SWIFTStandards XML design rules

version 2.3

Technical Specification

Table of contents

1 Introduction

XML is a technical standard defined by W3C (the World Wide Web Consortium) and leaves a lot of freedom for the exact way it is used in a particular application. Therefore, merely stating that XML is used is not sufficient, one must also explain HOW it will be used.

The use of XML is part of the overall approach for the development of SWIFTStandards. This development focuses on the correct definition of a business standard using modelling techniques[1]. The resulting business standard is captured in UML (Unified Modelling Language[2]) and is stored in an electronic repository, the "SWIFTStandards Repository"[3]. Business messages are defined in UML class diagrams and XML is then used as a physical representation (i.e. the syntax) of the defined business messages. A set of **XML design rules**, called **SWIFTStandards XML**, define in a very detailed and strict way how this physical XML representation is derived from the business message in the UML class diagram.

This document explains these XML design rules.

This document does NOT explain how a message should be created in UML. It explains, once a message is created in UML, how it will be mapped into XML.

2 Mapping rules from UML to SWIFTStandards XML

2.1 General mapping rules

Mapping rules from UML to SWIFTStandards XML are governed by the following design choices:

- SWIFTStandards XML representation to be as structured as possible:
  - Business information is expressed as XML elements/values;
  - Metadata information is expressed as XML attributes. XML attributes are not to be conveyed 'on the wire' in the XML instance, unless required to remove ambiguity.
- The current work is based on W3C's Recommendation of May, 2001.
- The names used in SWIFTStandards XML are the XML names or, when absent, the UML names.

[1] This approach is called SWIFTStandards Modelling. You can find more information on this approach in the SWIFTStandards White Paper on www.swift.com.

[2] You can find more information about UML on the Object Management Group website at: http://www.omg.org/uml

[3] You can find more information on this repository in the SWIFTStandards White Paper on www.swift.com.

- SWIFTStandards XML elements are derived from the UML representation of a business message. They can only be derived from UML-classes, UML-roles or UML-attributes.
- Each SWIFTStandards XML element must be traceable to the corresponding UML model element.
- Currently SWIFTStandards XML only runtime Schemas are generated. Runtime schema's only contains information required to validate XML instances. No documentation nore implementation information (e.g elementID, version, etc.) is mentioned.

2.2 SWIFTStandards XML elements

For the SWIFTStandards XML runtime Schema, any SWIFTStandards XML element has the following structure:

```
<SWIFTStandardsXMLTag
[xsi:type="class_name"][RepresentationClassAttribute="value"]>
```

2.2.1 SWIFTStandards XMLTag

SWIFTStandards XMLTag is assigned according to following rules:

For a SWIFTStandards XML element derived from a class if that class contains the stereotype <<message>>[4]:

- The XML name of the class or by default the name of the class.

For a SWIFTStandards XML element derived from a role:

- The XML name of the role or by default the name of the role. If no rolename is specified in the UML model, the name (XML name or name by default) of the class which is at the end of the aggregation.

For a SWIFTStandards XML element derived from an attribute:

- The XML name of the attribute or by default the name of the attribute.

2.2.2 xsi:type

2.2.2.1 In the schema

By using xsi:type in the instance, the schema does not need to define any additional attribute on types. The xsi:type implicitely refers to a type defined in the schema.

[4] Classes that don't contain the stereotype <<message>> do NOT have a corresponding XML element.

2.2.2.2 In the corresponding instance

In case of polymorphism, the attribute “xsi:type” is required to choose the desired type in the SWIFTStandards XML instance.

summarizing:

SWIFTStandards XML element derived from	Type
Class	Class name
Role	Name of the class at the end of the aggregation
Attribute	• Name of the class of the attributes’ type • attribute type name (for primitive types)

Remark: by name, it is meant the XML name or by default the UML name.

2.2.3 Representation Class Attribute

When user defined-datatypes are stereotyped by a certain representation classes, an XML attribute might be required to remove ambiguity. See the chapter on data types for more details

2.3 Specific mapping rules

All model elements, defined accordingly to the SWIFTStandards methodology, are based on following UML structures. Hence, by defining the conversion rules from those structures into SWIFTStandards XML we can convert any UML model into its corresponding SWIFTStandards XML Schema and instance.

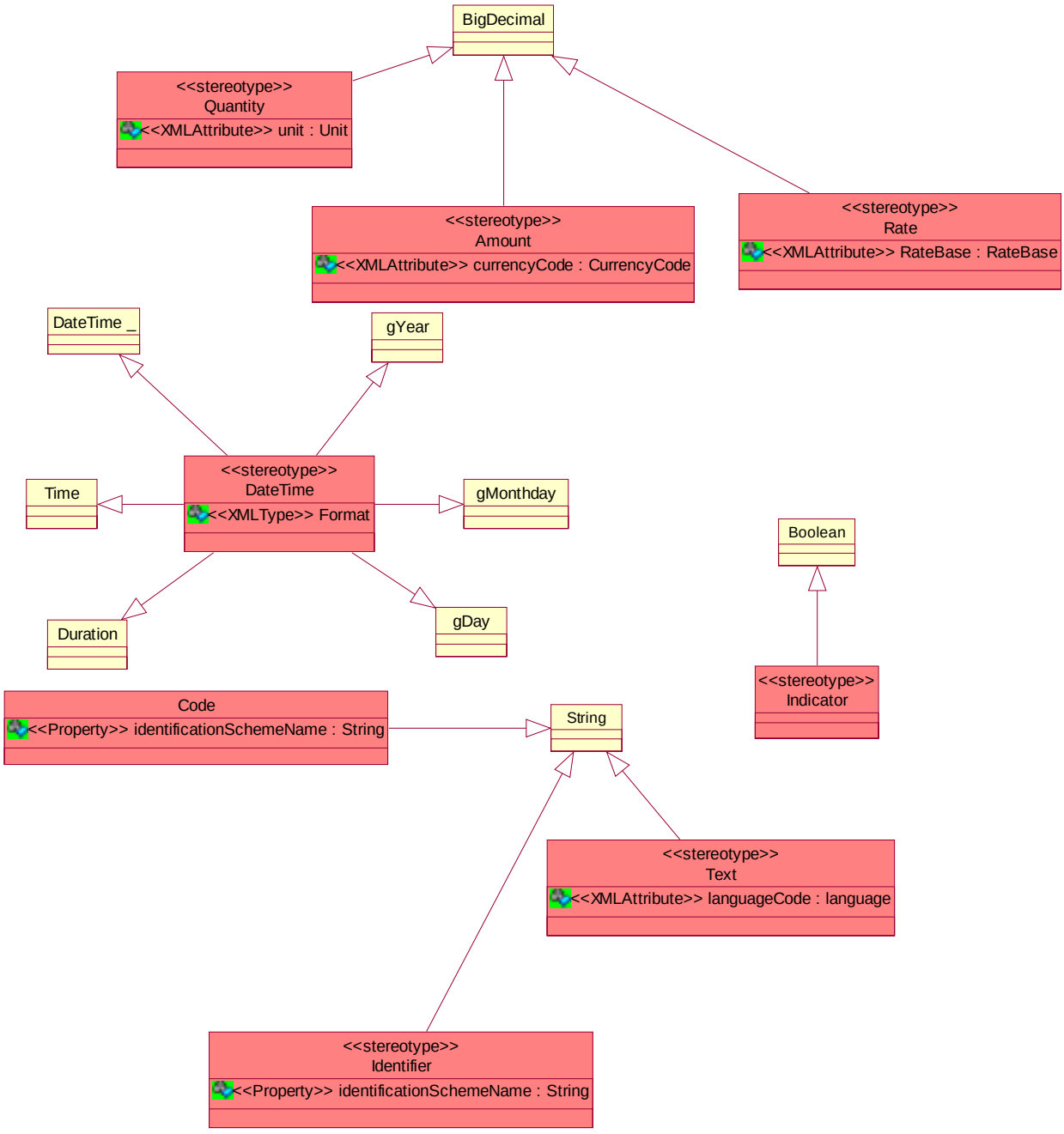
2.3.1 Data types

In a message model, all class attributes have a type, which we call **data types** for the purpose of this chapter. Data types define the structure of a class attribute.

118

119 2.3.1.1 Representation class meta-model

120



Notes:

Each data type is identified by a class diagram and stereotyped by a representation class. A representation class has a number of characteristics that are passed on ('inherited by') all data types that are using that representation class. In this way, characteristics common to a number of datatypes are grouped together.

Stereotype <<XMLAttribute>> indicates that the values this attribute can be declared in the XML Schema in case of ambiguity, and will appear in the XML instance.

Stereotype <<Property>> (not shown here) indicates that the values this attribute will NEVER be declared in the XML Schema.

Stereotype <<XMLType>> (only used in representation class DateTime) indicates that any user defined data type will have to declare the primitive datatype (Time, gDay, gMonth,….) it will use.

2.3.1.2 Primitive Data types

SWIFTStandards XML primitive data types are encoded as defined by W3C, defined at http://www.w3.org/TR/xmlschema-2/#dt-encoding. Following XML primitive types are supported:

UML Name	XML Name	Description
String	string	Set of finite sequences of UTF-8 characters
Boolean	boolean	Has the value space of boolean constants "True" or "False"
Integer	integer	Corresponds to 32 bits integer type
BigDecimal	decimal	Arbitrary precision decimal numbers
Date	date	Corresponds to a date. See ISO 8601. Format CCYY-MM-DD
Time	time	Corresponds to a time. See ISO8601. Format HH:MM:SS +- offset to UTC
DateTime	dateTime	Corresponds to a date and time. See ISO8601. Format CCYY-MM-DDTHH:MM:SS +- offset to UTC
Duration	duration	Corresponds to a period in time. See ISO8601. Format PnYnMnDTnHnMnS
gDay	gDay	It is a set of one-day long, annually periodic instances. The time zone must be UTC. Lexical representation:--MM-DD.
gMonth	gMonth	Represents a time period that starts at midnight on the first day of the month and lasts until the midnight that ends the last day of the month. Lexical representation: --MM--.

gYear	gYear	Represents a time period that starts at the midnight that starts the first day of the year and ends at the midnight that ends the last day of the year. It is a set of one-year long, non-periodic instances. Lexical representation: CCYY
gMonthday	gMonthday	It is a set of one-day long, monthly periodic instances. Lexical representation: ---DD. The time zone must be UTC.

2.3.1.3 User-defined data types

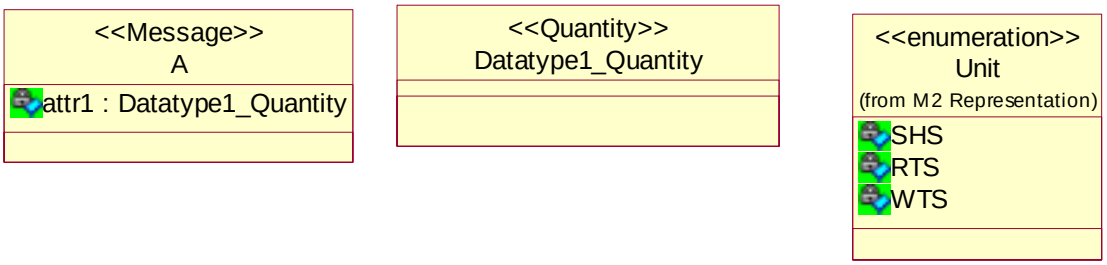
It is possible to define non-primitive data types by deriving either from a primitive type or from another non-primitive data type. Remark that in UML neither primitive nor non-primitive data types may have attributes. Those non-primitive datatypes can be used as UML types for UML attributes with the added benefit that the value space of the original primitive type (e.g. String) can be constrained by introducing invariants on the non-primitive data type. Those invariants will be mapped to facets when generating XML Schemas.

In order to apply facets, the XML types that are generated for those data types must be simpleTypes or complexTypes with simpleContent, and not complexTypes[5].

A user-defined data type maps to an XML SimpleType. This SimpleType restricts an XML primitive type.

Where necessary (in case of ambiguity), the representation class attribute maps to an XML attribute.

2.3.1.3.1 Data type using representation class <<Quantity>>



[5] XML Schema validation constraint: Facets cannot be applied to complexTypes without simpleContent.

Properties:

- Since the representation class Quantity (see metamodel) has an attribute with a type named Unit which is stereotyped as being a <<XMLAttribute>>, the corresponding Schema defines for element <attr1> an attribute named ‘unit’ with a enumerated list of values a specified in the Class ‘Unit’.
- An enumerated value is constrained within a list of possible values.
- The values for the enumerated items are taken from the UML initial value given to each of the UML enumerated attributes.
- Suppose this data type has an additional constraint (=XML facet) that the maximum quantity may not exceed 20000 units.

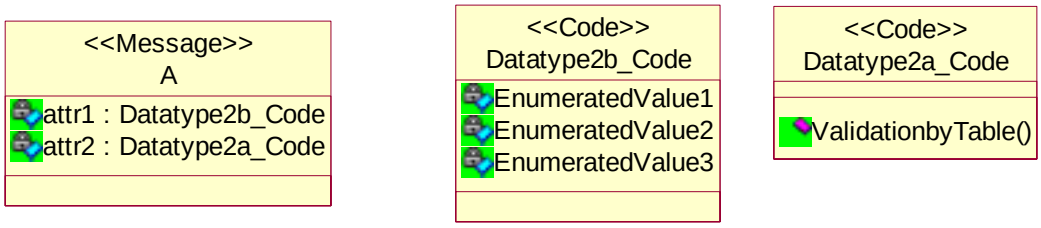
Instance:

```
<A>
  <attr1 unit=”SHS”>1000</attr1>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype1_Quantity"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype1_Quantity">
  <xs:simpleContent>
   <xs:restriction base="xs:decimal">
       <xs:maxInclusive value="20000">
      <xs:attribute name="unit" type="Unit"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="Unit">
   <xs:restriction base="xs:string">
      <xs:enumeration value="SHS"/>
      <xs:enumeration value="RTS"/>
      <xs:enumeration value="WTS"/>
   </xs:restriction>
</xs:simpleType>
```

2.3.1.3.2 Data type using representation class <<Code>>



Properties:

- Each user-defined datatype using <<Code> can indicate whether the list is an internal list (i.e. specified in the schema), or external (i.e. not specified in the schema). This is done using the invariant 'ValidationbyTable'. Datatype2b_Code is an enumeration of which one of the Enumerated Values has to be chosen in the instance.
- An enumerated value is constrained within a list of possible values.
- The values for the enumerated items are taken from the UML initial value given to each of the UML enumerated attributes.

UML	**SWIFTStandards XML instance**
Class contains an enumeration of possible values	SWIFTStandards XML element contains the chosen value

Instance:

```
<A>
  <attr1>EnumeratedValue2</attr1>
  <attr2>AnythingGoesHere</attr2>
</A>
```

Schema:

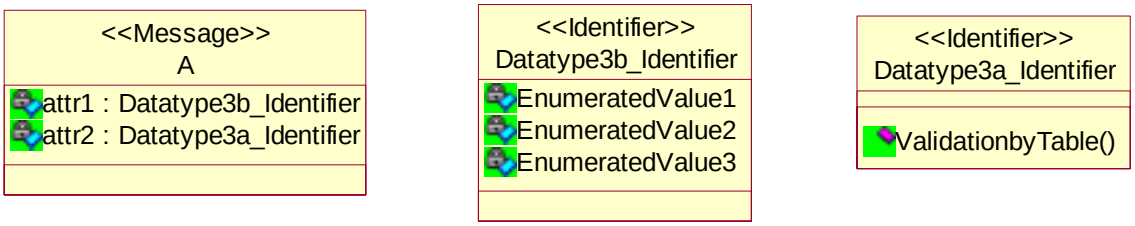
```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype2b_Code"/>
      <xs:element name="attr2" type="xs:Datatype2a_Code "/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype2b_Code">
   <xs:restriction base="xs:string">
      <xs:enumeration value="EnumeratedValue1"/>
      <xs:enumeration value="EnumeratedValue2"/>
      <xs:enumeration value="EnumeratedValue3"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Datatype2a_Code">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

198

199

200 **2.3.1.3.3 Data type using representation class <<Identifier>>**

201



202

203 Properties:

204 • Each user-defined datatype using <<Identifier> can indicate whether the list is an
205 internal list (i.e. specified in the schema), or external (i.e. not specified in the schema).
206 This is done using the invariant 'ValidationbyTable'. Datatype3b_Identifier is an
207 enumeration of which one of the Enumerated Values has to be chosen in the instance.

208 • An enumerated value is constrained within a list of possible values.

209 • The values for the enumerated items are taken from the UML initial value given to each
210 of the UML enumerated attributes.

211

UML	SWIFTStandards XML instance
Class contains an enumeration of possible values	SWIFTStandards XML element contains the chosen value

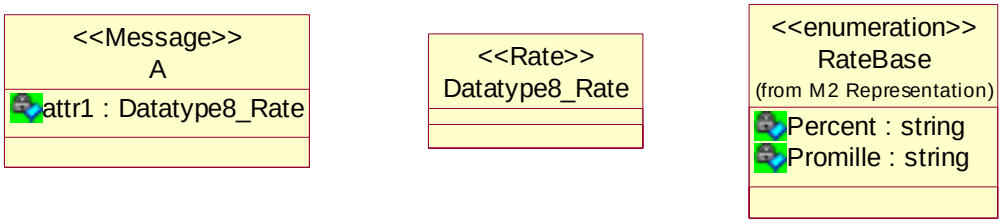
Instance:

```
<A>
  <attr1>EnumeratedValue2</attr1>
  <attr2>AnythingGoesHere</attr2>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype3b_Identifier"/>
      <xs:element name="attr2" type="xs:Datatype3a_Identifier "/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype3b_Identifier">
   <xs:restriction base="xs:string">
      <xs:enumeration value="EnumeratedValue1"/>
      <xs:enumeration value="EnumeratedValue2"/>
      <xs:enumeration value="EnumeratedValue3"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Datatype3a_Identifier">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

2.3.1.3.4 Data type using representation class <<Rate>>



Properties:

- Since the representation class Rate (see metamodel) has an attribute with a type named RateBase which is stereotyped as being a <<XMLAttribute>>, the corresponding Schema defines for element <attr1> an attribute named 'ratebase' with a enumerated list of values a specified in the Class 'RateBase'.
- An enumerated value is constrained within a list of possible values.
- The values for the enumerated items are taken from the UML initial value given to each of the UML enumerated attributes.

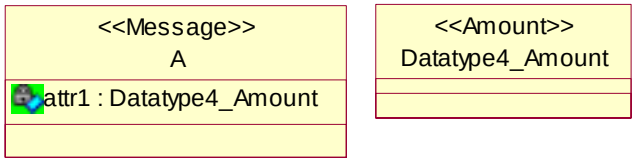
Instance:

```
<A>
  <attr1 ratebase="Percent">95.6</attr1>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype8_Rate"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype8_Rate">
  <xs:simpleContent>
   <xs:restriction base="xs:decimal">
      <xs:attribute name="ratebase" type="RateBase"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="RateBase">
   <xs:restriction base="xs:string">
      <xs:enumeration value="Percent"/>
      <xs:enumeration value="Promille"/>
   </xs:restriction>
</xs:simpleType>
```

2.3.1.3.5 Data type using representation class <<Amount>>



Properties:

- Since the representation class Amount (see metamodel) has an attribute with a type named CurrencyCode which is stereotyped as being a <<XMLAttribute>>, the corresponding Schema should define for element <attr1> an attribute named 'currencyCode' with a enumerated list of values a specified in the Class 'CurrencyCode'. However in this case, since we do not own this list (owned by ISO), it is considered to be an external list to avoid having to update the standard each time one of the values of the code list changes. Hence the XML attribute must appear in the instance (to avoid ambiguity), but the content is NOT validated by Schema.

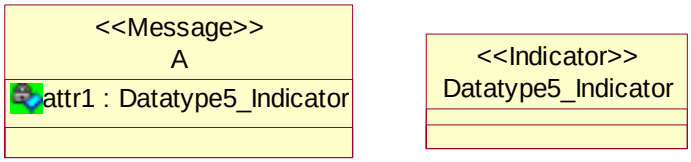
Instance:

```
<A>
  <attr1 currencyCode=”USD”>95.6</attr1>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype4_Amount"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype4_Amount">
  <xs:simpleContent>
   <xs:restriction base="xs:decimal">
      <xs:attribute name="currencyCode" type="CurrencyCode"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="CurrencyCode">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

2.3.1.3.6 Data type using representation class <<Indicator>>



Properties:

- A datatype stereotyped by representation class <<Indicator>> indicates that the attribute must have a boolean value (true or false).

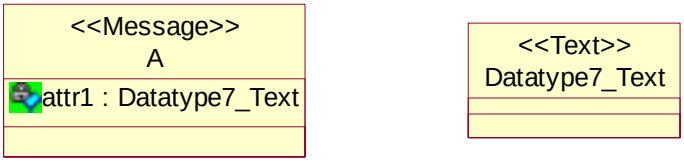
Instance:

```
<A>
  <attr1>true</attr1>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype5_Indicator"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype5_Indicator">
   <xs:restriction base="xs:boolean">
   </xs:restriction>
</xs:simpleType>
```

2.3.1.3.7 Data type using representation class <<Text>>



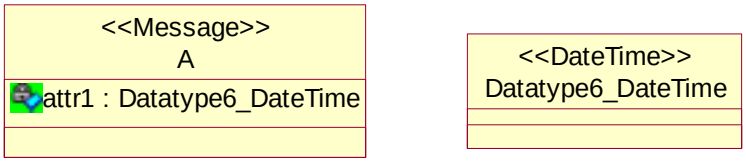
Instance:

```
<A>
  <attr1>any narrative text</attr1>
</A>
```

Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype7_Text"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype7_Text">
  <xs:simpleContent>
   <xs:restriction base="xs:string">
      <xs:attribute name="languageCode" type="xs:language"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

2.3.1.3.8 Data type using representation class <<DateTime>>



Properties:

- Representation class 'DateTime' has a meta attribute Format which is stereotyped <<XMLType>>. This means that any datatype that is using representation class <<DateTime>> has to indicate from which XML primitive datatype it is restricting.
- Suppose an additional constraint is added namely that the date should be equal or later than January first, 2002.

Instance:

```
<A>
  <attr1>2002-11-23</attr1>
</A>
```

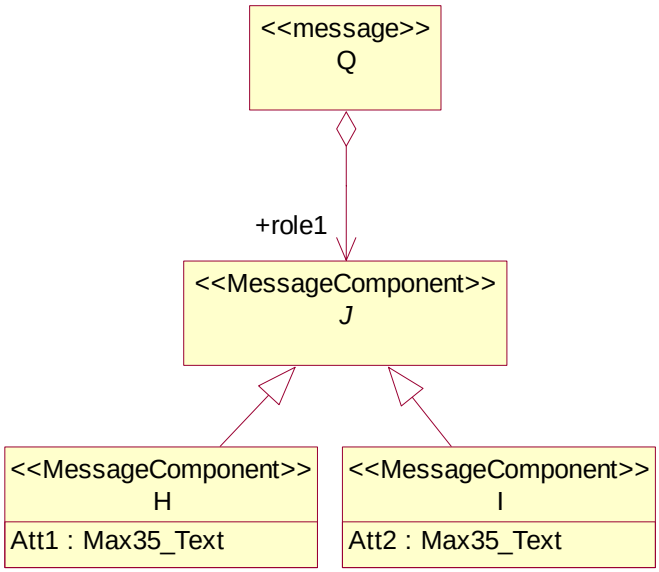
Schema:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype6_DateTime"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype6_DateTime">>
   <xs:restriction base="xs:dateTime">
      <xs:minInclusive value="2002-01-01T00:00:00"/>
   </xs:restriction>
</xs:simpleType>
```

2.3.1.4 Enumerated types

2.3.1.4.1 Basic pattern

- In the example below, two different types can play role1: either Att1 or Att2.
- In the SWIFTStandards XML representation, a SWIFTStandards XML attribute is introduced to express the actual type.



Instance:

```
<Q xmlns="urn:swift:xsd:$Q" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <role1 xsi:type="H">
      <Att1>data1</Att1>
   </role1>
</Q>
```

or

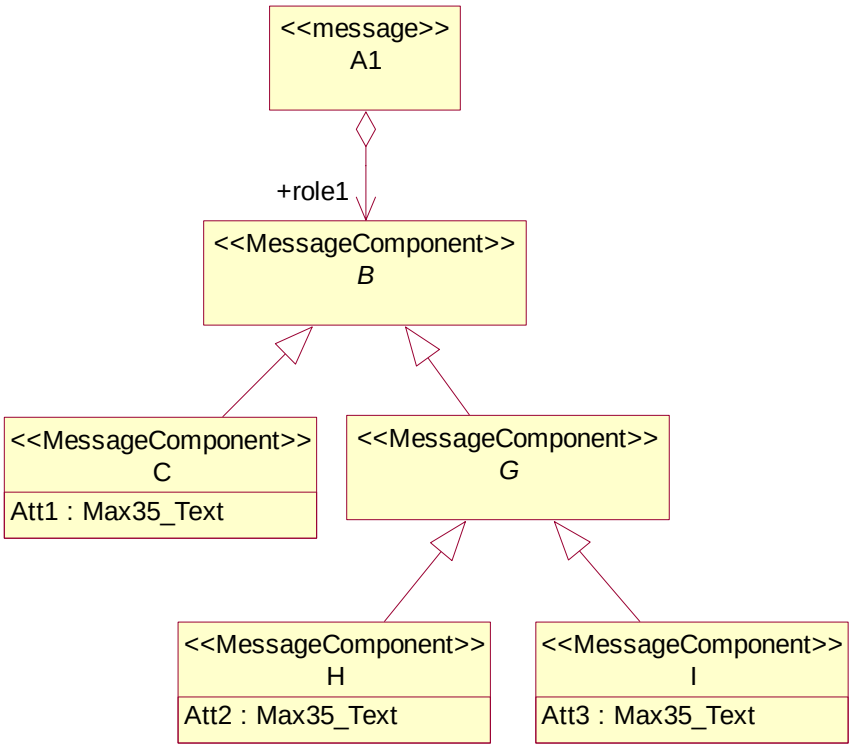
```
<Q xmlns="urn:swift:xsd:$Q" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <role1 xsi:type="I">
      <Att2>data2</Att2>
   </role1>
</Q>
```

Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 15:58:10-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$Q"
targetNamespace="urn:swift:xsd:$Q">
<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
  <xs:element name="Q" type="Q"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="Q">
 <xs:sequence>
  <xs:element name="role1" type="J"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H">
 <xs:complexContent>
  <xs:extension base="J">
   <xs:sequence>
    <xs:element name="Att1" type="Max35_Text"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="J" abstract="true"/>

<xs:complexType name="I">
  <xs:complexContent>
```

```
   <xs:extension base="J">
    <xs:sequence>
     <xs:element name="Att2" type="Max35_Text"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
  <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

2.3.1.4.2 Re-use pattern



Instance:

```
<A1 xmlns="urn:swift:xsd:$A1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <role1 xsi:type="C">
      <Att1>data1</Att1>
   </role1>
</A1>
```

or

```
<A1 xmlns="urn:swift:xsd:$A1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <role1 xsi:type="H">
      <Att2>data2</Att2>
   </role1>
</A1>
```

or

```
<A1 xmlns="urn:swift:xsd:$A1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <role1 xsi:type="I">
      <Att3>data3</Att3>
   </role1>
</A1>
```

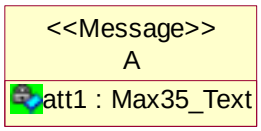
Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 15:58:10-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$A1"
targetNamespace="urn:swift:xsd:$A1">
<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
   <xs:element name="A1" type="A1"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="A1">
 <xs:sequence>
   <xs:element name="role1" type="B"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="G" abstract="true">
 <xs:complexContent>
  <xs:extension base="B"/>
 </xs:complexContent>
```

```
</xs:complexType>

<xs:complexType name="B" abstract="true"/>

<xs:complexType name="C">
 <xs:complexContent>
  <xs:extension base="B">
   <xs:sequence>
    <xs:element name="Att1" type="Max35_Text"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="I">
 <xs:complexContent>
  <xs:extension base="G">
   <xs:sequence>
    <xs:element name="Att3" type="Max35_Text"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="H">
 <xs:complexContent>
  <xs:extension base="G">
   <xs:sequence>
    <xs:element name="Att2" type="Max35_Text"/>
   </xs:sequence
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
  <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

2.3.2 Class

UML	XML instance
Class name with a role name	Role becomes an element. The class itself has no corresponding SWIFTStandards XML element.
Class name without a role name: • The class is aggregated but the role name is not given; or • The class has the stereotype <<message>>	The class name becomes the SWIFTStandards XML element name



Instance:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att1>data</att1>
</A>
```

Schema:

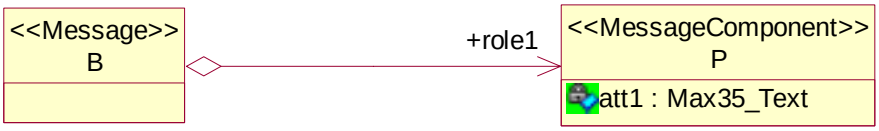
```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 05 16:21:43-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$A"
targetNamespace="urn:swift:xsd:$A">
   <xs:element name="Document" type="Document"/>

   <xs:complexType name="Document">
      <xs:sequence>
         <xs:element name="A" type="A"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="A">
      <xs:sequence>
         <xs:element name="att1" type="Max35_Text"/>
```

```
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="Max35_Text">
      <xs:restriction base="xs:string">
         <xs:length value="35"/>
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```

2.3.3 Simple composition

- A parent-child relationship between two classes is expressed by a role;
- The parent-class maps to a SWIFTStandards XML element with its name as the tag (see pattern "class name without a role");
- The role of the child-class maps to a SWIFTStandards XML element tag. The child class is not mapped.

UML	SWIFTStandards XML instance
Parent class	See "Class" pattern
Child class	SWIFTStandards XML element with role name as tag. This element is contained within the parent element



Instance:

```
<B xmlns="urn:swift:xsd:$B" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <role1>
      <att1>data</att1>
   </role1>
</B>
```

Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 05 16:21:43-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$B"
targetNamespace="urn:swift:xsd:$B">
```

```
<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
   <xs:sequence>
      <xs:element name="B" type="B"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="B">
   <xs:sequence>
      <xs:element name="role1" type="P"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="P">
   <xs:sequence>
      <xs:element name="att1" type="Max35_Text"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
   <xs:restriction base="xs:string">
      <xs:length value="35"/>
   </xs:restriction>
</xs:simpleType>

</xs:schema>
```

2.3.4 Class attributes

- A class can also contain attributes;
- A class attribute is described using a name and a type;
- By default ,the first SWIFTStandards XML child elements within its parents are the attributes, followed by the roles. However, you can define the sequence of all the child elements belonging to a class.

UML	SWIFTStandards XML instance
Parent class	See "Class" pattern
Child class	SWIFTStandards XML element with role name as tag. This element is contained within the parent element.
Class containing attributes	SWIFTStandards XML elements with attribute name as tag. This element is contained within the parent element.

Instance:

```
<D xmlns="urn:swift:xsd:$D" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att1>false</att1>
   <role1>
      <att2>data2</att2>
   </role1>
</D>
```

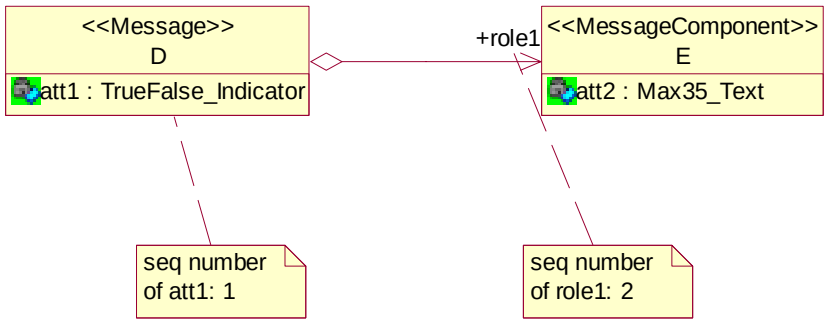
Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 05 16:21:43-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$D"
targetNamespace="urn:swift:xsd:$D">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
   <xs:sequence>
      <xs:element name="D" type="D"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="D">
   <xs:sequence>
      <xs:element name="att1" type="TrueFalse_Indicator"/>
      <xs:element name="role1" type="E"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="E">
   <xs:sequence>
      <xs:element name="att2" type="Max35_Text"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
   <xs:restriction base="xs:string">
      <xs:length value="35"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TrueFalse_Indicator">
   <xs:restriction base="xs:boolean"/>
```

```
</xs:simpleType>

</xs:schema>
```

2.3.4.1 Element order

To manage the order in which XML elements are generated from a given UML model, each UML attribute and role (automatically or manually) gets assigned a sequence number (see previous schema and instance).

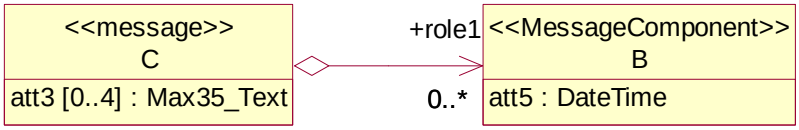


2.3.5 Composition of vectorial attributes (Collections)

- The cardinality expresses the number of occurrences of elements. The default value is 1, in which case it can be omitted; else it is represented as a **range** e.g **`0..*`** .
- Use a range-cardinality to express a collection of elements, which can be represented either as a collection of attributes or roles. In the example below, C contains a collection of A's expressed as attributes (att3) and a collection of Bs expressed as roles (role1).
- Schemas can validate exactly the cardinality.

Cardinality	Description	Schema representation
`1`	`Exactly one`	Element name="A"
`0..1`	`Optional`	Element name="A" minOccurs="0"

		maxOccurs=”1”
0..n	Any number of occurrences	Element name=”A” minOccurs=”0” maxOccurs=”unbounded”
1..n	At least one	Element name=”A” minOccurs=”1” maxOccurs=”unbounded”
1..4	From 1 to 4	Element name=”A” minOccurs=”1” maxOccurs=”4”
0..3	From 0 to 3	Element name=”A” minOccurs=”0” maxOccurs=”3”



Instance :

```
<C xmlns="urn:swift:xsd:$C" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att3>data1a</att3>
   <att3>data1b</att3>
   <role1>
      <att5>2001-01-01</att5>
   </role1>
</C>
```

Schema:

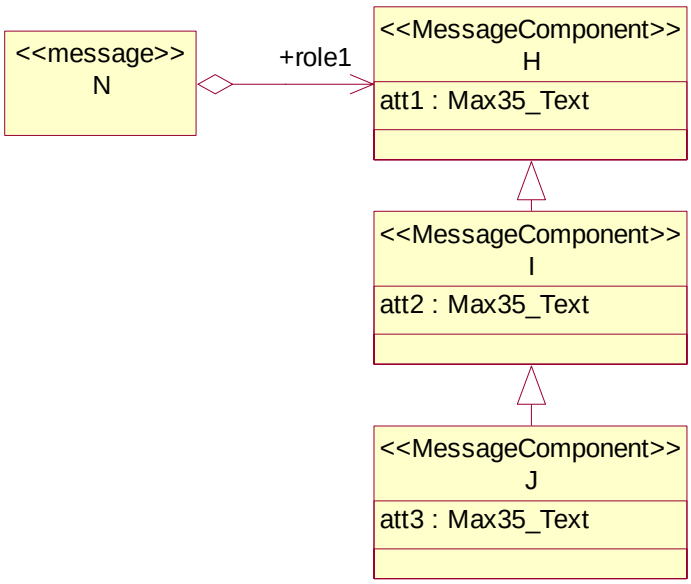
```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 13:40:40-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$C"
targetNamespace="urn:swift:xsd:$C">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
    <xs:element name="C" type="C"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="C">
 <xs:sequence>
  <xs:element name="att3" type="Max35_Text" minOccurs="0" maxOccurs="4"/>
  <xs:element name="role1" type="B" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="B">
 <xs:sequence>
   <xs:element name="att5" type="xs:dateTime"/>
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
   <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

2.3.6 Inheritance

It is possible to re-use business elements by specializing existing elements. This process - also called virtual containment - impacts element order and generated Schemas.

- In the example below the business element H contains an attribute att1. The business element I, which re-uses H, contains att2 and att1; the latter attribute is inherited from H. The business element J, which re-uses I, contains att3, att2 and att1; the last two attributes being inherited from I respectively H.
- This means that a container N containing H, can also contain I, as I "is-a" H; etc… This process is



Instance:

```
<N xmlns="urn:swift:xsd:$N" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <role1 xsi:type="H">
      <att1>data1</att1>
   </role1>
</N>
```

or

```
<N xmlns="urn:swift:xsd:$N" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
```

```
   <role1 xsi:type="I">
      <att1>data1</att1>
      <att2>data2</att2>
   </role1>
</N>
```

or

```
<N xmlns="urn:swift:xsd:$N" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <role1 xsi:type="J">
      <att1>data1</att1>
      <att2>data2</att2>
      <att3>data3</att3>
   </role1>
</N>
```

Schema:

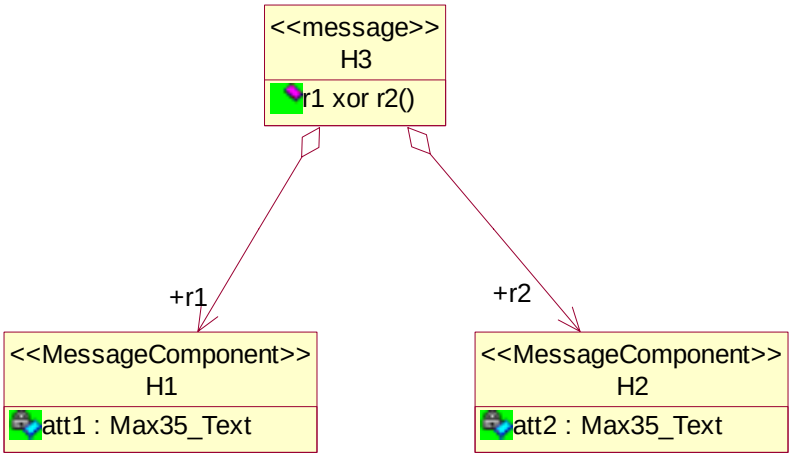
```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 13:40:40-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$N"
targetNamespace="urn:swift:xsd:$N">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
   <xs:element name="N" type="N"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="N">
 <xs:sequence>
   <xs:element name="role1" type="H"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="I">
 <xs:complexContent>
   <xs:extension base="H">
    <xs:sequence>
      <xs:element name="att2" type="Max35_Text"/>
   </xs:sequence>
   </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="H">
 <xs:sequence>
   <xs:element name="att1" type="Max35_Text"/>
 </xs:sequence>
</xs:complexType>

```

```
<xs:complexType name="J">
 <xs:complexContent>
  <xs:extension base="I">
   <xs:sequence>
     <xs:element name="att3" type="Max35_Text"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
   <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Notes:

- Inherited attributes appear first;
- Inheritance is cumulative: always add attributes, never remove them;
- It is an error in the pattern to redefine an attribute that already exists in a base class.
- XML schemas do not support multiple inheritance.

2.3.7 Enumerated roles using XOR invariant



Instance:

```
<H3 xmlns="urn:swift:xsd:$H3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <r1>
  <att1>data1</att1>
 </r1>
</H3>
```

or

```
<H3 xmlns="urn:swift:xsd:$H3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <r2>
  <att2>data2</att2>
 </r2>
</H3>
```

Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 16:55:10-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$H3"
targetNamespace="urn:swift:xsd:$H3">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
  <xs:element name="H3" type="H3"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H3">
 <xs:sequence>
  <xs:choice>
   <xs:element name="r1" type="H1"/>
   <xs:element name="r2" type="H2"/>
  </xs:choice>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H2">
 <xs:sequence>
  <xs:element name="att2" type="Max35_Text"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H1">
 <xs:sequence>
  <xs:element name="att1" type="Max35_Text"/>
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
```

```
  <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Note: multiplicity for enumerated roles is treated as follows:

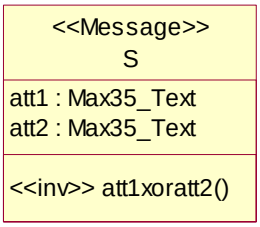
UML notation	**UML notation**	**Schema notation**	**means**
r1 0..1	r2 0..1	minOccurs=”0” maxOccurs=”1”	r1 or r2 may be present, but not both. This means both may be absent as well.
r1 0..n	r2 0..n	minOccurs=”0” maxOccurs=”unbounded”	r1 or r2 may be present up to n times, but not both. This means both may be absent as well.
r1 1	r2 1	-	r1 or r2 must be present, but not both (= XOR).
r1 1..n	r2 1..n	minOccurs=”1” maxOccurs=”unbounded”	r1 or r2 must be present up to n times, but not both (= XOR).
r1 0..n	r2 1..n	A choice between <xsd:element name= « r1 » with minOccurs=”0” maxOccurs=”unbounded” and <xsd:element name= « r2» minOccurs=”1” maxOccurs=”unbounded”	r1 may be present up to n times or r2 must be present up to n times, but not both (= XOR).

Note: some rules regarding the XOR in UML:

- Any XML name may be given to the operation

- the XOR operation has to be declared in a specific way in its “operation specification box”.
- It is not allowed to make an XOR between a role of the current class and a role of a sub- or superclass.
- The XOR invariant only applies to the roles mentioned in the XOR. Consequently, some roles may not be part of the XOR. Hence when roles are added, they are not part of the XOR until they are also added in the XOR invariant.

2.3.8 Enumerated attributes using XOR invariant




```
<S xmlns="urn:swift:xsd:$S" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att1>data1</att1>
</S>
```

or

```
<S xmlns="urn:swift:xsd:$S" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att2>data2</att2>
</S>
```

Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 16:55:10-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$S"
targetNamespace="urn:swift:xsd:$S">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
  <xs:element name="S" type="S"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="S">
 <xs:sequence>
```

```
  <xs:choice>
   <xs:element name="att1" type="Max35_Text"/>
   <xs:element name="att2" type="Max35_Text"/>
  </xs:choice>
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string"
  <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

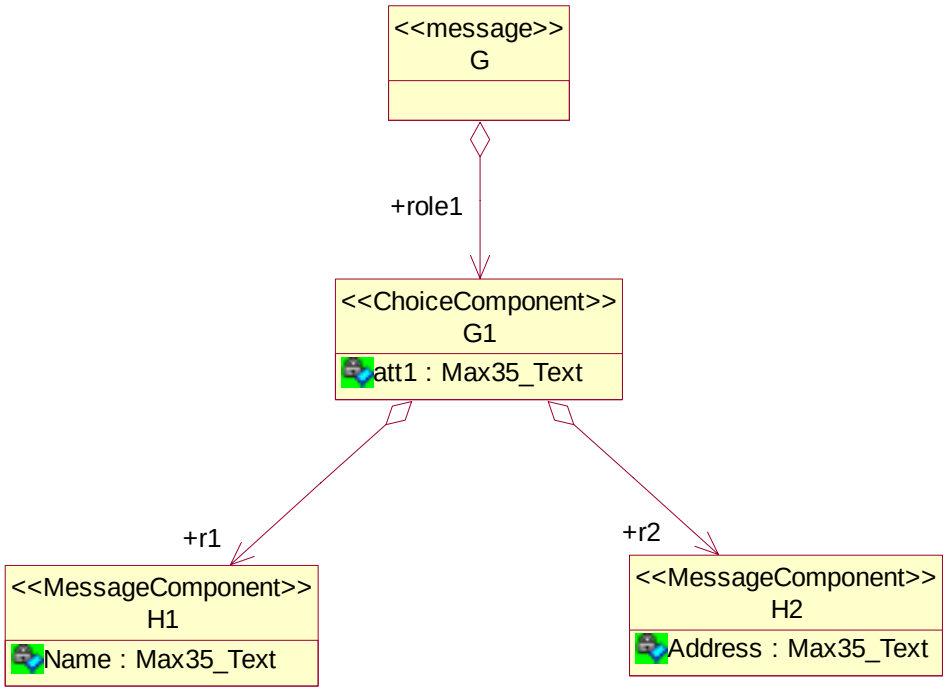
Note: some rules regarding the XOR in UML:

- Any valid XML name may be given to the operation
- the XOR operation has to be declared in a specific way in its “operation specification box”.
- It is not allowed to make an XOR between an attribute of the current class and an attribute of a sub- or superclass.
- The XOR invariant only applies to the attributes mentioned in the XOR. Consequently, some attributes within the class may not be part of the XOR. Hence when attributes are added to the class, they are not part of the XOR until they are also added in the XOR invariant.

2.3.9 Enumerated roles and attributes using <<choice>> stereotype

This pattern models a choice between roles and/or attributes.

All roles between the superclass containing the <<choice>> stereotype and its subclasses are part of the choice, as well as all attributes in the superclass. Consequently, when a role / attribute is added, it becomes automatically part of the choice (as opposed to the XOR invariant pattern where a new role / attribute does not automatically become part of the choice). When a role / attribute is removed, it is automatically removed from the choice.



Instance:

```
<G xmlns="urn:swift:xsd:$G" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <att1>data1</att1>
</G>
```

or

```
<G xmlns="urn:swift:xsd:$G" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <r1>
      <Name>data2</Name>
   </r1>
</G>
```

or

```
<G xmlns="urn:swift:xsd:$G" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <r2>
      <Address>data3</Address>
   </r2>
</G>
```

Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema version 2.2 - Generated by SWIFTStandards Workstation
(build:R2.2.0.10) on Sep 07 16:55:10-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns="urn:swift:xsd:$G"
targetNamespace="urn:swift:xsd:$G">

<xs:element name="Document" type="Document"/>

<xs:complexType name="Document">
 <xs:sequence>
  <xs:element name="G" type="G"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="G">
 <xs:sequence>
  <xs:choice>
   <xs:element name="att1" type="Max35_Text"/>
   <xs:element name="r1" type="H1"/>
   <xs:element name="r2" type="H2"/>
  </xs:choice>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H1">
 <xs:sequence>
  <xs:element name="Name" type="Max35_Text"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="H2">
 <xs:sequence>
  <xs:element name="Address" type="Max35_Text"/
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="Max35_Text">
 <xs:restriction base="xs:string">
  <xs:length value="35"/>
 </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Note: the aggregation of a <<choice>> may not have a multiplicity. However the members of a <<choice>> are allowed to have one. These multiplicities are treated as follows:

UML notation	UML notation	Schema notation	means
r1 0..1	r2 0..1	minOccurs="0" maxOccurs="1"	r1 or r2 may be present, but not both. This means both may be absent as well.
r1 0..n	r2 0..n	minOccurs="0" maxOccurs="unbounded"	r1 or r2 may be present up to n times, but not both. This means both may be absent as well.
r1 1	r2 1	-	r1 or r2 must be present, but not both (= XOR).
r1 1..n	r2 1..n	minOccurs="1" maxOccurs="unbounded"	r1 or r2 must be present up to n times, but not both (= XOR).
r1 0..n	r2 1..n	A choice between <xsd:element name= « r1 » with minOccurs="0" maxOccurs="unbounded" and <xsd:element name= « r2» minOccurs="1" maxOccurs="unbounded"	r1 may be present up to n times or r2 must be present up to n times, but not both (= XOR).

1031
1032

3 Schema design rules

3.1 Common design rules and usage

- Should only be used to validate the message (though this validation is limited if we compare with pure software validation)
- Should not replace the UML model.

3.2 Schema Design rules

3.2.1 XML name clash support within the scope of a message

3.2.1.1 General behaviour of SWIFTStandards XML attributes

The schema will be generated only for validation purposes.

3.2.1.2 Case 1: 2 UML role names are the same and have the same content model

This is not an issue, as those role names will be defined in two different complexTypes.

3.2.1.3 Case 2: 2 UML role or 2 attribute names are the same, and they have a different content model

This is not an issue for schemas as long as the roles or attributes belong to different classes.

3.2.1.4 Case 3: 2 UML attribute names are the same, and their respective UML types are the same.

Same as 3.2.1.2

3.2.1.5 Case 4: A UML role name and a UML attribute name are the same

This is not an issue for schemas as long as the role and attribute belong to different classes.

3.2.1.6 Case 5: Two classes in different packages have the same name

As the name of the class will be used for naming the associated complexType in the schema, this is NOT allowed.

3.2.2 XML schema features used in SWIFTStandards XML

3.2.2.1 Namespaces in XML schema and XML instances

SWIFTStandards XML schema and XML instances use four name spaces:

- the default (non qualified) namespace. All schema have their own default namespace generated according to the following regular expression: “urn:swift:xsd:$+”. Where the “+” must be replaced by the message name possibly prefixed by the collaboration name separated by a ‘.’.
- xs: W3C XML schema namespace (not used in instances)
- xsi: W3C XML schema-instance namespace
- a target namespace (for schema only) which is the same as the default namespace.

Schema:

```
<schema
    xmlns=”urn:swift:xsd:$NoticeOfExecution”
    xmlns:xs=” http://www.w3.org/2001/XMLSchema”
    targetNamespace=”urn:swift:xsd:$NoticeOfExecution”>
```

Instance:

```
<NoticeOfExecution
    xmlns=”urn:swift:xsd:$NoticeOfExecution”
    xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
```

3.2.2.2 Schema location in the XML instance

The rootelement of the business payload carries the location (as an Universal Resource Information) of the XML Schema, in the form of the XML attribute xsi:SchemaLocation.

It is not allowed to have xsi:SchemaLocation to appear in any element (much like xmlns) but only in the rootelelement of the business payload.

Instance:

```
<NoticeOfExecution xsi:schemaLocation=”file://file_path”>
```

3.2.2.3 XML facets on simple types

The following sections describe the facets that will be introduced in the XML schema.

Those XML schema facets are based on the UML invariants which have been specified in the UML model according to the section 3.4. Summary of UML invariants related to schema production.

3.2.2.3.1 pattern

Pattern matching allows lexical validation on strings, which syntax can be described using regular expressions, (commonly referred to as "Perl expressions").

This facet only applies to strings.

The exact syntax of the allowed regular expressions is defined in appendix E of "XML Schema Part 2: Datatypes" (XML Schema's W3C Recommendation May 2001).

For instance:

```
<xs:simpleType name='BIC'>
  <xs:restriction base='string'>
    <xs:pattern value='[a-zA-Z]{4,4}[a-zA-Z]{2,2}[a-zA-Z0-9]{2,2}[a-zA-Z0-9]{0,3}'/>
  </xs:restriction>
</xs:simpleType>
```

3.2.2.3.2 length, minLength, maxLength

XML schema allows restriction of the value space of any string value (i.e.: double, integer, date etc are not affected) by using the following constraining facets:

- length
- minLength
- maxLength

Those facets only apply on strings, and their values must be positive integer values.

For instance, a BankAddress is a string of 10 characters minimum and 40 characters maximum:

```
<xs:simpleType name='BankAddress'>
  <xs:restriction base='string'>
    <xs:minLength value='10'/>
    <xs:maxLength value='40'/>
  </xs:restriction>
</xs:simpleType>
```

3.2.2.3.3 minInclusive, maxInclusive, minExclusive, maxExclusive

XML schema allows restriction of the value space of any numerical value by using the following constraining facets:

- minInclusive
- minExclusive
- maxInclusive
- maxExclusive

Those facets only apply to numerical values (Integer, Long, BigDecimal, Float, Double) and to time measurement related values (Date, Time,…) and their value must be constants of the same type than the numeric value they apply to.

For instance, the financial instrument below must contain between 1 and 100000 securities:

```
<xs:complexType name='SecuritiesInstrument'>
  <xs:sequence>
    <xs:element  name='ISIN' type='string'/>
    <xs:element  name='Quantity' >
      <xs:simpleType>
        <xs:restriction base='xs:decimal'>
          <xs:minInclusive value='1'/>
          <xs:maxInclusive value='100000'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

3.2.2.3.4 enumeration

XML schema allows restriction of the value space of an enumeration by using the enumeration constraining facet.

This facet only applies to enumerations, and their value must be part of the original enumeration from which they restrict.

For instance, a class M containing an attribute b of type E1 with an XML Invariant restricting the enumerated value to Value2:

```
<xs:complexType name="M">
    <xs:sequence>
        <xs:element name = "b">
            <xs:simpleType>
                <xs:restriction base="E1">
```

```
                    <xs:enumeration value ="Value2"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name = "E1">
    <xs:restriction base = "xs:string">
        <xs:enumeration value = "Value1"/>
        <xs:enumeration value = "Value2"/>
    </xs:restriction>
</xs:simpleType>
```

3.2.2.3.5 totalDigits, fractionDigits

Fixed point decimal values need a totalDigits specification (i.e. the maximum number of decimal digits in values of datatypes derived from decimal: totalDigits), as well as a fractionDigits specification (i.e. the maximum number of decimal digits in the fractional part of values of datatypes derived from decimal: fractionDigits).

The value of the totalDigits facet must be a positive integer.

The value of the fractionDigits facet must be a non-negative integer.

For instance, requiring a totalDigits of 8 digits with 2 digits after the decimal point on an amount would translate to the following instance:

```
<xs:simpleType name='Amount'>
    <xs:restriction base='xs:decimal'>
      <xs:totalDigits value='8'/>
      <xs:fractionDigits value='2'/>
    </xs:restriction>
</xs:simpleType>
```

3.2.2.4 Nillable

To be used in conjunction with the XML-nil attribute. The Schema attribute nillable specifies whether the instance can carry a nil value. Default value is false.

In the following schema:

```
<xs:complexType name=' FinancialInstrument'>
  <xs:sequence>
    <xs:element  name='ISIN' type='string'/>
    <xs:element  name='Quantity' type='xs:decimal' nillable='true'/>
  </xs:sequence>
</xs:complexType >
```

Only the Quantity can carry a nil value.

It should be noted that nillable is not a facet, but an attribute (as abstract, minOccurs, maxOccurs, …). This implies that, in the schema's context, nillable applies to an element (and not a type).

Therefore the nillable option should consequently not be encoded as an invariant on a class in the UML model. It will thus be set either at the attribute or role level (in which case the corresponding element in the schema would be nillable).

In the XML instance document, the XML attribute **nil** can be used to indicate that an element has no value.

Assuming the following schema:

```
<xs:complexType name='OrderOfBuy'>
  <xs:element  Securities type='FinancialInstrument'/>
</xs:complexType >

<xs:complexType name=' FinancialInstrument'>
  <xs:element  name='ISIN' type='string'/>
  <xs:element  name='Quantity' type='xs:decimal' nillable='true'/>
</xs:complexType >
```

An order-of-buy XML instance with no quantity of securities specified (as opposed to a value of zero) will be expressed as:

```
<OrderOfSell>
  <Securities>
    <ISIN>BE1234567890 </ISIN>
    <Quantity xsi:nil='true'/>
  <Securities>
</ OrderOfSell >
```

Note that an alternative to not using the 'nil' XML-attribute is to omit the nill element. By doing so we introduce an ambiguity between **not** specifying an optional element and specifying an optional element which value is **nil**.

3.3 Granularity of Schemas

There is one Schema per message.

3.4 Summary of UML invariants related to schema production

Those invariants will be defined as user properties on methods having the <<inv>> stereotypes, on the tab called XML Invariants.

XML facet	Applies on UML type	Value of type	Schema example
pattern	String	Defined in Appendix E of "XML Schema Part 2: Datatypes"	<xs:simpleType name='BIC'> <xs:restriction base='string'> <xs:pattern value=' [a-z]{2,4}'/> </xs:restriction> </xs:simpleType>
length	String	Non-negative integer	<xs:simpleType name='BIC'> <xs:restriction base='string'> <xs:length value='12'/> </xs:restriction> </xs:simpleType>
minLength	String	Non-negative integer	<xs:simpleType name='BIC'> <xs:restriction base='string'> <xs:minLength value='8'/> </xs:restriction> </xs:simpleType>
maxLength	String	Non-negative integer	<xs:simpleType name='BIC'> <xs:restriction base='string'> <xs:maxLength value='12'/> </xs:restriction> </xs:simpleType>
totalDigits	Integer, Long, Float, Double, BigDecimal	Positive integer	<xs:simpleType name='BEF'> <xs:restriction base='xs:decimal'> <xs:totalDigits value='3'/> </xs:restriction> </xs:simpleType>
fractionDigits	Float, Double, BigDecimal	Non-negative integer	<xs:simpleType name='USD'> <xs:restriction base='xs:decimal'> <xs:fractionDigits value='2'/> </xs:restriction> </xs:simpleType>

minInclusive	Integer, Long, Float, Double, BigDecimal	Constant of the same type as the UML type	<xs:simpleType name='Salary'> <xs:restriction base='xs:decimal'> <xs:minInclusive value='40000'/> </xs:restriction> </xs:simpleType>
minExclusive	Integer, Long, Float, Double, BigDecimal	Constant of the same type as the UML type	<xs:simpleType name='Salary'> <xs:restriction base='xs :decimal'> <xs:minExclusive value='40000'/> </xs:restriction> </xs:simpleType>
maxInclusive	Integer, Long, Float, Double, BigDecimal	Constant of the same type as the UML type	<xs:simpleType name='Taxes'> <xs:restriction base='xs:decimal'> <xs:maxInclusive value='90000'/> </xs:restriction> </xs:simpleType>
maxExclusive	Integer, Long, Float, Double, BigDecimal	Constant of the same type as the UML type	<xs:simpleType name='Taxes'> <xs:restriction base='xs:decimal'> <xs:maxExclusive value='90000'/> </xs:restriction> </xs:simpleType>

4 Naming Conventions and Taxonomy

See Naming Conventions appendix

5 Character set

SWIFTStandards XML uses UTF-8 as the (default) character encoding mechanism, for the following reasons:

- It has the most efficient method of character representation:
  - It is the shortest method to represent the characters which are currently the most commonly used in a financial environment (ASCII and EBCDIC characters)
  - It can still represent almost any known character

- It is interoperable with many other encoding schemes through (automatable) conversion algorithms.

Example:

<?xml version=”1.0” encoding=”UTF-8”?>

6 Appendices

A Naming conventions and taxonomy scheme

A.1 Introduction

The purpose of this appendix is to explain the methodology to be used when generating meaningful SWIFTStandards XML tags. It provides the general principles of classification of SWIFTStandards XML tags.

It is very important to have a structure in place to 'control' the way data elements are tagged.

- It will allow us to keep SWIFTStandards XML as condensed as possible
- It will limit a proliferation of different usages by different developers
- It provides a way to easily trace and manage a SWIFTStandards XML (and UML) repository which is based on meaningful tags.

A.2 Constraints / Assumptions

- To use normalised names: to abide to the tagging constraints imposed by the W3C XML specification v1.0., C++ and Java
- To have one namespace that will contain all business elements covered by S.W.I.F.T.
- To have no elements that can be expressed in SWIFTStandards XML and cannot be expressed in UML and vice-versa.
- To have business information which is expressed as SWIFTStandards XML elements/values and meta data information which is expressed as SWIFTStandards XML attributes.
- To have Schema's that support inheritance. Consequently, attributes and aggregates can be reused or overridden.

A.3 Naming rules

As already stated, the SWIFTStandards XML name is the XML name assigned to the UML element or by default the UML name. Hence, all below rules apply for XML names, SWIFTStandards XML names and UML names of elements without XML names.

A.3.1 General rules

Use the English vocabulary.

Abide to the (character) restrictions described in SWIFTStandards XML for naming elements:

- All names must start with an alphabetic character.
- All characters following the first characters must be alphabetic characters, numeric characters, or '_'.

Apply camel case convention:

- Names for elements and attributes may be made up of multiple words each consisting of alphanumeric characters.
- Each word starts with a capital letter.
- All white spaces between words are removed.

End of document