INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2009-2012

STUDY GROUP 2

TD 318 Rev.1 (PLEN/2)-E

English only

Original: English

Question(s):	9/2	Geneva, 21-29 March 2012
TD		
Source:	Editors	
Title:	Draft X.782 (ex X.ws-xml): Guidelines for defining Web-services for managed objects and management interfaces (for consent)	

Draft Recommendation ITU-T X.782

Guidelines for defining Web-services for managed objects and management interfaces

Summary

This Recommendation defines a set of services required to support service-oriented Web-services based interfaces, and along with ITU-T X.782 composes a framework for Web-services based network management interface. It specifies protocol requirements, how some well-known Web-services should be used in network management interfaces, and defines some network management specific support services. The WSDL interface definitions for the network management specific support services are also provided.

This Recommendation defines a set of guidelines for managed object modelling and management interface for Web-service based network management. 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 defining some basic data types, generic managed object (MO) and generic MO accessing methods. This Recommendation and ITU-T Recommendation Q.818 together compose a framework for Web-service based network management interfaces with a wide range of applications.

Keywords

Web Service (WS), Web Services Description Language (WSDL), eXtensible Markup Language (XML), XML Schema, Distributed Processing, Managed Objects, Network Management Interfaces.

Contact:	WANG Ying MIIT/BUPT China	Tel: Fax: Email:	+86-10-62283119 ext. 8514 +86-10-62283412 wangy@bupt.edu.cn
Contact:	WANG Zhi-li MIIT/BUPT China	Tel: Fax: Email:	+86-10-62283119 ext. 8726 +86-10-62283412 zlwang@bupt.edu.cn
Contact:	Cheng Qiao-gang ZTE China	Tel: Fax: Email:	+86-755-26770000-3711 +86-755-26773583 chen.qiaogang@zte.com.cn

Contents

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 paradigm can be introduced into network management interface, 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 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-service in network management interfaces;
- Generic accessing methods for managed objects;
- Inheritance of managed objects and interfaces;
- Information modelling guidelines for Web-service based interface;
- 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 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 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.*

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

[ITU-T E.164] Recommendation ITU-T E.164 (2011), *The international public telecommunication numbering plan.*

[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).*

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

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

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

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

3 Definitions

3.1 Terms defined elsewhere:

This Recommendation uses the following terms defined elsewhere:

3.1.1 manager [ITU-T M.3020]

3.1.2 agent [ITU-T M.3020]

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

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 Guideline for the Definition of Managed Object
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
OASIS	Advancing Open Standards for the Information Society
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
W3C	World Wide Web Consortium
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, is included in Annex A. The WSDL and XML are written in a 9-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 Web-Services based management framework

Web Services has been widely used in the IT industry. Appendix I provides more information on the features of Web services technology. Web services can be used in network management interfaces, similarly as the CORBA technology.

This Recommendation jointly 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 service using OASIS Web Service base Notification [OASIS WSN].
- Usage of OASIS UDDI Service Registration [OASIS UDDI].
- Definition of Heartbeat service.
- Definition of Multiple Object Operations (MOO) Service.
- Definition of 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 concern super-classes, naming of managed objects and service-oriented interfaces, operations and notifications.

Web-service is a service-oriented technology compared with the traditional CORBA/IDL and GDMO/CMIP management paradigm. An object-oriented interface analysis and design (OOAD) focuses on the class level, i.e., encapsulates behaviour and related data in the same object, and an object expose one or more interfaces for accessing their encapsulated states and attributes. The service-oriented interface design separates the encaplolated data and state from the behaviour, thus

the loose coupling can be achieved in 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 pattern. The managemnet and controlling functions are defined using a serivce-grained approach, that is, the interace operations are organized with the unit of a ceitain service (management function sets, such as Cofiguration management, Performance management, etc.), not a indivitual 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 prefdefined 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 of the operation includes a request, and an optional corresponding response when needed.
- Each MOC is defined as a 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 the RDN list.
- A generic service is defined in this framework, which contains 5 generic object accessing methods: createMO, deleteMO, getMOAttribute, setMOAttribute, and getPackages. And 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 to be sent from the Agent to the Manager should follow the format and behaviour defined in [OASIS WSN]. The controlling 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 interface

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.

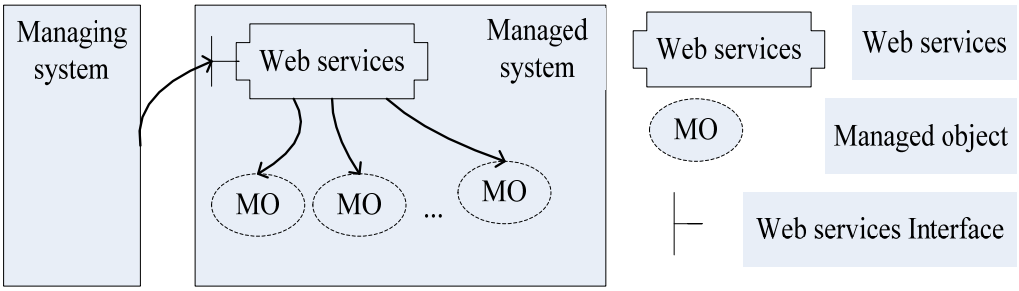

Figure 1 Web services role

Figure 1 shows how a managing system accesses a managed system that supports Web services interface. A Web services interface performs 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

A 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, a MO is the abstraction of such a resource that represents its properties for the purpose of management. A MO may include attributes that provide information used to characterize itself and operations that represent its behaviors. 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 behaviors. A service-oriented MO is a managed entity that represents a manageable service resource in terms of shared state and behavior where state and behavior are separated through outsourcing of the behavior to an assigned so-called "managing entity" (e.g., a service and its interface) that takes a steward role with regard to the behaviors of its allocated managed entities. Since MO's state and behavior can be separated, state can be described by XML Schema and behavior by Web services' interface in WSDL. One important benefit of using XML document to store MO's state is that Web services uses 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 generic managed object class

Managed object class is the 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 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 is 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 XML Schema specification, and it can be just mapped to the “element” in XML Schema. The attribute “objectClass” is used to identify the class type of this MO instance. The attribute “objectInstance” is used to uniquely identify a MO instance, and the data type is using NameType (also it has the same semantic of DN, Distinguished Name) as specified in (1). The attribute “packages” is a set of strings to indicate capacities the MO supports. The attribute “creationSource” indicates whether a MO is created by automatically in 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 Annex A.2.

(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 Annex A.2. 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 all the rest of MOs must directly or indirectly inherit from ManagedObject. Specialization of a "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 “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 showed 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 XML Schema only supports single inheritance for data types and it is not suggested to use multiple inheritance. If semantic of multiple inheritance is 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 a certain capabilities (for example, related attributes), or provide conditional support capabilities. A package can be defined as a XSD complexType, with a "_P" as its name suffix. When it is used, the element may have a type of the complexType representing for this package, with minOccurs="0" maxOccurs="1" as the qualifiers, which indicates 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 attribute 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	Descriptions
administrativeState	AdministrativeStateType	See [M.3701] for more details
availabilityStatus	AvailabilityStatusSetType	See [M.3701] for more details
backedUpStatus	BackedUpStatusType	See [M.3701] for more details
controlStatus	ControlStatusSetType	See [M.3701] for more details
creationSource Note	SourceIndicatorType	See [M.3701] for more details
externalTime	ExternalTimeType	
objectClass Note	ObjectClassType	It indicates a MOC
objectInstance Note	NameType	It indicates a MO instance
operationalState	OperationalStateType	See [M.3701] for more details
packages Note	StringSetType	It indicates the packages that are supported by a MO instance.
proceduralStatus	ProceduralStatusSetType	See [M.3701] for more details
standbyStatus	StandbyStatusType	See [M.3701] for more details
systemLabel	SystemLabelType	It indicates a label for a system.
unknownStatus	UnknownStatusType	See [M.3701] for more details
usageState	UsageStateType	See [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 Annex 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. Figure 2 gives the accessing procedure and the framework uses Web services technology to exchange information of MOs. Web services separates MOs' states and behaviours and exposes their behaviours through Web services interface. As 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 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 : EEUMERATION
setMOAttributes	- objectInstance : DN	- status

	- attributeNVMList : SEQUENCE OF {attributeName, attributeType, attributeValue, modifyType}	
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 a MO's attribute values 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 an request from the client, the server will return the requested attributes into the output parameter attributeNameAndValueList, where the attributeValue field will be encoded from an 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 a MO in existence. Besides using objectInstance to indicate the target MO whose values are to be modified, the operation also use a list of quadruples including attributeName, attributeType, attributeValue, and modifyType to set MO attributes. The first three attributes are the same as above. The modifyType 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 modifyType is optional, and if it is not specified, the “REPLACE” shall be assumed.

3) createMO - to create a 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 MO cannot be removed, the operation will return 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 actually 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 brings in some problems such as difficulty of data validation and comprehension. For Web services is service-oriented and service inheritance will brake up its loose coupling, it is not recommended. In order to keep data consistency of MOs, it can be considered using singleton design pattern.

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 Recommendation [ITU-T Q.818].

A complete XML Schema and WSDL interface definition for the generic MO accessing methods can be found in Annex A.3.

(R) OBJECT-2. An implementation of the MO Accessing methods shall support all the operations describe above, whose WSDL is defined in Annex A.3.10 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. Operations inheritance introduced will lead to weaken these characteristics such as good interoperability and loosely couple.

While attributes of managed objects are described with XML Schema, it can be followed the part 4.2 Deriving Types by Extension in XML Schema Part 0: Primer Second Edition to create attributes inheritance. Since attributes data types of base managed objects are defined in complex type in XML Schema, Subclass can extend the base managed objects data type by the value of *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 use for element inheritance, which supports the semantics for Managed Object attribute inheritance.

10.2 Considerations for inheritance of interface operation

Although higher version of WSDL (version 2.0) supports the syntax of inheritance for operations, but it is not widely supported by the industry. Lower versions of WSDL (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-service 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 a MOC, the XSD complexType is used for modelling the MOC. A XSD complexType contains a sequence, which can include one or more

XSD element(s). Each compleType corresponding to a 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 a MOC are defined as the elements in the complexType corresponding to a MOC. Note here the attribute is the conceptual property of an entity which is modelled as a MOC, it is not the same as the keywords "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 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 [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 the XML schema based modelling:

- All the attributes of a MOC is defined as a XSD complexType, the name of the MOC should have a "_C" as name suffix, with the first character in capital, so that this complexType can be distinguished from other normal data type definitions. For example: ManagedObject_C, Equipment_C.
- An attribute of a MOC is defined as an element within the complexType presenting a MOC, and the first character of an attribute name should be in lower case.

- A package can be defined as a XSD complexType, with a "_P" as its name suffix, with the first character in capital.
- A normal data type definition should have a "Type" as its name suffix, with the first character in capital, which helps to provide more readability. 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 using XML Schema

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 to define rules, semantics and structure for XML documents. A schema provides programmatic validation of a structured XML document. An XML Schema is defined using a XML format in a file with a ".xsd" extension.

12.1.2 Schema specification version

The namespace for XML Schema 1.1 http://www.w3.org/2001/XMLSchema is same as XML Schema for 1.0 document and the namespace for 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 namespace 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 in [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:

---

[1] The text in this clause is originally described in [b-ATIS-I-000001], with some updates to fit this Web services based management framework.

- 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 ability to redefine a type, group and attribute group definitions defined in a supporting schema. Type redefinition affects the elements in the including schema as well as 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.

12.2.2 Schema design patterns

There are following four structural design patterns commonly mentioned related 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 definition of element 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” is used.

12.2.3 Designing for resiliency

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 section 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 instance document to contain XML data not directly constrained by the content model of the defining XML Schema. This allows extensibility mechanism with 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 creation of non-deterministic content models which can cause issues with some XML parsers. A schema developer will have to be 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 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 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 Recommendaitons 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 is to be used for ITU-T generated schemas:

  "http://www.itu.int/schemas/<ITU-T document number>" or

  "http://www.itu.int/schemas/<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.
- Example Schema attributes for version 1.0 of the EAS schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.itu.int/schemas/itu-t/x.782"
   xmlns:eas="http://www.itu.int/schemas/itu-t/x.782"
   version="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 use of 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 dimension is more than 1, 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 nonNegativeInteger or positiveInteger simple type is preferred to integer if possible.
- When defining 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 enumeration facet to restrict the legal values of the simple type.
- Use Union Types if the data types take two or more different set 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 don’t form part of the document but are passed to the application to perform application specific functions. Processing instructions don’t have to follow 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 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 in advance anticipate all future needs of a schema. The author should utilize schema definition that provides flexibility and allow implementers to carry private or custom data. In general, schema authors should anticipate the need for supporting

private data by including constructs to support inclusion of some data in a generic way (for example name-value pair).

However such inclusion might not suffice 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 needs of extensions to specific portions of schema and allow users to add private openContent data.

2) Type Inheritance using <xsd:extension>.

Use of such extension is discouraged as this potentially will lead to interpretability issues. For example, if an implementer extends a standard schema using xsd:extension to add an 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 Annex A.2.
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 Annex 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 A.3.

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

```
<!-- XML Schema Definition for common date types to be used in this framework.
       Filename : x782.xsd
-->
<?xml version="1.0" encoding="UTF-8"?>
<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">

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

</xsd:schema>
```

A.2 XML Schema definition for generic Managed Object

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns:x782="http://www.itu.int/xml-namespace/itu-t/x.782"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.782" >
 <xsd:complexType name="ManagedObject_C">
   <xsd:sequence>
     <xsd:element name="objectClass" type="string"/>
     <xsd:element name="objectInstance" type="x782:NamteType"/>
     <xsd:element name="packages" type="x782:PackageListType"/>
     <xsd:element name="creationSource" type="x782:SourceIndicatorType"/>
   </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

A.3 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"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:nmfd="http://www.itu.int/xml-namespace/itu-t/x.782/MOService.xsd"
targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOService.xsd"
elementFormDefault="qualified" attributeFormDefault="unqualified">
       <complexType name="PackageList">
           <sequence>
               <element name="package" type="string" minOccurs="0"
maxOccurs="unbounded"/>
           </sequence>
       </complexType>
       <complexType name="AttributeNameList">
           <sequence>
               <element name="attributeName" type="string" minOccurs="0"
maxOccurs="unbounded"/>
           </sequence>
       </complexType>
       <element name="getMOAttributes">
           <complexType>
               <sequence>
                   <element name="objectInstance" type="string"/>
                   <element name="attributeNameList" type="nmfd:AttributeNameList"/>
               </sequence>
           </complexType>
```

```
        </element>
        <simpleType name="StatusType">
            <restriction base="string">
                <enumeration value="OperationSucceed"/>
                <enumeration value="OperationFailed"/>
            </restriction>
        </simpleType>
        <complexType name="AttributeValueType">
            <sequence>
                <any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
        </complexType>
        <complexType name="AttributeNameAndValueType">
            <sequence>
                <element name="attributeName" type="xsd:string"/>
                <element name="attributeType" type="xsd:string"/>
                <element name="attributeValue" type="nmfd:AttributeValueType"/>
            </sequence>
        </complexType>
        <complexType name="AttributeNameAndValueSetType">
            <sequence>
                <element name="attributeNameAndValue"
type="nmfd:AttributeNameAndValueType" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
        <element name="getMOAttributesResponse">
            <complexType>
                <sequence>
                    <element name="attributeNameAndValueList"
type="nmfd:AttributeNameAndValueList"/>
                    <element name="status" type="nmfd:StatusType"/>
                </sequence>
            </complexType>
        </element>
        <simpleType name="ModifyType">
            <restriction base="string">
                <enumeration value="REPLACE"/>
                <enumeration value="ADDValues"/>
                <enumeration value="REMOVEValues"/>
                <enumeration value="SETToDefault"/>
            </restriction>
        </simpleType>
        <complexType name="AttributeNVMType">
```

```
            <sequence>
                <element name="attributeName" type="string"/>
                <element name="attributeType" type="string"/>
                <element name="attributeValue" type="nmfd:AttributeValueType"/>
                <element name="modify" type="nmfd:ModifyType"/>
            </sequence>
        </complexType>
        <complexType name="AttributeNVMList">
            <sequence>
                <element name="attributeNVMList" type="nmfd:AttributeNVMType"
minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
        <element name="setMOAttributes">
            <complexType>
                <sequence>
                    <element name="objectInstance" type="string"/>
                    <element name="attributeNVMList" type="nmfd:AttributeNVMList"/>
                </sequence>
            </complexType>
        </element>
        <element name="setMOAttributesResponse">
            <complexType>
                <sequence>
                    <element name="status" type="nmfd:StatusType"/>
                </sequence>
            </complexType>
        </element>
        <element name="createMO">
            <complexType>
                <sequence>
                    <element name="objectClass" type="string"/>
                    <element name="objectInstance" type="string"/>
                    <element name="attributeNameAndValueList"
type="nmfd:AttributeNameAndValueList"/>
                </sequence>
            </complexType>
        </element>
        <element name="createMOResponse">
            <complexType>
                <sequence>
                    <element name="status" type="nmfd:StatusType"/>
                </sequence>
            </complexType>
```

```
        </element>
        <element name="deleteMO">
            <complexType>
                <sequence>
                    <element name="objectInstance" type="string"/>
                </sequence>
            </complexType>
        </element>
        <element name="deleteMOResponse">
            <complexType>
                <sequence>
                    <element name="status" type="nmfd:StatusType"/>
                </sequence>
            </complexType>
        </element>
        <element name="getPackages">
            <complexType>
                <sequence>
                    <element name="objectInstance" type="string"/>
                </sequence>
            </complexType>
        </element>
        <element name="getPackagesResponse">
            <complexType>
                <sequence>
                    <element name="status" type="nmfd:StatusType"/>
                    <element name="packages" type="nmfd:PackageList"/>
                </sequence>
            </complexType>
        </element>
</schema>
```

(2) ITU MO Access Service WSDL definition

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:nmfd="http://www.itu.int/xml-
namespace/itu-t/x.782/MOService.xsd" xmlns:nmfs="http://www.itu.int/xml-namespace/itu-
t/x.782/MOService.wsdl" name="MOService" targetNamespace="http://www.itu.int/xml-
namespace/itu-t/x.782/MOService.wsdl">
        <types>
            <schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
                <import namespace="http://www.itu.int/xml-namespace/itu-
t/x.782/MOService.xsd" schemaLocation="MOService.xsd"/>
            </schema>
```

```
</types>
<message name="getMOAttributesRequest">
    <part name="parameter" element="nmfd:getMOAttributes"/>
</message>
<message name="getMOAttributesResponse">
    <part name="parameter" element="nmfd:getMOAttributesResponse"/>
</message>
<message name="setMOAttributesRequest">
    <part name="parameter" element="nmfd:setMOAttributes"/>
</message>
<message name="setMOAttributesResponse">
    <part name="parameter" element="nmfd:setMOAttributesResponse"/>
</message>
<message name="createMORequest">
    <part name="parameter" element="nmfd:createMO"/>
</message>
<message name="createMOResponse">
    <part name="parameter" element="nmfd:createMOResponse"/>
</message>
<message name="deleteMORequest">
    <part name="parameter" element="nmfd:deleteMO"/>
</message>
<message name="deleteMOResponse">
    <part name="parameter" element="nmfd:deleteMOResponse"/>
</message>
<message name="getPackagesRequest">
    <part name="parameter" element="nmfd:getPackages"/>
</message>
<message name="getPackagesResponse">
    <part name="parameter" element="nmfd:getPackagesResponse"/>
</message>
<portType name="MOServicePortType">
    <operation name="getMOAttributes">
        <input message="nmfs:getMOAttributesRequest"/>
        <output message="nmfs:getMOAttributesResponse"/>
    </operation>
    <operation name="setMOAttributes">
        <input message="nmfs:setMOAttributesRequest"/>
        <output message="nmfs:setMOAttributesResponse"/>
    </operation>
    <operation name="createMO">
        <input message="nmfs:createMORequest"/>
        <output message="nmfs:createMOResponse"/>
    </operation>
    <operation name="deleteMO">
        <input message="nmfs:deleteMORequest"/>
```

```
            <output message="nmfs:deleteMOResponse"/>
        </operation>
        <operation name="getPackages">
            <input message="nmfs:getPackagesRequest"/>
            <output message="nmfs:getPackagesResponse"/>
        </operation>
    </portType>
    <binding name="MOServiceBinding" type="nmfs:MOServicePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getMOAttributes">
            <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOService/getMOAttributes"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace=" http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </input>
            <output>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace=" http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </output>
        </operation>
        <operation name="setMOAttributes">
            <soap:operation soapAction=" http://www.itu.int/xml-namespace/itu-
t/x.782/MOService/setMOAttributes"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </input>
            <output>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace=" http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </output>
        </operation>
        <operation name="createMO">
            <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOService/createMO"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </input>
            <output>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.itu.int/xml-namespace/itu-t/x.782/MOService/" use="encoded"/>
            </output>
        </operation>
        <operation name="deleteMO">
            <soap:operation soapAction="http://www.itu.int/xml-namespace/itu-
t/x.782/MOService/deleteMO"/>
            <input>
```

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

Appendix I

Overview of Web-service technology and application scenarios in network management interfaces

(This annex 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 a 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 encapsulates 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 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 source code but also data behind 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 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 lower the cost in developing new applications and running environment and integration.

I.2 Suitable and unsuitable application scenarios of Web-service 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 difficult. It is usually more complex when using other communication middleware.

- Application Integration

It is known applications often need to run on programs in one kind of platform and obtain data from or send data to applications in some other platforms. Even in the same platform, a variety of software provided by different manufacturers often need to be integrated. Over Web-services, applications can expose 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 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 though this open interface, and the change of interfaces will result less impact on the development of web service applications.

(2) Unsuitable application scenarios in general

As Web-Service is a loosely coupled technology mainly designed for application interoperability and integration in heterogeneous environment, it also has week points, such as lower execution speed and less encoding efficiency compared to some other technologies (such as CORBA or DCOM). In certain use cases, it may not be the best choices to use Web-Service. 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 large amount of data

(3) Considerations for Web-services application scenarios in network management

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

- B2B/C2B interface

- F interface, and

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

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

It may not be a good choice to use Web-Services on EMS-NE management interfaces, as they are usually provided by the same vender, 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 higher execution speed in LAN environment.

Bibliography

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

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

[b-ATIS-I-000001] ATIS Specification ATIS-I-000001 (2011), ATIS XML Schema Development Guidelines.

_____