COMMITTEE T1 – TELECOMMUNICATIONS
Working Group T1M1 **T1M1/2001-100 R3**
Kansas City, MO, September 10-11, 2001

DRAFT STANDARD

TITLE: **tML Guidelines for mapping UML notation to XML Schemas and vice versa**
SOURCE*: **Sprint**
PROJECT: **tML Mapping Guidelines**

ABSTRACT

This technical proposed standard provides guidelines for defining (generating) Telecommunications Markup Language (tML) Schemas based on Unified Modeling Language (UML) notation design models and vice versa.

This work is proposed to help (for example) TMN paradigm independent design models to be mapped with little or no effort to an eXtensible Markup Language (XML) implementation. Efforts underway in the ANSI T1 and ITU-T bodies to create implementation independent models will take advantage of this recommendation.

NOTICE

This contribution has been prepared to assist Accredited Standards Committee T1–Telecommunications. This document is offered to the Committee as a basis for discussion and is not a binding proposal on Sprint or any other company. The requirements are subject to change in form and numerical value after more study. Sprint specifically reserves the right to add to, amend, or withdraw the statements contained herein.

* CONTACT: Raymond E. Reeves; email: Raymond.E.Reeves@mail.sprint.com; Tel: 1-913-534-3405; Fax: 1-913-534-5136

TABLE OF CONTENTS

List of Figures

Draft Recommendation

tML Guidelines for mapping UML notation to XML schemas and vice versa

(2001)

1 Scope

The TMN architecture defined in Recommendation M.3010–2000 introduces concepts from distributed processing and includes the use of multiple management protocols. The initial TMN interface specifications for intra- and inter-TMN interfaces were developed using the Guidelines for the Definition of Managed objects (GDMO) notation from OSI Systems Management with Common Management Information Protocol (CMIP) as the protocol. The inter-TMN interface (X) included both CMIP and CORBA GIOP/IIOP as possible choices at the application layer.

Telecommunications Extensible Markup (tML), an application of the Extensible Markup Language (XML), is being considered for use in the TMN architecture primarily due to its flexibility for structured information definition. This acceptance is expected to enhance the availability of tML-based information syntax definition due to better development tools and widespread expertise in XML Schemas definition. This XML technology, developed by the World Wide Web Consortium (W3C), is also being considered by multiple industries and standardization bodies like ebXML, OASIS, BizTalk, etc. While GDMO/ASN.1 information models provide solutions for interoperability between manager and agent systems, tML defines structured information that does not required highly complex encoding mechanisms.

At the same time, efforts within the TMN community are striving to define a paradigm and technology independent set of specifications for TMN using the UML notation. This is identified as the Unified TMN Requirements, Analysis and Design.

1.1 Purpose

The scope of this contribution is to define guidelines suitable for the mapping of UML based information models to XML Schemas.

Re-using a generic information model for a variety of network technologies and network management applications and mapping to XML Schemas (or other technology) will speed the introduction of network services while keeping network management system development costs down.

A primary goal of the tML framework and this “XML Schema to UML mapping” recommendation (proposed standard) is the re-use of these information models by enabling their translation to XML Schemas in a quick (and automated) fashion with little change in semantics.

Applicability of the guidelines contained herein extends to all domains that follow a model driven approach based on UML for system development and specification. Examples include efforts in areas like T1M1 UMA/UOM, DSL Forum, etc.

1.2 Application

Recommendation M.3020 defines three phases in the development of a TMN specification. The three phases are Requirements, Analysis and Design. Figure 1 shows this process and the scope of this Recommendation for developing XML Schemas based specification relative to this process.

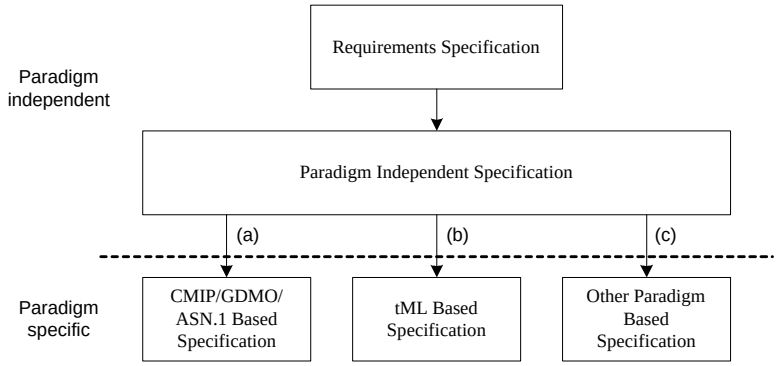


Figure 1. tML Based Specification

The requirements and analysis are specified using an approach that is not specific to a network management technology paradigm. The output from the analysis phase, the paradigm independent specification, is used as input to the paradigm specific design phase.

In the design phase, network management paradigm specific features are used to define information models. These paradigm specific specifications incorporate both behavior (normally in natural language) and formal interface signatures (e.g., GDMO/ASN.1, IDL, XML Schemas).

The arrows marked as (a), (b) and (c) show that the analysis output is mapped, for example, to a GDMO/ASN.1 based model to use with CMIP or XML Schema models to use with the choice of message transport mechanism, respectively (or IDL models to use with CORBA/IIOP). There are no prescriptive rules available at this time to generate these models. It may be possible to develop such rules in the future in M.3020. Meanwhile, this Recommendation proposes a set of guidelines and rules for the arrow shown as (b).

In developing the transformation from UML notation to XML Schemas, this recommendation uses an approach that not prescriptively translates every element of the syntax. Rather, the elements are translated from the UML model in a way that preserves most of the semantics.

This approach preserves the requirements and semantics of the models developed to meet the telecommunication context. It is applied when the managing and managed systems are designed to communicate using information exchange (instance documents) based on XML Schemas.

1.3 Issues

This recommendation does not address the use-case, state machine, activity, interaction and physical views.

2 References

2.1 Normative References

At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all 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.

- Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds,. W3C, 6 October 2000. See http://www.w3.org/TR/2000/REC-xml-20001006.
- Namespaces in XML, Tim Bray et al., eds,. W3C, 14 January 1999. See http://www.w3.org/TR/1999/REC-xml-names-19990114.
- XML Schema Part 1: Structures. Henry Thompson et al., eds,. W3C Recommendation, 2 May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-1-200010502.
- XML Schema Part 2: Datatypes . Paul Biron et al., eds,. W3C Recommendation, 2 May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-2-200010502.
- T1M1/2001-84R1. Proposed XML Schema for common tML types used in proposed standard XML Schemas for ANSI T1M1.3 – Coding and Language Data Representation (CLDR) standards.
- T1M1/2001-118R1. UML model and XML Schemas for DSL Service Flow-Thru fulfillment management interface.
- T1M1.5/2001-164R3. XML Schemas for UOM Volume II UOIM: UOM Volume III.

3 Definitions and Abbreviations

3.1 Definitions from W3C Recommendation XML Schema Part 1

The following terms used in this Recommendation are defined in the XML Schema Part 1: Structure (W3C Rec. XML Schema Part 1):

- XML Schema
- Schema component
- Target namespace
- Declaration
- Definition
- Type definition
- Simple type
- Complex type
- Type restriction
- Type extension
- Base type

3.2 Abbreviations

This Recommendation uses the following abbreviations:

ASN.1	Abstract Syntax Notation #1
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DSL	Digital Subscriber Line
DSLsp	DSL Service Provisioning
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Interoperability Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	HTTP Secure
IDL	Interface Definition Language
ITU-T	International Telecommunication Union – Telecom
MIB	Manage Information Base
OAM&P	Operations, Administration, Maintenance, and Provisioning
tML	XML for TMN
TMN	Telecommunications Management Network
UML	Unified Modeling Language
UTRAD	Unified TMN Requirements, Analysis and Design
W3C	World Wide Web Consortium
XML	Extensible Markup Language

4 Unified Modeling Language (UML) Translation

This section provides guidelines for creating XML Schema information models from structural information contained in UML models. The sections below describe how each of the GDMO templates and ASN.1 types are to be translated to XML Schemas.

4.1 Model Management View

UML model management view deals with packages and dependency relationships between them.

4.1.1 UML Package

In UML, models are partitioned into packages. Every element of a model must belong to one package. UML imposes no rule on how to partition models into packages, however the appropriate partitioning of models based on rational principles, like common functionality is recommended.

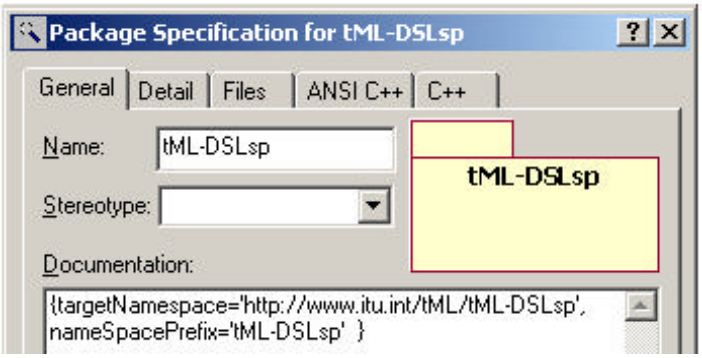
In effect, a UML package defines a namespace for all the UML elements contained in it. UML package names are used in external references for referring to definitions contained in other packages from the referring package. In essence, the UML package concept maps to the XML Schema and namespace.

This recommendation proposes mapping UML packages (considering the package name) to XML Schemas with a target namespace so that namespace prefixes could be used in external references. If no namespace property is specified in the UML package, the package name will be used as both the XML Schema name and target namespace name. The location of the package element property in the UML model is out of scope for this recommendation.

Proposed package element properties include:

UML property	***tML significance***	***Default***
targetNamespace	targetNamespace name for the schema related with the package.	Package name
nameSpacePrefix	Suggested prefix (xmlns:) to be used in the schemas using declarations from this schema.	Package name
schemaLocation	Suggested location for storing the schema. This is the value to be used in “schemaLocation=” value pairs.	Package name + “.xsd”
id	Equivalent to the id attribute for the <schema> element.	Package name
version	Equivalent to the version attribute for the <schema> element.	Date+Time

For example,



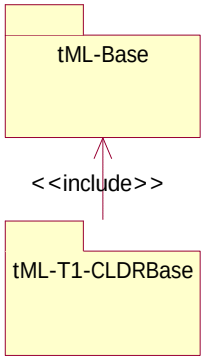

```
<schema
    targetNamespace="http://www.itu.int/tML/tML-DSLsp"
    xmlns:tML-DSLsp="http://www.itu.int/tML/tML-DSLsp"

    xmlns="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="2001/06/20 14:26:34"
    id="tML-DSLsp.xsd"
    …
>
<!-- etc -->
</schema>
```

Dependencies among packages, both access and import dependencies, shall be mapped to the “import” mechanism provided by XML Schemas. Because XML Schema does not allow multiple default namespaces, it is required that every imported definitions be prefixed with the corresponding namespace prefix.

XML Schema does not support nested namespaces, for this reason, classifiers contained within children packages will be treated as if contained in the parent package. Where name collisions exist, classifiers will be renamed to have a prefix containing its package’s name.

XML Schemas allow for a special relation between schemas and their target namespaces. Two XML Schemas may have the same target namespace and, in consequence, a containment dependency could exist between them. tML specific stereotype is suggested in the package dependencies. This new stereotype for the dependency is <<include>> (or <<generalization>>) –this information could be conveyed as a property of the dependency– and would map to the “include” mechanism provided by XML Schemas.

For example,




```
<?xml version="1.0"?>
<schema
   targetNamespace="http://www.itu.int/tML/tML-Base"
   xmlns:tML-CLDR="http://www.itu.int/tML/tML-Base"
   id="tML-T1-CLDRBase.xsd"
   …
   …
>
<include schemaLocation="tML-Base.xsd"/>

<!-- etc -->
<!-- etc -->

</schema>
```

The only visibility semantics of packages mapped to XML Schemas is public.

4.2 Static View

This UML model view models concepts of a domain and internal concepts created as part of the implementation of an application. The main parts of this view are classes and their relationships: association, generalization, and various kinds of dependencies. This view is the foundation on which the other views are built and is expressed as class diagrams.

4.2.1 Classifiers

Classifiers, which are discrete concepts in the model, include class, interface, and data type. Class is the most familiar term. These classifiers can be mapped to XML Schema data types (simple or complex). unless the classifier is stereotyped, in which case the corresponding XML Schema concept will depend on the stereotype used. UML data type classifiers map nicely to simple types (simpleType) and the other classifiers can be mapped to complex types (complexType), unless otherwise specified through stereotypes.

As a UML class classifier, XML Schema data types are general descriptions of the structure for elements: its content model. tML Instance documents have to obey the constraints defined in the content model of the base type.

The relationship of classifiers and XML Schemas is as follows:

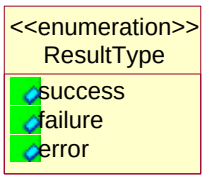
Classifier Properties	**Type**
Name	Name
Abstraction	Complex types in tML can be designated as abstractions much like the <<abstract>> stereotype in UML. To identify a complex type as abstract, the types 'abstract' has to be assigned the "true" value.
Visibility	Only public is supported (other require separate mechanisms)
Operations	New Type with notation: ClassifierName_OperationName_"Operation" and parameters as elements.
Attributes	Elements
Attribute containment	The only supported containment semantic is "by value" (Further study of XLink and XPointer is needed to support "by reference" semantics.)

4.2.1.1 Data types

4.2.1.1.1 Enumeration

An enumeration is a data type whose instances for a list of named literal values. Usually, both the enumeration name and its literal values are declared. XML Schemas allow the creation of enumerations on simpleTypes. By default, enumerations are considered to be based on the string base type. However, in order for UML to express the diverse types of enumerations that XML Schemas allow, enumerations not based on strings have to have refinement association to its base data type or simpleType. If the dependency from an enumeration to a datatype is not specified, refinement should be assumed.

For instance,




```
<simpleType name="ResultTypeType">
    <restriction base="string">
        <enumeration value="success"/>
        <enumeration value="failure"/>
        <enumeration value="error"/>
    </restriction>
</simpleType>
```

Note: The values of all 'value' properties of an enumeration or datatype classifier are considered to be part of the list of allowed values of the enumeration. For example, a property-list for ResultType containing '{value="unknown"}' extends ResultType to {success, failure, error, unknown}.

For embedded enumerations where the attributes have as its type an enumeration with the list of values included like, it must translate to an element with anonymous type.

For instance,

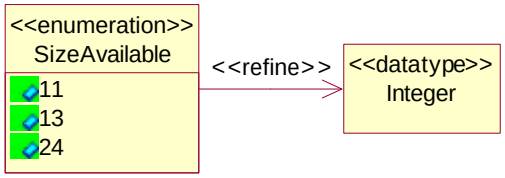
<<req>> ResultType: enumeration {success, failure, error}

```
<element name="ResultType">
    <simpleType>
        <restriction base="string">
            <enumeration value="success"/>
            <enumeration value="failure"/>
            <enumeration value="error"/>
        </restriction>
    </simpleType>
</element>
```

4.2.1.1.2 General datatypes

Any UML datatype stereotyped classifier may refer to UML datatypes (e.g. String, Integer, Single, Object, Long, Boolean, Byte, Date, Double, Currency, etc.) or a XML Schema datatype (e.g. hexInteger, anyType, duration, dateTime, etc.). The corresponding formal mappings from XML Schema to UML datatype, and vice versa is left for further study because many of them are language dependent.

For instance,




```
<simpleType name="SizeAvailableType">
    <restriction base="integer">
        <enumeration value="11"/>
        <enumeration value="13"/>
        <enumeration value="24"/>
    </restriction>
</simpleType>
```

4.2.1.2 Class

Class is the most common classifier and most of the content of this section applies to other classifiers.

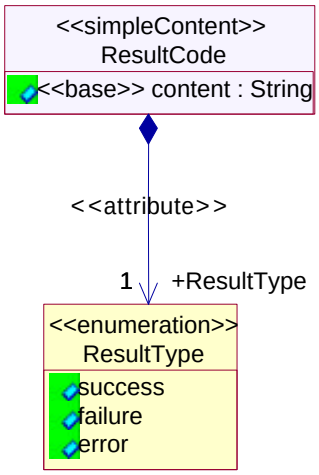
Classes can be stereotyped or can be left undifferentiated as well. Several stereotypes carry specific semantics for tML specific implementation. The following is a list of such identified stereotypes:

UML Stereotype	***XML Schema significance***
elementGroup	In this case, the class maps to a XML element group with name equal to the 'elementGroup' stereotyped class. For convenience reasons, the XML element group name may include the 'Group' postfix.
attributeGroup	The class maps to a tML attribute group with name equal to the 'attributeGroup' stereotyped class. For convenience reasons, the tML attribute group name may include the 'Attributes' postfix.
simpleContent	Often, it is needed to have an element which contains several attributes. This can be model through a UML classifier with <<simpleContent>> stereotype. In such case, the only attribute allowed in the classifier attribute compartment is the 'content' attribute with the <<base>> stereotype. The attribute type specifies the simpleContent attribute model. In the case of this being absent.
choice	Mapping choice classes uses the same rules of other classes. However, the UML attributes (XML elements) defined within the specific class (XML complex type) are treated as part of a choice group within the XML type content model.

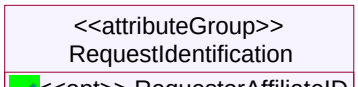
For example:

UML	***XML Schema***
	

```
<group name="ServiceAddressAndConstraintGroup">
    <sequence>
        <element name="ServiceAddress"
                 type="tML-DSLBase:NationalAddressType"/>
        <element name="ServiceRequestConstraint"
                 type="tML-DSLBase:ServiceRequestConstraintType"
                 minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</group>
```




```
<element name="ResultCode">
    <complexType>
        <simpleContent>
            <extension base="string">
                <attribute name="ResultType"
                           type="tML-DSLBase:ResultTypeType"
                           use="required"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
```




```
<attributeGroup name="RequestIdentificationAttributes">
    <attribute name="RequestorAffiliateID"
               type="tML-DSLBase:tradingPartnerIdCodeType" use="optional"/>
    <attribute name="RequestorOrderID"
               type="tML-DSLBase:orderIDType" use="required"/>
    <attribute name="RequestorVersionID"
               type="tML-DSLBase:versionIDType" use="optional"/>
</attributeGroup>
```

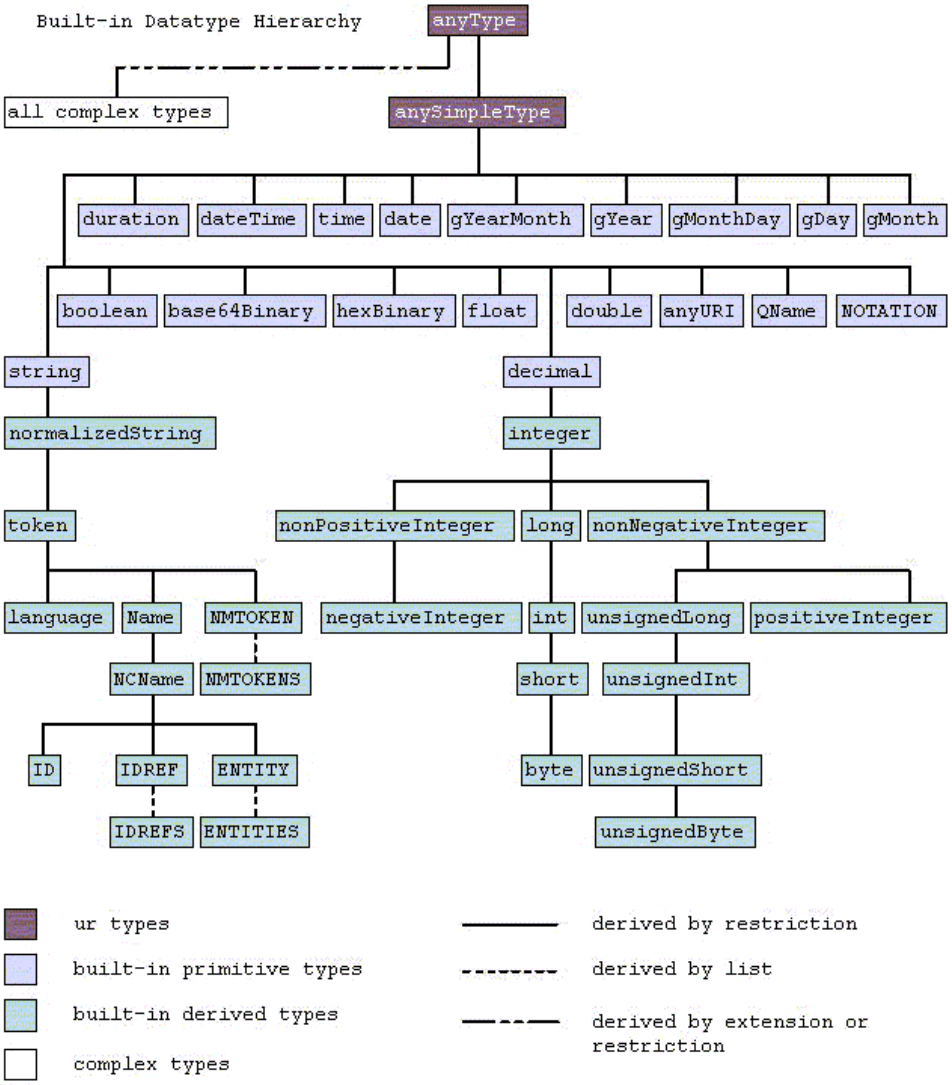
4.2.1.2.1 Basic Types

XML Schema defines the following hierarchy of built-in basic datatypes (a.k.a 'types'), derived from and defined in the XML Schemas specification[1], to which UML predefined data types may be translated:

[1] See XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, §3.

As per the W3C XML Schema Specification, "[p]rimitive datatypes are those that are not defined in terms of other datatypes; they exist ab initio." And, "[d]erived datatypes are those that are defined in terms of other datatypes."

Some of the built-in datatypes include: anyType, boolean, byte, double (for double-precision floating-point numbers), float (for single-precision floating-point numbers), long (for large integers), object (for object references), integer, short (for small integers), hexBinary, string, and Char (as defined in XML 1.0 2nd edition). Enumerations, which are a basic type in many languages (including IDL) is obtained through the enumeration constraining facet that could be applied to most built-in types.



This guideline uses the *string* type for all strings. The actual character encoding is defined in the encoding declaration contained in the xml processing instruction. In an encoding declaration, the values "UTF-8", "UTF-16", "ISO-10646-1", "ISO-8859-2", …, "ISO-8859-9" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "ISO-8859-1", "ISO-8859-2", …, "ISO-8859-9" should be used for the parts of ISO 8859, and the values "ISO-2022-JP", "Shift_JIS", and "EUC-JP" should be used for the various encoded forms of JIS X-0208-1997.

A sample declaration of wide character encoding follows:

```
<?xml version="1.0" encoding="UTF-16"?>
<? ?  ? ? ? ? ="1 9 9 9 ? 3 ? 1 ? ">
    <? >? ? ? ? </? >
    <? ? >? ? ? ? ? ? ? ? ? ? ? ? ? </? ? >
    <!-- ? ? ? ?  -->
    <? ? >&? ? ? ;</? ? >
    <? ?  ? ? ? ? ? ="? ? ? ? "/>
</? ? >
```

4.2.1.2.2 Attributes

The UML attribute concept maps primarily to XML Schema elements, however, under certain circumstances and the choice of the modeler, it could be mapped to a XML Schema attribute.

The default syntax for UML attributes is:

[<<stereotype>>] visibility name : type-expression [multiplicity ordering] = initial-value { property-string }

- Where the *stereotype*s recognized are those resembling the multiplicity semantics. For instance, `opt` is understood as `0..1`, `req` is construed as `1..1`.
- Where only public *visibility* is supported by XML Schemas. Other visibility semantics require other mechanisms for enforcement. (Actually, all forms of nonpublic visibilities are language-dependent.)
- Where the attribute *name* shall map to the element name.
- Where the attribute *type-expression* refers to other Classifiers or tML type.
- Where *multiplicity* indicates a range of allowable cardinalities a value sequence may assume. The minOccurs and maxOccurs facets are used.
- Where *ordering* is not expressible with just tML unless other advanced features are included (like is the case of XPointer and XPath)
- Where the *initial-value* of simple typed elements maps to the default facet supported only by simpleTyped elements in tML.

For example:

```
public measurement : integer 2..5 ordered
```

or

```
<<2..5>> public measurement : integer
```

would map to

```
<element name="measurement" type="integer" minOccurs="2" maxOccurs="5"/>
```

And,

```
<<opt>> private salary : float
```

would map to

```
<element name="salary" type="double" minOccurs="0"/>
```

The following is a list of properties that can be associated with Classifiers attributes and datatypes, which map directly to the equally named corresponding XML Schema facet:

UML Properties	
maxLength	maxInclusive
minLength	minExclusive
length	maxExclusive
minInclusive	

4.2.1.2.3 Operations

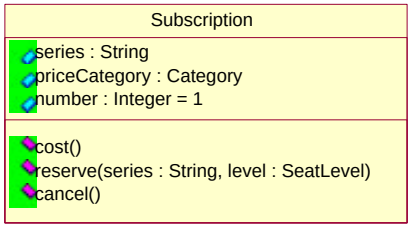
Though XML Schemas are not good for modeling the operational aspects of a system, the UML operation concept could map well to a new complexType with naming convention *ClassifierName_OperationName_"Operation"* and parameters treated as XML elements or UML classifier attributes.

The default syntax for UML attributes is:

[<<stereotype>>] visibility name (parameter-list) : return-type-expression { property-string }

- Where the *parameter-list* is interpreted as a list of attribute definitions.
  - The syntax used is *kind name : type-expression = default-value*
  - Where *kind* could be in, out or inout. No enforceable semantics are carried.
- Where *return-type-expression* cannot be expressed in tML terms.
- Where the rest of semantics are treated much like in the case of attributes.

For example:




```
<complexType name="SubscriptionType">
    <sequence>
        <element name="series" type="string"/>
        <element name="priceCategory" type="ns:CategoryType"/>
        <element name="number" type="integer" default="1" />
    </sequence>
</complexType>

<complexType name="Subscription_Cost_Operation">
    <complexContent>
        <extension base="ns:InteractionType"/>
    </complexContent>
</complexType>
<complexType name="Subscription_Reserve_Operation">
    <complexContent>
        <extension base="ns:InteractionType">
            <sequence>
                <element name="series" type="string"/>
                <element name="level" type="ns:SeatLevelType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="Subscription_Cancel_Operation">
    <complexContent>
        <extension base="ns:InteractionType"/>
    </complexContent>
</complexType>
```

4.2.2 Relationships

UML defines several types of relationship between classifiers. The various relationships and suggested mapping to XML Schema concepts follow:

UML Relationship	***Suggested XML Schema mapping***	***Comment***
Generalization & Realization	Simple and Complex types differ: - Complex type derivation by extension - Simple type derivation by restriction and union for multiple inheritance	Overloading and multiple inheritance for complex types are discouraged

Flow & Usage	No mapping suggested in this version	
Dependencies	Package (access & import) dependencies are supported through import mechanism	'include' and 'generalize' package dependency stereotypes map to tML specific 'include' mechanisms.
Constraints & OCL statements	These can be incorporated as <annotation>s within the contextual elements	
XOR dependencies between associations	When two or more non-stereotyped associations (compositions, aggregations, etc.) from class X to classes Y1, Y2, ..., Yn exist, these are mapped as part of a choice group in the complex Type definition of X.	
AND dependencies between associations	When two or more non-stereotyped associations (compositions, aggregations, etc.) from class X to classes Y1, Y2, ..., Yn exist and each has a minimum occurrence of zero, these are mapped to a sequence group in the complex Type definition of X. The minimum occurrence (UML multiplicity) of each of the elements (UML attributes) is to be one, regardless of the initial UML multiplicity of the association end. The minimum occurrence of the sequence group will be zero to allow for the case where all the elements are absent.	

4.2.2.1 Associations

Association express discrete connections between objects or other instances in a system. Because of the rich semantics that associations carry and some of XML Schemas rules, the following initial concept and semantics mapping are recommended:

UML concept	***tML mapping***
Association class	A complex type which includes an ID element for both of the participating classes
Qualified association	Left for further study
Bi-directionality	Because of tML documents strict tree structure, (it is initially suggested that) all associations should have navigability in exactly one direction.
Aggregation and composition	A strong containment is assumed (composition). If the association has a <<attribute>> stereotype, the composite type will include the part as an attribute. Otherwise it will be included as an element. The element or attribute defined should have the same name as the role name for the part class association end. The multiplicity will be the same as the association end multiplicity.

4.2.3 Constraints

UML includes the definition of a constraint language, called Object Constraint Language (OCL), useful for describing existence and universal properties. Mapping of these semantics to XML Schema concepts requires further study. Meanwhile, it is suggested mapping UML constraint statement (i.e. OCL) to an XML Schema annotation element (<annotation>).

4.3 Use Case View

The mapping of this view's concepts to XML Schema semantics is left for further study.

4.4 State Machine View

The mapping of this view's concepts to XML Schema semantics is left for further study.

4.5 Activity View

The mapping of this view's concepts to XML Schema semantics is left for further study.

4.6 Interaction View

The mapping of this view's concepts to XML Schema semantics is left for further study.

4.7 Physical View

The mapping of this view's concepts to XML Schema semantics is left for further study.

5 Style Idioms for tML Specifications

This section defines a set of style idioms for the XML Schema to be used in mapped UML models. Having a set of style idioms will result in XML Schema specifications with a consistent style. This may require some additional work by editors, but this extra effort is worth the increased readability of the tML specifications. It is important to keep in perspective that style conventions are for the benefit of the reader, not necessarily to the benefit of the author.

5.1 Use Consistent Indentation

This section demonstrates the indentation style that may be used in the XML Schemas. As an example, an excerpt from the Digital Subscriber Line (DSL) Forum Service Provisioning (DSLsp) definitions is shown below:

```
<attributeGroup name="InteractionIdentificationAttributes">
    <annotation>
        <documentation xml:lang="">
            WT-063 Refactored Table 5, 6 and 7
        </documentation>
    </annotation>
    <attribute name="InteractionType" type="tML-DSLBase:ServiceFulfillmentTypeType" use="required"/>
    <attribute name="EntityID" type="tML-DSLBase:tradingPartnerIdCodeType" use="required"/>
    <attribute name="InteractionTimeStamp" type="tML-DSLBase:DateAndTimeType" use="required"/>
    <attribute name="InterfaceVersion" type="tML-DSLBase:versionIDType" use="optional"/>
</attributeGroup>
```

5.2 Use Consistent Case for Identifiers

Several languages enforce case rules while others have de-facto rules. These rules allow readers to easily distinguish identifiers of different type leading to increased readability. XML Schema does enforce case, so the following rules are proposed.

- Type declarations shall have every embedded word capitalized except for the first word capitalized.
- All others (e.g., elements and entities) shall have the first letter of every embedded word capitalized.

5.3 Decouple types from elements

Whenever a possible, create an element separated from the definition of its type and reuse types already defined. See the example:

```
<element name="DaysOfTheWeek" type="DaysOfTheWeekType"/>
<complexType name="DaysOfTheWeekType">
    <!-- etc -->
</complexType>

<!-- tML instance document -->
<sunnyDays>
    <bitStringNamed>
        <sunday/>
        <monday/>
        <wednesday/>
    </bitStringNamed>
</sunnyDays>
```

5.4 Use a Consistent Type Suffix

Append the suffix “Type” to all XML Schema derived types. This idiom increases readability by clearly separating type identifiers from other identifiers.

5.5 Use a Consistent Suffix for Attribute Group Types.

For attribute groups use a suffix of “Attributes” to distinguish them.

5.6 Use a Consistent Suffix for Element Group Types.

For element groups use a suffix of “Group” to distinguish them.

For example:

```
<group name="PrequalificationInformationGroup">
    <annotation>
        <documentation>
            NOTE: Table 63 element 21.7 (CO Capacity Constraint) is underspecified: GenericConstraint assumed.
        </documentation>
    </annotation>
    <sequence>
        <element name="PrequalificationReferenceID" type="tML-DSLBase:GenericIDType" minOccurs="0"/>
        <element name="QualifiedAddress" type="tML-DSLBase:NationalAddressType" minOccurs="0"/>
        <element name="QualifiedTelephoneNumber" type="tML-DSLBase:TelephoneNumberType" minOccurs="0"/>
        <element name="QualifiedLoopCircuitID" type="tML-DSLBase:GenericIDType" minOccurs="0"/>
        <element name="CentralOffice" type="tML-DSLBase:NationalCentralOfficeType" minOccurs="0"/>
        <element name="LoopCharacteristics" type="tML-DSLBase:NationalLoopCharacteristicsType" minOccurs="0"/>
        <element name="COCapacityConstraint" type="tML-DSLBase:GenericConstraintType" minOccurs="0"/>
    </sequence>
</group>
```

5.7 Assume No Global Identifier Spaces

To reduce name collisions and promote reuse, all identifiers shall be scoped to a particular context (e.g., module, and interface).

5.8 Global Level Definitions

All named type and exportable element definitions shall be at the global level. Nested type definitions or local types shall be unnamed. Avert cluttering the global space with unnecessary type and element definitions.

5.9 Explicit vs. Implicit namespaces

In order to ease creation of XML Schemas and readability, have qualified element namespace prefix.

6 Examples

Refer to T1M1/2001-84R1 and T1M1.5/2001-164R3 for extensive examples of applying the guidelines defined in this document.