



1
2
3
4
5
6
7
8
9
10

SWIFTStandards XML design rules version 2.3

Technical Specification

10 **Table of contents**

11 **1 Introduction3**

12 **2 Mapping rules from UML to SWIFTStandards XML.....3**

13 2.1 General mapping rules 3

14 2.2 SWIFTStandards XML elements 4

15 2.2.1 SWIFTStandards XMLTag..... 4

16 2.2.2 xsi:type 4

17 2.2.3 Representation Class Attribute..... 5

18 2.3 Specific mapping rules 6

19 2.3.1 Data types 6

20 2.3.2 Class 24

21 2.3.3 Simple composition..... 25

22 2.3.4 Class attributes 26

23 2.3.5 Composition of vectorial attributes (Collections)..... 28

24 2.3.6 Inheritance..... 31

25 Enumerated roles using XOR invariant 33

26 Enumerated attributes using XOR invariant 36

27 2.3.9 Enumerated roles and attributes using <<choice>> stereotype 37

28 **3 Schema design rules41**

29 3.1 Common design rules and usage 41

30 3.2 Schema Design rules 41

31 3.2.1 XML name clash support within the scope of a message 41

32 3.2.2 XML schema features used in SWIFTStandards XML 42

33 3.3 Granularity of Schemas 46

34 3.4 Summary of UML invariants related to schema production 47

35 **4 Naming Conventions and Taxonomy48**

36 **5 Character set48**

37 **6 Appendices50**

38 **End of document52**

39
40

40 1 Introduction

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

45 The use of XML is part of the overall approach for the development of SWIFTStandards.
46 This development focuses on the correct definition of a business standard using modelling
47 techniques¹. The resulting business standard is captured in UML (Unified Modelling
48 Language²) and is stored in an electronic repository, the “SWIFTStandards Repository”³.
49 Business messages are defined in UML class diagrams and XML is then used as a physical
50 representation (i.e. the syntax) of the defined business messages. A set of **XML design**
51 **rules**, called **SWIFTStandards XML**, define in a very detailed and strict way how this
52 physical XML representation is derived from the business message in the UML class
53 diagram.

54 This document explains these XML design rules.

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

57 2 Mapping rules from UML to SWIFTStandards XML

58 2.1 General mapping rules

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

- 61 • SWIFTStandards XML representation to be as structured as possible:
 - 62 – Business information is expressed as XML elements/values;
 - 63 – Metadata information is expressed as XML attributes. XML attributes are not to be
64 conveyed ‘on the wire’ in the XML instance, unless required to remove ambiguity.
- 65 • The current work is based on W3C’s Recommendation of May, 2001.
- 66 • The names used in SWIFTStandards XML are the XML names or, when absent, the
67 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.

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

77 **2.2 SWIFTStandards XML elements**

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

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

82

83 **2.2.1 SWIFTStandards XMLTag**

84 SWIFTStandards XMLTag is assigned according to following rules:

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

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

88 For a SWIFTStandards XML element derived from a role:

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

92 For a SWIFTStandards XML element derived from an attribute:

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

94 **2.2.2 xsi:type**

95 **2.2.2.1 In the schema**

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

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

98 **2.2.2.2 In the corresponding instance**

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

101 *summarizing:*

SWIFTStandards XML element derived from	Type
Class	Class name
Role	Name of the class at the end of the aggregation
Attribute	<ul style="list-style-type: none"> Name of the class of the attributes' type attribute type name (for primitive types)

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

103

104 **2.2.3 Representation Class Attribute**

105

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

109

110 **2.3 Specific mapping rules**

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

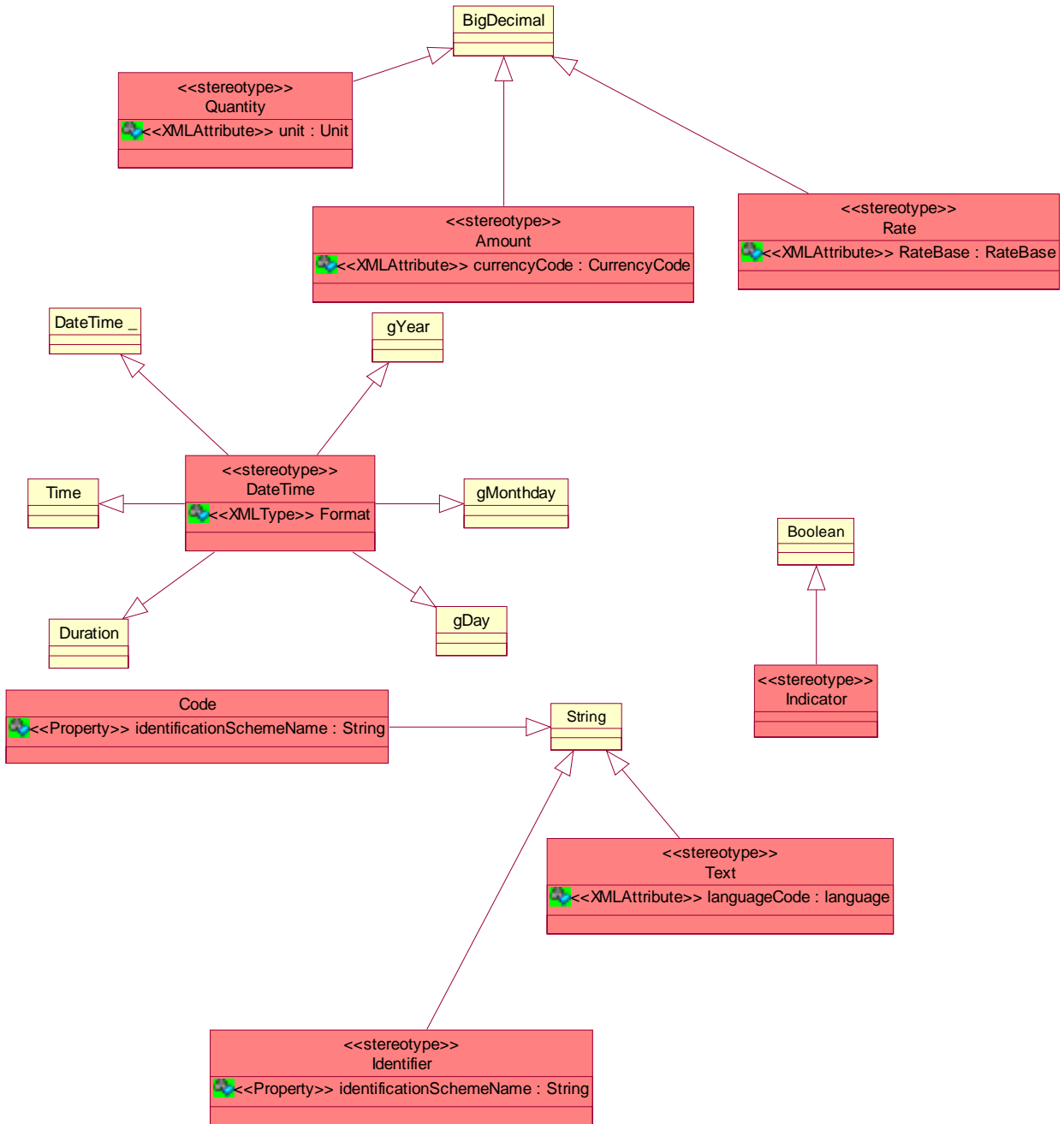
115 **2.3.1 Data types**

116 In a message model, all class attributes have a type, which we call **data types** for the
117 purpose of this chapter. Data types define the structure of a class attribute.
118

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.

132

133

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:

138

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.

139

140 **2.3.1.3 User-defined data types**

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

148 In order to apply facets, the XML types that are generated for those data types must be
 149 simpleTypes or complexTypes with simpleContent, and not complexTypes⁵.

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

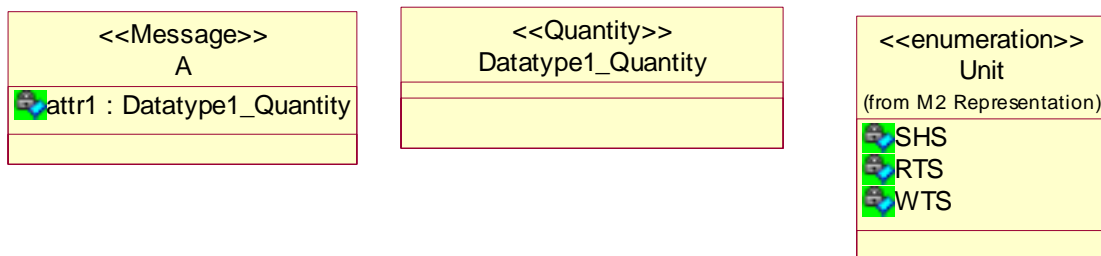
152 Where necessary (in case of ambiguity), the representation class attribute maps to an XML
 153 attribute.

154

155

156 **2.3.1.3.1 Data type using representation class <<Quantity>>**

157



158

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

159 Properties:

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

169

170

171 Instance:

```
172 <A>
173   <attr1 unit="SHS">1000</attr1>
174 </A>
```

175

176 Schema:

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

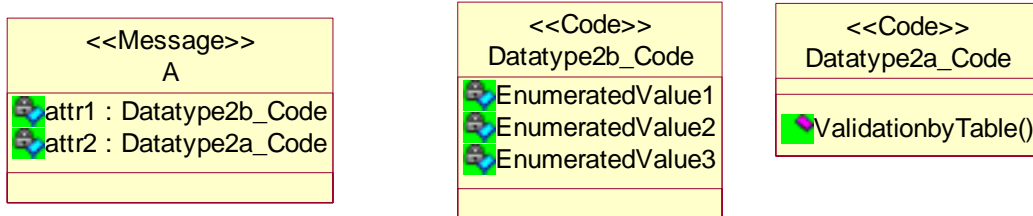
<!-- class: A -->
<xs:complexType name="A">
  <xs:sequence>
    <xs:element name="attr1" type="xs:Datatype1_Quantity" />
  </xs:sequence>
</xs:complexType>

<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>
```

177

177

178 **2.3.1.3.2 Data type using representation class <<Code>>**

179

180

181 Properties:

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

189

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

190

191 Instance:

```

192 <A>
193   <attr1>EnumeratedValue2</attr1>
194   <attr2>AnythingGoesHere</attr2>
195 </A>
  
```

196

197 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>

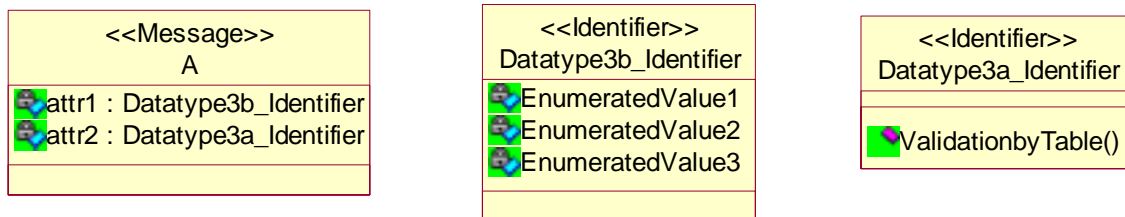
```

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

212

213 Instance:

214

215

216

217

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

218

219 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>
```

220

221

222

223

224

225

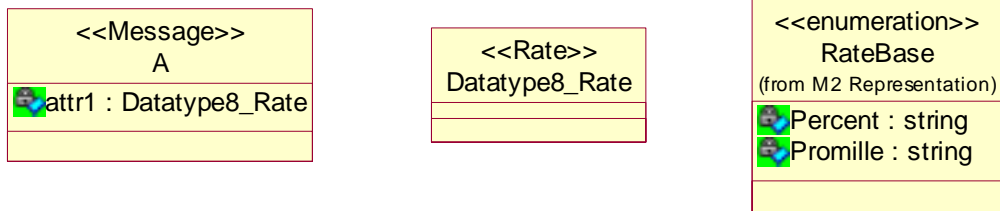
226

226

227

228 **2.3.1.3.4 Data type using representation class <<Rate>>**

229



230

231 Properties:

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

239

240

241 Instance:

```

242 <A>
243   <attr1 ratebase="Percent">95.6</attr1>
244 </A>
  
```

245

246 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>

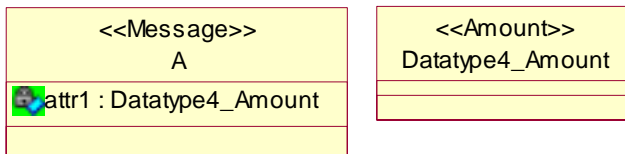
```

247

248

249 **2.3.1.3.5 Data type using representation class <<Amount>>**

250



251

252 **Properties:**

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

261

262

263 Instance:

```
264 <A>
265   <attr1 currencyCode="USD">95.6</attr1>
266 </A>
```

267

268 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>

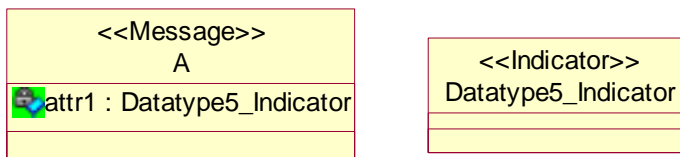
<xs:simpleType name="CurrencyCode">
  <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>
```

269

270

271 **2.3.1.3.6 Data type using representation class <<Indicator>>**

272



273

274 Properties:

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

277

278

279 Instance:

```
280 <A>
281   <attr1>>true</attr1>
282 </A>
```

283

284 Schema:

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

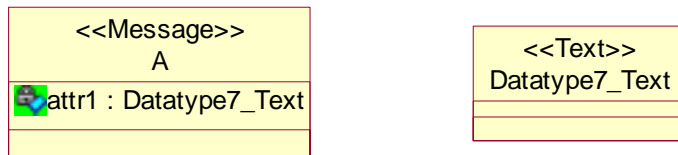
<!-- class: A -->
<xs:complexType name="A">
  <xs:sequence>
    <xs:element name="attr1" type="xs:Datatype5_Indicator"/>
  </xs:sequence>
</xs:complexType>

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

285

286 **2.3.1.3.7 Data type using representation class <<Text>>**

287



288

289 Instance:

```
290 <A>
291   <attr1>any narrative text</attr1>
292 </A>
```

293

294 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>

```

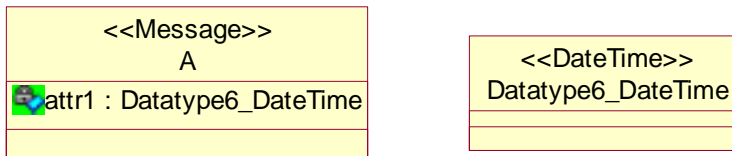
295

296

297 **2.3.1.3.8 Data type using representation class <<DateTime>>**

298

299



300

301

302 Properties:

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

308

309

310 Instance:

```

311 <A>
312   <attr1>2002-11-23</attr1>
313 </A>

```

314

315 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>

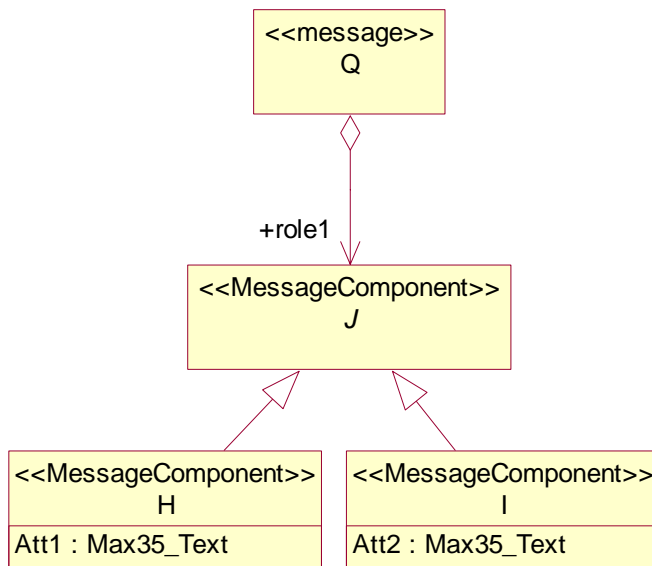
<xs:simpleType name="Datatype6_DateTime">
  <xs:restriction base="xs:dateTime">
    <xs:minInclusive value="2002-01-01T00:00:00" />
  </xs:restriction>
</xs:simpleType>

```

316

317 **2.3.1.4 Enumerated types**318 **2.3.1.4.1 Basic pattern**

- 319
- In the example below, two different types can play role1: either Att1 or Att2.
 - 320
 - 321
- 322



323

324

325 Instance:

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

332 or

```
333 <Q xmlns="urn:swift:xsd:$Q" xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
334 instance">  
335   <role1 xsi:type="I">  
336     <Att2>data2</Att2>  
337   </role1>  
338 </Q>  
339
```

340

341 Schema:

```
342 <?xml version="1.0" encoding="UTF-8"?>  
343 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation  
344 (build:R2.2.0.10) on Sep 07 15:58:10-->  
345 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
346 elementFormDefault="qualified" xmlns="urn:swift:xsd:$Q"  
347 targetNamespace="urn:swift:xsd:$Q">  
348   <xs:element name="Document" type="Document"/>  
349  
350   <xs:complexType name="Document">  
351     <xs:sequence>  
352       <xs:element name="Q" type="Q"/>  
353     </xs:sequence>  
354   </xs:complexType>  
355  
356   <xs:complexType name="Q">  
357     <xs:sequence>  
358       <xs:element name="role1" type="J"/>  
359     </xs:sequence>  
360   </xs:complexType>  
361  
362   <xs:complexType name="H">  
363     <xs:complexContent>  
364       <xs:extension base="J">  
365         <xs:sequence>  
366           <xs:element name="Att1" type="Max35_Text"/>  
367         </xs:sequence>  
368       </xs:extension>  
369     </xs:complexContent>  
370   </xs:complexType>  
371  
372   <xs:complexType name="J" abstract="true"/>  
373  
374   <xs:complexType name="I">  
375     <xs:complexContent>
```

```

376 <xs:extension base="J">
377   <xs:sequence>
378     <xs:element name="Att2" type="Max35_Text" />
379   </xs:sequence>
380 </xs:extension>
381 </xs:complexContent>
382 </xs:complexType>
383
384 <xs:simpleType name="Max35_Text">
385   <xs:restriction base="xs:string">
386     <xs:length value="35" />
387   </xs:restriction>
388 </xs:simpleType>
389
390 </xs:schema>
391

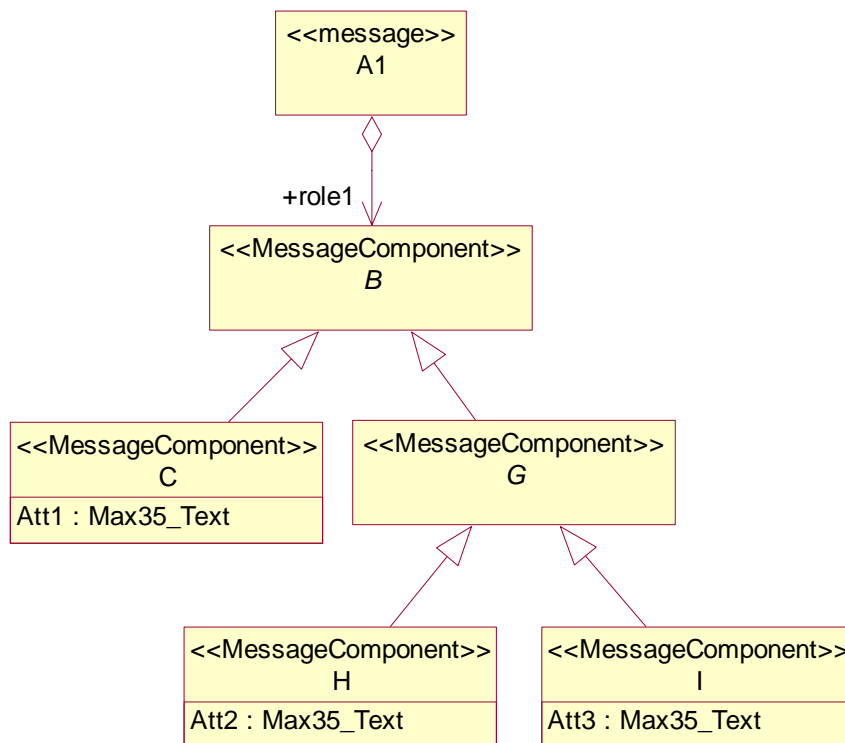
```

392

393

394 **2.3.1.4.2 Re-use pattern**

395



396

397

398

399

400 Instance:

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

407 or

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

414 or

```
415 <A1 xmlns="urn:swift:xsd:$A1 "  
416 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
417   <role1 xsi:type="I">  
418     <Att3>data3</Att3>  
419   </role1>  
420 </A1>  
421
```

422

423 Schema:

```
424 <?xml version="1.0" encoding="UTF-8"?>  
425 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation  
426 (build:R2.2.0.10) on Sep 07 15:58:10-->  
427 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema "  
428 elementFormDefault="qualified" xmlns="urn:swift:xsd:$A1 "  
429 targetNamespace="urn:swift:xsd:$A1 ">  
430 <xs:element name="Document" type="Document" />  
431  
432 <xs:complexType name="Document">  
433   <xs:sequence>  
434     <xs:element name="A1" type="A1" />  
435   </xs:sequence>  
436 </xs:complexType>  
437  
438 <xs:complexType name="A1">  
439   <xs:sequence>  
440     <xs:element name="role1" type="B" />  
441   </xs:sequence>  
442 </xs:complexType>  
443  
444 <xs:complexType name="G" abstract="true">  
445   <xs:complexContent>  
446     <xs:extension base="B" />  
447   </xs:complexContent>
```

```
448 </xs:complexType>
449
450 <xs:complexType name="B" abstract="true"/>
451
452 <xs:complexType name="C">
453   <xs:complexContent>
454     <xs:extension base="B">
455       <xs:sequence>
456         <xs:element name="Att1" type="Max35_Text"/>
457       </xs:sequence>
458     </xs:extension>
459   </xs:complexContent>
460 </xs:complexType>
461
462 <xs:complexType name="I">
463   <xs:complexContent>
464     <xs:extension base="G">
465       <xs:sequence>
466         <xs:element name="Att3" type="Max35_Text"/>
467       </xs:sequence>
468     </xs:extension>
469   </xs:complexContent>
470 </xs:complexType>
471
472 <xs:complexType name="H">
473   <xs:complexContent>
474     <xs:extension base="G">
475       <xs:sequence>
476         <xs:element name="Att2" type="Max35_Text"/>
477       </xs:sequence>
478     </xs:extension>
479   </xs:complexContent>
480 </xs:complexType>
481
482 <xs:simpleType name="Max35_Text">
483   <xs:restriction base="xs:string">
484     <xs:length value="35"/>
485   </xs:restriction>
486 </xs:simpleType>
487
488 </xs:schema>
489
490
```

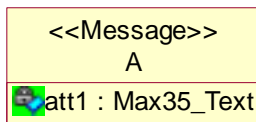
490

491 **2.3.2 Class**

492

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: <ul style="list-style-type: none"> 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

493



494

495 **Instance:**

```

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

500

501 **Schema:**

```

502 <?xml version="1.0" encoding="UTF-8"?>
503 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation
504 (build:R2.2.0.10) on Sep 05 16:21:43-->
505 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
506 elementFormDefault="qualified" xmlns="urn:swift:xsd:$A"
507 targetNamespace="urn:swift:xsd:$A">
508   <xs:element name="Document" type="Document"/>
509
510   <xs:complexType name="Document">
511     <xs:sequence>
512       <xs:element name="A" type="A"/>
513     </xs:sequence>
514   </xs:complexType>
515
516   <xs:complexType name="A">
517     <xs:sequence>
518       <xs:element name="att1" type="Max35_Text"/>
  
```



```

519     </xs:sequence>
520 </xs:complexType>
521
522     <xs:simpleType name="Max35_Text">
523       <xs:restriction base="xs:string">
524         <xs:length value="35"/>
525       </xs:restriction>
526     </xs:simpleType>
527
528 </xs:schema>

```

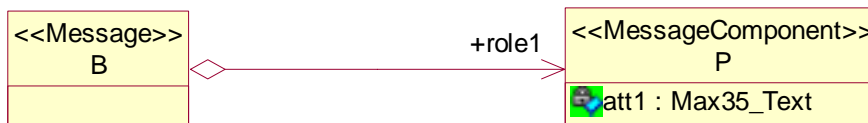
529 2.3.3 Simple composition

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

535

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

536



537 Instance:

```

538 <B xmlns="urn:swift:xsd:$B" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
539 instance">
540   <role1>
541     <att1>data</att1>
542   </role1>
543 </B>

```

544

545 Schema:

```

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

```

```

552 <xs:element name="Document" type="Document" />
553
554
555 <xs:complexType name="Document">
556   <xs:sequence>
557     <xs:element name="B" type="B" />
558   </xs:sequence>
559 </xs:complexType>
560
561 <xs:complexType name="B">
562   <xs:sequence>
563     <xs:element name="role1" type="P" />
564   </xs:sequence>
565 </xs:complexType>
566
567 <xs:complexType name="P">
568   <xs:sequence>
569     <xs:element name="att1" type="Max35_Text" />
570   </xs:sequence>
571 </xs:complexType>
572
573 <xs:simpleType name="Max35_Text">
574   <xs:restriction base="xs:string">
575     <xs:length value="35" />
576   </xs:restriction>
577 </xs:simpleType>
578
579 </xs:schema>
580

```

581

582 2.3.4 Class attributes

- 583 ▪ A class can also contain attributes;
- 584 ▪ A class attribute is described using a name and a type;
- 585 ▪ By default ,the first SWIFTStandards XML child elements within its parents are the
- 586 attributes, followed by the roles. However, you can define [the sequence of all the](#)
- 587 [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.

588



589

590 Instance:

```

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

598

599 Schema:

```

600 <?xml version="1.0" encoding="UTF-8"?>
601 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation
602 (build:R2.2.0.10) on Sep 05 16:21:43-->
603 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
604 elementFormDefault="qualified" xmlns="urn:swift:xsd:$D"
605 targetNamespace="urn:swift:xsd:$D">
606
607 <xs:element name="Document" type="Document"/>
608
609 <xs:complexType name="Document">
610   <xs:sequence>
611     <xs:element name="D" type="D"/>
612   </xs:sequence>
613 </xs:complexType>
614
615 <xs:complexType name="D">
616   <xs:sequence>
617     <xs:element name="att1" type="TrueFalse_Indicator"/>
618     <xs:element name="role1" type="E"/>
619   </xs:sequence>
620 </xs:complexType>
621
622 <xs:complexType name="E">
623   <xs:sequence>
624     <xs:element name="att2" type="Max35_Text"/>
625   </xs:sequence>
626 </xs:complexType>
627
628 <xs:simpleType name="Max35_Text">
629   <xs:restriction base="xs:string">
630     <xs:length value="35"/>
631   </xs:restriction>
632 </xs:simpleType>
633
634 <xs:simpleType name="TrueFalse_Indicator">
635   <xs:restriction base="xs:boolean"/>
  
```

```

636 </xs:simpleType>
637
638 </xs:schema>
639

```

640

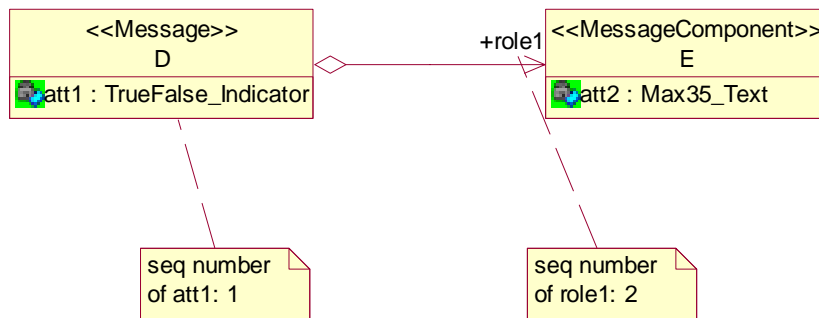
641

642 2.3.4.1 Element order

643

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

647



648

649

650 2.3.5 Composition of vectorial attributes (Collections)

- 651 • The cardinality expresses the number of occurrences of elements. The default value is
 652 1, in which case it can be omitted; else it is represented as a **range** e.g **0..***.
- 653 • Use a range-cardinality to express a collection of elements, which can be represented
 654 either as a collection of attributes or roles. In the example below, C contains a
 655 collection of A's expressed as attributes (att3) and a collection of Bs expressed as roles
 656 (role1).
- 657 • Schemas can validate exactly the cardinality.

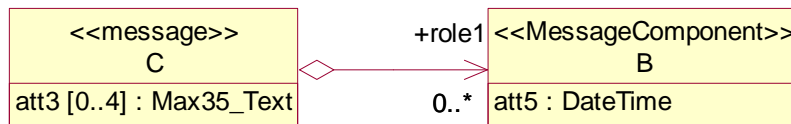
658

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"

659

660



661

662 Instance :

```

663 <C xmlns="urn:swift:xsd:$C" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
664 instance">
665   <att3>data1a</att3>
666   <att3>data1b</att3>
667   <role1>
668     <att5>2001-01-01</att5>
669   </role1>
670 </C>
  
```

671

672 Schema:

```

673 <?xml version="1.0" encoding="UTF-8"?>
674 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation
675 (build:R2.2.0.10) on Sep 07 13:40:40-->
676 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
677 elementFormDefault="qualified" xmlns="urn:swift:xsd:$C"
678 targetNamespace="urn:swift:xsd:$C">
679
680 <xs:element name="Document" type="Document"/>
681
682 <xs:complexType name="Document">
683   <xs:sequence>
684     <xs:element name="C" type="C"/>
685   </xs:sequence>
686 </xs:complexType>
687
688 <xs:complexType name="C">
689   <xs:sequence>
690     <xs:element name="att3" type="Max35_Text" minOccurs="0" maxOccurs="4"/>
691     <xs:element name="role1" type="B" minOccurs="0" maxOccurs="unbounded"/>
692   </xs:sequence>
  
```

```
693 </xs:complexType>
694
695 <xs:complexType name="B">
696   <xs:sequence>
697     <xs:element name="att5" type="xs:dateTime"/>
698   </xs:sequence>
699 </xs:complexType>
700
701 <xs:simpleType name="Max35_Text">
702   <xs:restriction base="xs:string">
703     <xs:length value="35"/>
704   </xs:restriction>
705 </xs:simpleType>
706
707 </xs:schema>
708
```

709
710
711
712

712

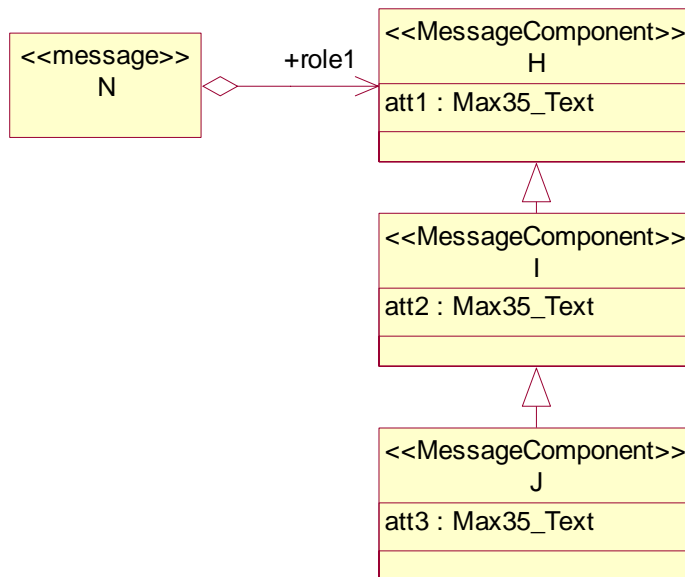
713 **2.3.6 Inheritance**

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

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

722

723



724

725 **Instance:**

```

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

732 **or**

```

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

```
735 <role1 xsi:type="I">  
736 <att1>data1</att1>  
737 <att2>data2</att2>  
738 </role1>  
739 </N>
```

740 or

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

749

750 Schema:

```
751 <?xml version="1.0" encoding="UTF-8"?>  
752 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation  
753 (build:R2.2.0.10) on Sep 07 13:40:40-->  
754 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
755 elementFormDefault="qualified" xmlns="urn:swift:xsd:$N"  
756 targetNamespace="urn:swift:xsd:$N">  
757  
758 <xs:element name="Document" type="Document"/>  
759  
760 <xs:complexType name="Document">  
761 <xs:sequence>  
762 <xs:element name="N" type="N"/>  
763 </xs:sequence>  
764 </xs:complexType>  
765  
766 <xs:complexType name="N">  
767 <xs:sequence>  
768 <xs:element name="role1" type="H"/>  
769 </xs:sequence>  
770 </xs:complexType>  
771  
772 <xs:complexType name="I">  
773 <xs:complexContent>  
774 <xs:extension base="H">  
775 <xs:sequence>  
776 <xs:element name="att2" type="Max35_Text"/>  
777 </xs:sequence>  
778 </xs:extension>  
779 </xs:complexContent>  
780 </xs:complexType>  
781  
782 <xs:complexType name="H">  
783 <xs:sequence>  
784 <xs:element name="att1" type="Max35_Text"/>  
785 </xs:sequence>  
786 </xs:complexType>  
787
```



```

788 <xs:complexType name="J">
789   <xs:complexContent>
790     <xs:extension base="I">
791       <xs:sequence>
792         <xs:element name="att3" type="Max35_Text"/>
793       </xs:sequence>
794     </xs:extension>
795   </xs:complexContent>
796 </xs:complexType>
797
798 <xs:simpleType name="Max35_Text">
799   <xs:restriction base="xs:string">
800     <xs:length value="35"/>
801   </xs:restriction>
802 </xs:simpleType>
803
804 </xs:schema>

```

805

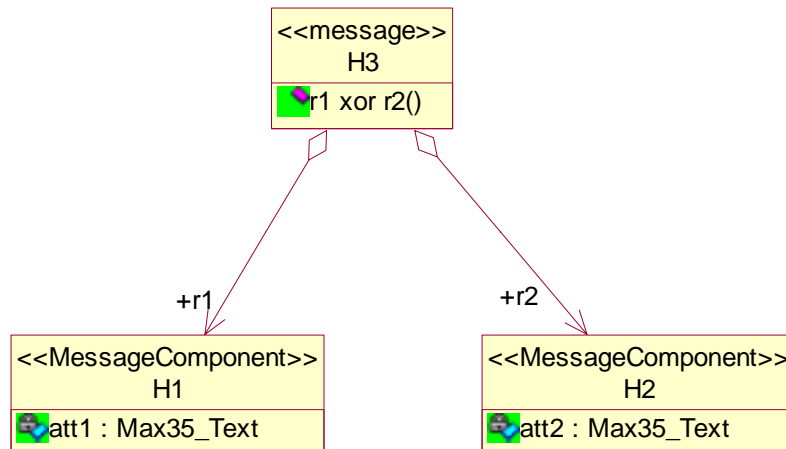
806 Notes:

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

811

812 2.3.7 Enumerated roles using XOR invariant

813



814

815 Instance:

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

822 or

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

829

830 Schema:

```
831 <?xml version="1.0" encoding="UTF-8"?>  
832 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation  
833 (build:R2.2.0.10) on Sep 07 16:55:10-->  
834 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
835 elementFormDefault="qualified" xmlns="urn:swift:xsd:$H3"  
836 targetNamespace="urn:swift:xsd:$H3">  
837  
838 <xs:element name="Document" type="Document"/>  
839  
840 <xs:complexType name="Document">  
841 <xs:sequence>  
842 <xs:element name="H3" type="H3"/>  
843 </xs:sequence>  
844 </xs:complexType>  
845  
846 <xs:complexType name="H3">  
847 <xs:sequence>  
848 <xs:choice>  
849 <xs:element name="r1" type="H1"/>  
850 <xs:element name="r2" type="H2"/>  
851 </xs:choice>  
852 </xs:sequence>  
853 </xs:complexType>  
854  
855 <xs:complexType name="H2">  
856 <xs:sequence>  
857 <xs:element name="att2" type="Max35_Text"/>  
858 </xs:sequence>  
859 </xs:complexType>  
860  
861 <xs:complexType name="H1">  
862 <xs:sequence>  
863 <xs:element name="att1" type="Max35_Text"/>  
864 </xs:sequence>  
865 </xs:complexType>  
866  
867 <xs:simpleType name="Max35_Text">  
868 <xs:restriction base="xs:string">
```

```

869 <xs:length value="35" />
870 </xs:restriction>
871 </xs:simpleType>
872
873 </xs:schema>

```

874

875

876 **Note:** multiplicity for enumerated roles is treated as follows:

877

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).

878

879 **Note:** some rules regarding the XOR in UML:880

- Any XML name may be given to the operation

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

888 2.3.8 Enumerated attributes using XOR invariant

<<Message>> S
att1 : Max35_Text att2 : Max35_Text
<<inv>> att1 xoratt2()

889
890 <S xmlns="urn:swift:xsd:\$S" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
891 instance">
892 <att1>data1</att1>
893 </S>

894 or

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

899

900 Schema:

901 <?xml version="1.0" encoding="UTF-8"?>
902 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation
903 (build:R2.2.0.10) on Sep 07 16:55:10-->
904 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
905 elementFormDefault="qualified" xmlns="urn:swift:xsd:\$S"
906 targetNamespace="urn:swift:xsd:\$S">
907 <xs:element name="Document" type="Document"/>
908 <xs:complexType name="Document">
909 <xs:sequence>
910 <xs:element name="S" type="S"/>
911 </xs:sequence>
912 </xs:complexType>
913 <xs:complexType name="S">
914 <xs:sequence>
915 <xs:element name="S" type="S"/>
916 </xs:sequence>
917 </xs:complexType>

```

918 <xs:choice>
919 <xs:element name="att1" type="Max35_Text" />
920 <xs:element name="att2" type="Max35_Text" />
921 </xs:choice>
922 </xs:sequence>
923 </xs:complexType>
924
925 <xs:simpleType name="Max35_Text">
926 <xs:restriction base="xs:string"
927 <xs:length value="35" />
928 </xs:restriction>
929 </xs:simpleType>
930
931 </xs:schema>

```

932

933

934

935 **Note:** some rules regarding the XOR in UML:

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

945

946 **2.3.9 Enumerated roles and attributes using <<choice>> stereotype**

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

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

953

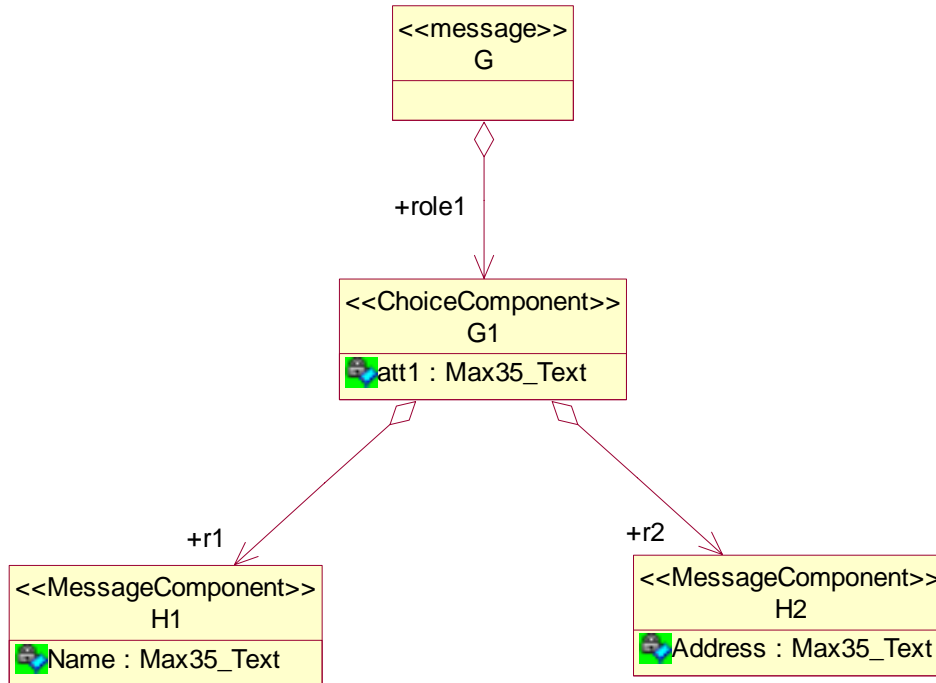
954

955

956

957

958



959

960 Instance:

```

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

965 or

```

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

972 or

```

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

979

980

981 Schema:

```
982 <?xml version="1.0" encoding="UTF-8"?>
983 <!--Schema version 2.2 - Generated by SWIFTStandards Workstation
984 (build:R2.2.0.10) on Sep 07 16:55:10-->
985 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
986 elementFormDefault="qualified" xmlns="urn:swift:xsd:$G"
987 targetNamespace="urn:swift:xsd:$G">
988
989 <xs:element name="Document" type="Document"/>
990
991 <xs:complexType name="Document">
992 <xs:sequence>
993 <xs:element name="G" type="G"/>
994 </xs:sequence>
995 </xs:complexType>
996
997 <xs:complexType name="G">
998 <xs:sequence>
999 <xs:choice>
1000 <xs:element name="att1" type="Max35_Text"/>
1001 <xs:element name="r1" type="H1"/>
1002 <xs:element name="r2" type="H2"/>
1003 </xs:choice>
1004 </xs:sequence>
1005 </xs:complexType>
1006
1007 <xs:complexType name="H1">
1008 <xs:sequence>
1009 <xs:element name="Name" type="Max35_Text"/>
1010 </xs:sequence>
1011 </xs:complexType>
1012
1013 <xs:complexType name="H2">
1014 <xs:sequence>
1015 <xs:element name="Address" type="Max35_Text"/>
1016 </xs:sequence>
1017 </xs:complexType>
1018
1019 <xs:simpleType name="Max35_Text">
1020 <xs:restriction base="xs:string">
1021 <xs:length value="35"/>
1022 </xs:restriction>
1023 </xs:simpleType>
1024
1025 </xs:schema>
```

1026

1027

1028 **Note:** the aggregation of a <<choice>> may not have a multiplicity. However the members
1029 of a <<choice>> are allowed to have one. These multiplicities are treated as follows:

1030

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

1032

1033 **3 Schema design rules**

1034 **3.1 Common design rules and usage**

- 1035 • Should only be used to validate the message (though this validation is limited if we
- 1036 compare with pure software validation)
- 1037 • Should not replace the UML model.
- 1038

1039 **3.2 Schema Design rules**

1040 **3.2.1 XML name clash support within the scope of a message**

1041 **3.2.1.1 General behaviour of SWIFTStandards XML attributes**

1042 The schema will be generated only for validation purposes.

1043 **3.2.1.2 Case 1: 2 UML role names are the same and have the same content** 1044 **model**

1045 This is not an issue, as those role names will be defined in two different complexTypes.

1046 **3.2.1.3 Case 2: 2 UML role or 2 attribute names are the same, and they have a** 1047 **different content model**

1048 This is not an issue for schemas as long as the roles or attributes belong to different classes.

1049 **3.2.1.4 Case 3: 2 UML attribute names are the same, and their respective** 1050 **UML types are the same.**

1051 Same as 3.2.1.2

1052 **3.2.1.5 Case 4: A UML role name and a UML attribute name are the same**

1053 This is not an issue for schemas as long as the role and attribute belong to different classes.

1054 **3.2.1.6 Case 5: Two classes in different packages have the same name**

1055 As the name of the class will be used for naming the associated complexType in the
1056 schema, this is NOT allowed.

1057 **3.2.2 XML schema features used in SWIFTStandards XML**

1058 **3.2.2.1 Namespaces in XML schema and XML instances**

1059 SWIFTStandards XML schema and XML instances use four name spaces:

- 1060 ▪ the default (non qualified) namespace. All schema have their own default
1061 namespace generated according to the following regular expression:
1062 “urn:swift:xsd:\$+”. Where the “+” must be replaced by the message name possibly
1063 prefixed by the collaboration name separated by a ‘.’.
- 1064 ▪ xs: W3C XML schema namespace (not used in instances)
- 1065 ▪ xsi: W3C XML schema-instance namespace
- 1066 ▪ a target namespace (for schema only) which is the same as the default namespace.

1067 Schema:

```
1068 <schema  
1069     xmlns="urn:swift:xsd:$NoticeOfExecution"  
1070     xmlns:xs=" http://www.w3.org/2001/XMLSchema"  
1071     targetNamespace="urn:swift:xsd:$NoticeOfExecution">
```

1072 Instance:

```
1073 <NoticeOfExecution  
1074     xmlns="urn:swift:xsd:$NoticeOfExecution"  
1075     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

1076 **3.2.2.2 Schema location in the XML instance**

1077 The rootelement of the business payload carries the location (as an Universal Resource
1078 Information) of the XML Schema, in the form of the XML attribute `xsi:SchemaLocation`.

1079 It is not allowed to have `xsi:SchemaLocation` to appear in any element (much like `xmlns`)
1080 but only in the rootelement of the business payload.

1081 Instance:

```
1082 <NoticeOfExecution xsi:schemaLocation="file://file_path">
```

1083 **3.2.2.3 XML facets on simple types**

1084 The following sections describe the facets that will be introduced in the XML schema.

1085 Those XML schema facets are based on the UML invariants which have been specified in
1086 the UML model according to the section [3.4. Summary of UML invariants related to](#)
1087 [schema production](#).

1088 3.2.2.3.1 pattern

1089 Pattern matching allows lexical validation on strings, which syntax can be described using
1090 regular expressions, (commonly referred to as “[Perl](#) expressions”).

1091 This facet only applies to strings.

1092 The exact syntax of the allowed regular expressions is defined in appendix E of “XML
1093 Schema Part 2: Datatypes” (XML Schema’s W3C Recommendation May 2001).

1094 For instance:

```
1095 <xs:simpleType name='BIC'>
1096   <xs:restriction base='string'>
1097     <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}'/>
1098   </xs:restriction>
1099 </xs:simpleType>
```

1100

1101 3.2.2.3.2 length, minLength, maxLength

1102 XML schema allows restriction of the value space of any string value (i.e.: **double, integer,**
1103 **date etc are not affected**) by using the following constraining facets:

- 1104 ▪ length
- 1105 ▪ minLength
- 1106 ▪ maxLength

1107 Those facets only apply on strings, and their values must be positive integer values.

1108 For instance, a BankAddress is a string of 10 characters minimum and 40 characters
1109 maximum:

```
1110 <xs:simpleType name='BankAddress'>
1111   <xs:restriction base='string'>
1112     <xs:minLength value='10'/>
1113     <xs:maxLength value='40'/>
1114   </xs:restriction>
1115 </xs:simpleType>
```

1116

1117 **3.2.2.3.3 minInclusive, maxInclusive, minExclusive, maxExclusive**

1118 XML schema allows restriction of the value space of any numerical value by using the
1119 following constraining facets:

- 1120 ▪ minInclusive
- 1121 ▪ minExclusive
- 1122 ▪ maxInclusive
- 1123 ▪ maxExclusive

1124 Those facets only apply to numerical values (Integer, Long, BigDecimal, Float, Double)
1125 and to time measurement related values (Date, Time,...) and their value must be constants
1126 of the same type than the numeric value they apply to.

1127 For instance, the financial instrument below must contain between 1 and 100000 securities:

```
1128 <xs:complexType name='SecuritiesInstrument'>
1129   <xs:sequence>
1130     <xs:element name='ISIN' type='string'/>
1131     <xs:element name='Quantity' >
1132       <xs:simpleType>
1133         <xs:restriction base='xs:decimal'>
1134           <xs:minInclusive value='1'/>
1135           <xs:maxInclusive value='100000'/>
1136         </xs:restriction>
1137       </xs:simpleType>
1138     </xs:element>
1139   </xs:sequence>
1140 </xs:complexType>
1141
```

1142 **3.2.2.3.4 enumeration**

1143 XML schema allows restriction of the value space of an enumeration by using the
1144 enumeration constraining facet.

1145 This facet only applies to enumerations, and their value must be part of the original
1146 enumeration from which they restrict.

1147 For instance, a class M containing an attribute b of type E1 with an XML Invariant
1148 restricting the enumerated value to Value2:

```
1149 <xs:complexType name="M">
1150   <xs:sequence>
1151     <xs:element name = "b">
1152       <xs:simpleType>
1153         <xs:restriction base="E1">
```

```
1154         <xs:enumeration value = "Value2"/>
1155     </xs:restriction>
1156 </xs:simpleType>
1157 </xs:element>
1158 </xs:sequence>
1159 </xs:complexType>
1160 <xs:simpleType name = "E1">
1161     <xs:restriction base = "xs:string">
1162         <xs:enumeration value = "Value1"/>
1163         <xs:enumeration value = "Value2"/>
1164     </xs:restriction>
1165 </xs:simpleType>
1166
```

1167 3.2.2.3.5 totalDigits, fractionDigits

1168 Fixed point decimal values need a totalDigits specification (i.e. the maximum number of
1169 decimal digits in values of datatypes derived from decimal: totalDigits), as well as a
1170 fractionDigits specification (i.e. the maximum number of decimal digits in the fractional
1171 part of values of datatypes derived from decimal: fractionDigits).

1172 The value of the totalDigits facet must be a positive integer.

1173 The value of the fractionDigits facet must be a non-negative integer.

1174 For instance, requiring a totalDigits of 8 digits with 2 digits after the decimal point on an
1175 amount would translate to the following instance:

```
1176 <xs:simpleType name='Amount'>
1177     <xs:restriction base='xs:decimal'>
1178         <xs:totalDigits value='8'>
1179         <xs:fractionDigits value='2'>
1180     </xs:restriction>
1181 </xs:simpleType>
```

1182 3.2.2.4

1183 Nillable

1184 To be used in conjunction with the XML-nil attribute. The Schema attribute [nillable](#)
1185 specifies whether the instance can carry a nil value. Default value is [false](#).

1186 In the following schema:

```

1187     <xs:complexType name=' FinancialInstrument'>
1188         <xs:sequence>
1189             <xs:element name='ISIN' type='string' />
1190             <xs:element name='Quantity' type='xs:decimal' nillable='true' />
1191         </xs:sequence>
1192     </xs:complexType >

```

1193 Only the Quantity can carry a nil value.

1194

1195 It should be noted that nillable is not a facet, but an attribute (as abstract, minOccurs,
1196 maxOccurs, ...). This implies that, in the schema's context, nillable applies to an element
1197 (and not a type).

1198 Therefore the nillable option should consequently not be encoded as an invariant on a class
1199 in the UML model. It will thus be set either at the attribute or role level (in which case the
1200 corresponding element in the schema would be nillable).

1201 In the XML instance document, the XML attribute **nil** can be used to indicate that an
1202 element has no value.

1203 Assuming the following schema:

```

1204     <xs:complexType name='OrderOfBuy'>
1205         <xs:element Securities type='FinancialInstrument' />
1206     </xs:complexType >

1207     <xs:complexType name=' FinancialInstrument'>
1208         <xs:element name='ISIN' type='string' />
1209         <xs:element name='Quantity' type='xs:decimal' nillable='true' />
1210     </xs:complexType >

```

1211 An order-of-buy XML instance with no quantity of securities specified (as opposed to a
1212 value of zero) will be expressed as:

```

1213     <OrderOfSell>
1214         <Securities>
1215             <ISIN>BE1234567890 </ISIN>
1216             <Quantity xsi:nil='true' />
1217         </Securities>
1218     </ OrderOfSell >

```

1219 Note that an alternative to not using the 'nil' XML-attribute is to omit the nil element. By
1220 doing so we introduce an ambiguity between **not** specifying an optional element and
1221 specifying an optional element which value is **nil**.

1222 3.3 Granularity of Schemas

1223 There is one Schema per message.

1224 **3.4 Summary of UML invariants related to schema production**

1225 Those invariants will be defined as user properties on methods having the <<inv>>
1226 stereotypes, on the tab called XML Invariants.

1227

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

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 >

1228 4 Naming Conventions and Taxonomy

1229 See [Naming Conventions appendix](#)

1230 5 Character set

1231 SWIFTStandards XML uses UTF-8 as the (default) character encoding mechanism, for the
1232 following reasons:

- 1233 • It has the most efficient method of character representation:
 - 1234 • It is the shortest method to represent the characters which are currently the most
 - 1235 • commonly used in a financial environment (ASCII and EBCDIC characters)
- 1236 • It can still represent almost any known character

1237 • It is interoperable with many other encoding schemes through (automatable) conversion
1238 algorithms.

1239 Example:

1240 <?xml version="1.0" encoding="UTF-8"?>

1241

1241

1242 **6 Appendices**

1243

1244 **A Naming conventions and taxonomy scheme**

1245 **A.1 Introduction**

1246 The purpose of this appendix is to explain the methodology to be used when generating
1247 meaningful SWIFTStandards XML tags. It provides the general principles of classification
1248 of SWIFTStandards XML tags.

1249 It is very important to have a structure in place to ‘control’ the way data elements are
1250 tagged.

- 1251 • It will allow us to keep SWIFTStandards XML as condensed as possible
- 1252 • It will limit a proliferation of different usages by different developers
- 1253 • It provides a way to easily trace and manage a SWIFTStandards XML (and UML)
1254 repository which is based on meaningful tags.

1255

1256 **A.2 Constraints / Assumptions**

- 1257 • To use normalised names: to abide to the tagging constraints imposed by the W3C
1258 XML specification v1.0., C++ and Java
- 1259 • To have one namespace that will contain all business elements covered by S.W.I.F.T.
- 1260 • To have no elements that can be expressed in SWIFTStandards XML and cannot be
1261 expressed in UML and vice-versa.
- 1262 • To have business information which is expressed as SWIFTStandards XML
1263 elements/values and meta data information which is expressed as SWIFTStandards
1264 XML attributes.
- 1265 • To have Schema’s that support inheritance. Consequently, attributes and aggregates can
1266 be reused or overridden.

1267

1268 **A.3 Naming rules**

1269 As already stated, the SWIFTStandards XML name is the XML name assigned to the UML
1270 element or by default the UML name. Hence, all below rules apply for XML names,
1271 SWIFTStandards XML names and UML names of elements without XML names.

1272 **A.3.1 General rules**

1273 Use the English vocabulary.

1274 Abide to the (character) restrictions described in SWIFTStandards XML for naming
1275 elements:

- 1276 • All names must start with an alphabetic character.
1277 • All characters following the first characters must be alphabetic characters, numeric
1278 characters, or ‘_’.

1279 Apply camel case convention:

- 1280 • Names for elements and attributes may be made up of multiple words each consisting of
1281 alphanumeric characters.
1282 • Each word starts with a capital letter.
1283 • All white spaces between words are removed.

1284
1285

1285

1286 **End of document**

1287

1288