of extending support for private data in a schema:

1) anticipate the needs of extensions to specific portions of schema and allow users to add private openContent data; or
2) type inheritance using <xsd:extension>.

Use of such extension is discouraged as this will potentially lead to interpretability issues. For example, if an implementer extends a standard schema using xsd:extension to add implementer specific elements and attributes, a resultant XML data might not parse correctly by other implementers who base their applications on the original standard schema.

Such cases can be avoided if both sides use the same XML schema. Extension can be used in this framework, but the extended types should also be included as a public standard.

13 Compliance and conformance

This clause defines the criteria that must be met by other standards documents claiming compliance to these guidelines and the functions that must be implemented by systems claiming conformance to this Recommendation.

13.1 Standards document compliance

Any specification claiming compliance with these guidelines shall:

1) Define all classes that model resources as a derivation (direct or indirect) from the *ManagedObject_C* described in clause 8.2.1 and defined in the XML schema in clause A.1.
2) Support the attributes inheritance using the mechanism specified in clause 10.1.
3) Use the definitions for generic attribute types found in clause 8.2.4 wherever applicable.
4) Use the common data types defined in the XML schema in clause A.1 whenever appropriate.
5) Adhere to the modelling guidelines for web services-based interfaces specified in clause 11.
6) Adhere to the XML schema design conventions specified in clause 12.

13.2 System conformance

An implementation claiming conformance to this Recommendation shall:

1) support all of the capabilities of the MO accessing methods described in clause 9, and the corresponding WSDL interface as defined in clause A.2.

13.3 Conformance statement guidelines

The conformance statement must identify a document and year of publication to make sure the right version of XML schema and WSDL is identified.

Annex A

Common WSDL and XML schema definitions

(This annex forms an integral part of this Recommendation.)

In this annex, the common definitions of WSDL interfaces as well as some common XML schema based data types are defined.

A.1 XML schema definitions for common data types and a generic managed object

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- XML Schema Definition for common data types to be used in this framework.
       Filename : x782.xsd -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:x782="http://www.itu.int/xml-namespace/itu-t/x.782"
   targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.782"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="1.0">


<xsd:simpleType name="RDNType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>


<xsd:complexType name="NameType">
   <xsd:sequence>
      <xsd:element name="rdn" type="x782:RDNType" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="NameSetType">
   <xsd:sequence>
      <xsd:element name="dn" type="x782:NameType" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="UIDType">
   <xsd:sequence>
      <!-- uri indicates the namespace where the constant is defined. -->
      <xsd:element name="uri" type="xsd:string"/>
      <!-- value indicates the constant value for this item in the above
namespace. -->
      <xsd:element name="value" type="xsd:unsignedLong"/>
   </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="UIDSetType">
   <xsd:sequence>
      <xsd:element name="uid" type="x782:UIDType" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MOClassListType">
   <xsd:sequence>
      <xsd:element name="moClass" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="AdministrativeStateType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="locked"/>
      <xsd:enumeration value="unlocked"/>
      <xsd:enumeration value="suttingDown"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="OperationalStateType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="disabled"/>
      <xsd:enumeration value="enabled"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="AvailabilityStatusType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="inTest"/>
      <xsd:enumeration value="failed"/>
      <xsd:enumeration value="powerOff"/>
      <xsd:enumeration value="offLine"/>
      <xsd:enumeration value="offDuty"/>
      <xsd:enumeration value="dependency"/>
      <xsd:enumeration value="degraded"/>
      <xsd:enumeration value="notInstalled"/>
      <xsd:enumeration value="logFull"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="AvailabilityStatusSetType">
   <xsd:sequence>
```

```
      <xsd:element name="availableState" type="x782:AvailabilityStatusType"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="BackedUpStatusType">
   <xsd:restriction base="xsd:boolean"/>
</xsd:simpleType>
<xsd:simpleType name="ControlStatusType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="subjectToTest"/>
      <xsd:enumeration value="partOfServicesLocked"/>
      <xsd:enumeration value="reservedForTest"/>
      <xsd:enumeration value="suspended"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ControlStatusSetType">
   <xsd:sequence>
      <xsd:element name="controlState" type="x782:ControlStatusType"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="ExternalTimeType">
   <xsd:restriction base="xsd:dateTime"/>
</xsd:simpleType>

<xsd:simpleType name="ObjectClassType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="ProceduralStatusType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="initializationRequired"/>
      <xsd:enumeration value="notInitialized"/>
      <xsd:enumeration value="initializing"/>
      <xsd:enumeration value="reporting"/>
      <xsd:enumeration value="terminating"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ProceduralStatusSetType">
   <xsd:sequence>
```

```
      <xsd:element name="proceduralState" type="x782:ProceduralStatusType"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="SourceIndicatorType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="resourceOperation"/>
      <xsd:enumeration value="managementOperation"/>
      <xsd:enumeration value="unknown"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="StandbyStatusType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="hotStandby"/>
      <xsd:enumeration value="coldStandby"/>
      <xsd:enumeration value="providingService"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="StringSetType">
   <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="SystemLabelType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="UsageStateType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="idle"/>
      <xsd:enumeration value="active"/>
      <xsd:enumeration value="busy"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="UnknownStatusType">
   <xsd:restriction base="xsd:boolean"/>
</xsd:simpleType>

<xsd:complexType name="AttributeValueType">
```

```
   <xsd:sequence>
      <xsd:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeNameAndValueType">
   <xsd:sequence>
      <xsd:element name="attributeName" type="xsd:string"/>
      <xsd:element name="attributeType" type="xsd:string"/>
      <xsd:element name="attributeValue" type="x782:AttributeValueType"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AttributeNameAndValueSetType">
   <xsd:sequence>
      <xsd:element name="attributeNameAndValue"
type="x782:AttributeNameAndValueType" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="AdditionalTextType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="AnyValueType">
   <xsd:sequence>
      <xsd:element name="typeURI" type="xsd:string"/>
      <xsd:element name="value" type="x782:AttributeValueType"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AdditionalInformationSetType">
   <xsd:sequence>
      <xsd:element name="additionalInfo" type="x782:AnyValueType" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="NotificationIDType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="NotificationIDSetType">
   <xsd:sequence>
      <xsd:element name="source" type="x782:NameType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CorrelatedNotificationType">
   <xsd:sequence>
      <xsd:element name="source" type="x782:NameType"/>
      <xsd:element name="notifIDs" type="x782:NotificationIDSetType"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CorrelatedNotificationSetType">
   <xsd:sequence>
      <xsd:element name="notifications" type="x782:CorrelatedNotificationType"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType><xsd:complexType name="AttributeChangeType">
   <xsd:sequence>
      <xsd:element name="attribugteName" type="xsd:string"/>
      <xsd:element name="attributeTypeURI" type="xsd:string"/>
      <xsd:element name="oldValue" type="x782:AttributeValueType"/>
      <xsd:element name="newValue" type="x782:AttributeValueType"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeChangeSetType">
   <xsd:sequence>
      <xsd:element name="attributeChange" type="x782:AttributeChangeType"
minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ProbableCauseType">
   <xsd:complexContent>
          <xsd:extension base="x782:UIDType"/>
       </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="PerceivedSeverityType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="indeterminate"/>
      <xsd:enumeration value="critical"/>
      <xsd:enumeration value="major"/>
      <xsd:enumeration value="minor"/>
      <xsd:enumeration value="warning"/>
      <xsd:enumeration value="cleared"/>
   </xsd:restriction>
```

```
</xsd:simpleType>

<xsd:simpleType name="TrendIndicationType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="lessSevere"/>
      <xsd:enumeration value="noChange"/>
      <xsd:enumeration value="moreSevere"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ThresholdIndicationType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="up"/>
      <xsd:enumeration value="down"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="ThresholdLevelIndType">
   <xsd:sequence>
      <xsd:element name="indication" type="x782:ThresholdIndicationType"/>
      <!-- observed value -->
      <xsd:element name="low" type="xsd:float" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="high" type="xsd:float"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ThresholdInfoType">
   <xsd:sequence>
      <xsd:element name="attributeID" type="xsd:string"/>
      <xsd:element name="observedValue" type="xsd:float"/>
      <xsd:element name="thresholdLevel" type="x782:ThresholdLevelIndType"/>
      <xsd:element name="armTime" type="xsd:dateTime"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ProposedRepairActionSetType">
   <xsd:complexContent>
      <xsd:extension base="x782:UIDType"/>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SuspectObjectType">
   <xsd:sequence>
      <xsd:element name="moClass" type="xsd:string"/>
      <xsd:element name="suspectedMOInstance" type="x782:NameType"/>
```

```
      <xsd:element name="failureProbability" type="xsd:unsignedShort"
minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SuspectObjectSetType">
   <xsd:sequence>
      <xsd:element name="suspectedMO" type="x782:SuspectObjectSetType"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SecurityAlarmCauseType">
   <xsd:complexContent>
      <xsd:extension base="x782:UIDType"/>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SecurityAlarmDetectorType">
   <xsd:sequence>
      <xsd:element name="mechanism" type="x782:UIDType"/>
      <xsd:element name="obj" type="x782:NameType"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ServiceUserType">
   <xsd:sequence>
      <xsd:element name="typeURI" type="xsd:string"/>
      <xsd:element name="value" type="x782:AttributeValueType"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ServiceProviderType">
   <xsd:sequence>
      <xsd:element name="typeURI" type="xsd:string"/>
      <xsd:element name="value" type="x782:AttributeValueType"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SpecificProblemSetType">
   <xsd:complexContent>
      <xsd:extension base="x782:UIDSetType"/>
   </xsd:complexContent>
</xsd:complexType>

<!-- XML Schema Definition for generic Managed Object -->
```

```
<xsd:complexType name="ManagedObject_C">
   <xsd:sequence>
      <xsd:element name="objectClass" type="xsd:string"/>
      <xsd:element name="objectInstance" type="x782:NameType"/>
      <xsd:element name="packages" type="x782:StringSetType"/>
      <xsd:element name="creationSource" type="x782:SourceIndicatorType"/>
   </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

A.2 WSDL and XML schema definition for common object accessing methods

(1) ITU MO access service XML schema definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- XML Schema Definition for data types to be used in MO access Service
specified in this Recommendation.
       Filename : x782_MOAccessService.xsd -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:x782="http://www.itu.int/xml-namespace/itu-t/x.782"
   xmlns:moas="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"
   targetNamespace="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="1.0">
<xsd:import namespace="http://www.itu.int/xml-namespace/itu-t/x.782"
schemaLocation="x782.xsd"/>
   <xsd:complexType name="AttributeNameListType">
      <xsd:sequence>
         <xsd:element name="attributeName" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="GetMOAttributesRequestType">
      <xsd:sequence>
         <xsd:element name="objectInstance" type="x782:NameType"/>
         <xsd:element name="attributeNameList"
type="moas:AttributeNameListType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="StatusType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="OperationSucceed"/>
         <xsd:enumeration value="OperationFailed"/>
      </xsd:restriction>
   </xsd:simpleType>
```

```
   <xsd:complexType name="GetMOAttributesResponseType">
      <xsd:sequence>
         <xsd:element name="attributeNameAndValueList"
type="x782:AttributeNameAndValueSetType"/>
         <xsd:element name="status" type=" moas:StatusType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="ModifyOptionType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="REPLACE"/>
         <xsd:enumeration value="ADDValues"/>
         <xsd:enumeration value="REMOVEValues"/>
         <xsd:enumeration value="SETToDefault"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="AttributeNVMType">
      <xsd:sequence>
         <xsd:element name="attributeName" type="xsd:string"/>
         <xsd:element name="attributeType" type="xsd:string"/>
         <xsd:element name="attributeValue" type="x782:AttributeValueType"/>
         <xsd:element name="modifyOption" type="moas:ModifyOptionType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="AttributeNVMListType">
      <xsd:sequence>
         <xsd:element name="attributeNVM" type=" moas:AttributeNVMType"
minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="SetMOAttributesRequestType">
      <xsd:sequence>
         <xsd:element name="objectInstance" type="x782:NameType"/>
         <xsd:element name="attributeNVMList" type="
moas:AttributeNVMListType"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="CreateMORequestType">
      <xsd:sequence>
         <xsd:element name="objectClass" type="xsd:string"/>
         <xsd:element name="objectInstance" type="x782:NameType"/>
         <xsd:element name="attributeNameAndValueList"
type="x782:AttributeNameAndValueSetType"/>
```

```
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="GetPackagesResponseType">
      <xsd:sequence>
         <xsd:element name="status" type="moas:StatusType"/>
         <xsd:element name="packages" type="x782:StringSetType"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

(2) ITU MO access service WSDL definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- WSDL Operation Definition for MO Access Service specified in this
Recommendation.
       Filename : x782_MOAccessService.wsdl -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:x782="http://www.itu.int/xml-namespace/itu-t/x.782"
xmlns:moas="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"
name="MOAccessService"
targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService">
<import namespace="http://www.itu.int/xml-namespace/itu-t/x.782"
location="x782.xsd"/>
<import namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"
location="x782_MOAccessService.xsd"/>

   <wsdl:message name="getMOAttributesRequest">
      <wsdl:part name="getMOAttributesInput"
type="moas:GetMOAttributesRequestType"/>
   </wsdl:message>
   <wsdl:message name="getMOAttributesResponse">
      <wsdl:part name="getMOAttributesOutput"
type="moas:GetMOAttributesResponseType"/>
   </wsdl:message>
   <wsdl:message name="setMOAttributesRequest">
      <wsdl:part name="setMOAttributesInput"
type="moas:SetMOAttributesRequestType"/>
   </wsdl:message>
   <wsdl:message name="setMOAttributesResponse">
      <wsdl:part name="status" type="moas:StatusType"/>
   </wsdl:message>
   <wsdl:message name="createMORequest">
      <wsdl:part name="createMOInput" type="moas:CreateMORequestType"/>
   </wsdl:message>
   <wsdl:message name="createMOResponse">
      <wsdl:part name="status" type="moas:StatusType"/>
   </wsdl:message>
```

```
   <wsdl:message name="deleteMORequest">
      <wsdl:part name="objectInstance" type="x782:NameType"/>
   </wsdl:message>
   <wsdl:message name="deleteMOResponse">
      <wsdl:part name="status" type="moas:StatusType"/>
   </wsdl:message>
   <wsdl:message name="getPackagesRequest">
      <wsdl:part name="objectInstance" type="x782:NameType"/>
   </wsdl:message>
   <wsdl:message name="getPackagesResponse">
      <wsdl:part name="getPackageOutput" type="moas:GetPackagesResponseType"/>
   </wsdl:message>
   <wsdl:portType name="MOAccessServicePortType">
      <wsdl:operation name="getMOAttributes">
         <wsdl:input message="moas:getMOAttributesRequest"/>
         <wsdl:output message="moas:getMOAttributesResponse"/>
      </wsdl:operation>
      <wsdl:operation name="setMOAttributes">
         <wsdl:input message="moas:setMOAttributesRequest"/>
         <wsdl:output message="moas:setMOAttributesResponse"/>
      </wsdl:operation>
      <wsdl:operation name="createMO">
         <wsdl:input message="moas:createMORequest"/>
         <wsdl:output message="moas:createMOResponse"/>
      </wsdl:operation>
      <wsdl:operation name="deleteMO">
         <wsdl:input message="moas:deleteMORequest"/>
         <wsdl:output message="moas:deleteMOResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getPackages">
         <wsdl:input message="moas:getPackagesRequest"/>
         <wsdl:output message="moas:getPackagesResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="MOAccessServiceBinding"
type="moas:MOAccessServicePortType">
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getMOAttributes">
         <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService/getMOAttributes"/>
         <wsdl:input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:input>
```

```
         <wsdl:output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="setMOAttributes">
         <soap:operation soapAction=" http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService/setMOAttributes"/>
         <wsdl:input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="createMO">
         <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService/createMO"/>
         <wsdl:input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="deleteMO">
         <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService/deleteMO"/>
         <wsdl:input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getPackages">
```

```
         <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService/getPackages"/>
         <wsdl:input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOAccessService"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="MOAccessService">
      <wsdl:port name="MOAccessService" binding="moas:MOAccessServiceBinding">
         <soap:address location="http://www.itu.int/xml-namespace/itu-
t/x.782/MOAccessService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Appendix I

Overview of web service technology and application scenarios in network management interfaces

(This appendix does not form an integral part of this Recommendation.)

In this appendix, the overview of web services technologies is given.

I.1 Characteristics of web service technology

Web services provide a simplified mechanism to connect applications regardless of the technology or devices they use, or their location. They are based on industry standard protocols with universal vendor support that can leverage the Internet for low cost communications, as well as other transport mechanisms. The loosely coupled messaging approach supports multiple connectivity and information sharing scenarios via services that are self-describing and can be automatically discovered.

Unlike traditional distributed environments, web services emphasize interoperability. Web services are independent of a particular programming language, whereas traditional environments tend to be bound to one language or another. Similarly, since they can be easily bound to different transport mechanisms, web services offer more flexibility in the choice of these mechanisms. Furthermore, unlike traditional environments, web services are often not bound to particular client or server frameworks. Overall, web services are better suited to a loosely coupled, coarse-grained set of relationships. Relying on XML gives web services an additional advantage, since XML makes it possible to use documents across heterogeneous environments.

Web service is an XML schema based technology which has the following benefits in software applications.

1) Good interoperability

Web service is universally interoperable because it uses platforms and language independent protocols such as SOAP. Web services technology provides a new level of interoperability between software applications. Many platform providers, software developers, and utility providers enable their software with SOAP, WSDL, and UDDI capabilities.

2) Loosely coupled

Web services are self-describing software modules which encapsulate discrete functionality. Web services are accessible via standard Internet communication protocols like XML and SOAP. These web services can be developed in many implementation languages, and any other applications or web services can access these services. Therefore, web services are loosely-coupled applications.

3) Broadly used

Web services are now broadly used in the IT services industry, for example e-business, business-to-business applications. Now several stable platforms are already provided to support the development of web service applications.

4) Software and data reusability

Web services support the component-based model in software development, which allows developers to reuse the building blocks created by others to assemble complex applications and extend them in new ways.

It is not only allowed to reuse the source code but also the data behind the source code of reuse in web services. Another kind of software reuse in web services is to integrate these functions in several relevant applications and expose them via web service interfaces.

5) Easy for service composition

Web services allow the definition of increasingly complex applications by progressively aggregating components at higher levels of abstraction. A client invoking a composite service can itself be exposed as a web service. Combining the functionality of several web services can form a new web service, which is called service composition. Service composition can be either performed by composing elementary or composite services.

Web services provide a standardized way to expose and access the functionality of applications as services, and there are specialized languages (such as BPEL) for business process definition and execution.

6) Low cost

Currently, there are many tools, products, and technologies supporting web service standards. This gives organizations a wide variety of choices which help to lower the costs of developing new applications, running costs, environmental costs and the cost of integration.

I.2 Suitable and unsuitable application scenarios of web services in network management

(1) Suitable application scenarios in general

Generally speaking, based on the characteristics of web services it is suitable for the following application scenarios.

– Communication across firewall

As web services use standard SOAP as the transferring protocol it can easily pass firewalls or proxy servers which may be located between different related applications without difficulty. It is usually more complex when using other communication middleware.

– Application integration

It is known that applications often need to run on programs on one kind of platform and obtain data from or send data to applications on some other platforms. Even on the same platform, a variety of software provided by different manufacturers often needs to be integrated. Over web services, applications can expose the functions and data to other applications to use in standard ways.

– B2B interface integration

Integration of cross-enterprise business transactions are usually called B2B integration. Through web services enterprise can integrate critical business applications and then expose them to the designated suppliers and customers, and the greatest benefit of using web services to implement B2B integration is that it can easily achieve interoperability.

– Open interface supporting good changeability

Web services' interfaces are defined in web services description language (WSDL). The WSDL defines services as collections of network endpoints or ports. The WSDL specification provides an XML format for documents for this purpose. The abstract definition of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions. In this way, WSDL describes the public open interface to the web service. Because of the separation of data definition and interfaces definition, client and server of web service applications can be developed separately and communicate through this open interface, and the change of interfaces will result in there being less impact on the development of web service applications.

(2) Unsuitable application scenarios in general

As a web service is a loosely coupled technology mainly designed for application interoperability and integration in a heterogeneous environment, it also has weak points such as a lower execution speed and less encoding efficiency compared to some other technologies (such as CORBA or DCOM). For certain use cases, using a web service may not be the best choice. For example:

– single machine system;

– isomorphic LAN applications (applications interworking in a single LAN environment with the same platform and underlying middleware);

– online interaction with a large amount of data.

(3) Considerations for web services application scenarios in network management

Based on the above analysis, the following interfaces in a network management domain are considered more suitable for web services application:

– B2B/C2B interface

– F interface

– high level OS-OS interface (service management layer or business management layer), etc.

For the above cases, good use can be made of the benefits of web services such as being loosely-coupled, inter-enterprise application integration, web-based access and good extensibility for new service applications.

It may not be a good choice to use web services on EMS-NE management interfaces, as they are usually provided by the same vendor, and may not be necessary as open interfaces.

For NMS-EMS management interfaces, web services may be used, but it may not be the best choice for some cases. For example, CORBA is more encoding efficient and has a higher execution speed in a LAN environment.

Bibliography

[b-ITU-T X.780] Recommendation ITU-T X.780 (2001), *TMN guidelines for defining CORBA managed objects*.

[b-ITU-T X.780.2] Recommendation ITU-T X.780.2 (2007), *TMN guidelines for defining service-oriented CORBA managed objects and façade objects*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks, open system communications and security**
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2012