SEMANTIC AND TECHNICAL STRUCTURE OF ELECTRONIC MESSAGES STORAGE AND RETRIEVAL OF MODELS

Andreas.pelekies@gefeg.com

SCOBDO

SCOBDO
SEMANTIC CONVERSION OF BUSINESS DOCUMENTS

Table of contents

Table of contents 1
Version history 2
Status 2
Introduction 3
Access to the SCOBDO repository 3
Structure of the repository 3
Reference data model 4
Message specifications 4
  Comparison of restriction.xsd and XSD-Types 7
Mapping from specification elements to the reference model 8
Codes 12

Version history

Version	Date	Remarks	Author
0.1	22-01-2017	Initial version	GEFEG
0.2	27-01-2017	After discussion with Fred	GEFEG
0.3	01-02-2017	Changed storage principles	GEFEG
1.0	31-12-2017	Final version	GEFEG

This is a living document. Functions may be added or changed as they are implemented.

Status

Public

Co-financed by the European Union
Connecting Europe Facility

Introduction

This document describes which technical documents will be stored in a repository how they are designed. Although JoinUp[1] as a collaborative platform created by the European Commission was chosen to be used at first, during the progress of the project the repository within JoinUp ceased to exist. So an alternative repository was created by the SCOBDO team members. This document gives an overview of and examples on the different formats used. The technical files (XSD files) that correspond to the different formats are available as separate files. The user interface and the workflow within the SCOBDO project are not part of this document.

Access to the SCOBDO repository

The repository of the SCOBDO project is based on the free Apache Subversion[2] software (SVN). It is accessible by using a freely available SVN client and using the following credentials for login:

URL: https://svn.gefeg.com/JoinUp
Username: JoinUp.Reader
PW: scobdo9215!

This access is limited to reading the saved data. A write access can be freely provided on individual request. Please contact support@gefeg.com for this or any other related question.

Structure of the repository

The `meta` folder contains XML schema files that define the structure of information saved in the repository. For details see the following paragraphs.

The `rdm` folder contains a XML schema collection for the applied reference data model. During the development of SCOBDO it was chosen to use the UN/CEFACT supply chain reference data model (SCRDM) with a limitation to the invoice aspect. More information about this reference data model can be found at the UN/CEFACT website[3]. The following figure shows just the basic structure of this data model.

The `specifications` folder contains the real information needed for the SCOBDO portal. The subdirectories are named by users of the SCOBDO client (specifiers). The name normally represents a specific invoice standard and version that can be used as a source or target within SCOBDO.
There exist three XML files defining the applied mapping. The `guideline.xml` defines all restrictions that are defined by a specifier and applied to the original XSD structure of the invoice standard. The structure is based on restrictions.xsd.

The `mappingfromrdm.xml` and `mappingtordm.xml` files are structured as defined in mappings.xsd and contain all mappings including the respective mapping functions.

[1] See http://joinup.ec.europa.eu
[2] See https://subversion.apache.org
[3] See https://uncefact.unece.org/pages/viewpage.action?pageId=5472398

The `xsd` folder contains the original XML schema files that define the structure of the given invoice standard.
If codelist mappings are defined, a `code_mappings` folder holds subfolders for each defined and mapped code list. Within the subfolder two XML files, `codemappingtordm.xml` and `codemappingfromrdm.xml` define the respective code mappings.

The following figure shows the basic stucture of the repository.

```
root/
|-- meta/
|    |-- Restrictions.xsd
|    |-- Mappings.xsd
|    └-- CodeListMapping.xsd
|-- rdm/
|    └-- uncefact/
|         |-- standard/
|         |-- codelist/
|         └-- data/
└-- specifications/
     |-- MySpec1 with version 1.2/
      |    |-- guideline.xml
      |    |-- mappingtordm.xml
      |    |-- mappingfromrdm.xml
      |    |-- xsd/
      |    |    └-- folder structure xsds/
      |    └-- code_mappings/
      |         └-- folder for each code mapping (name: shortname)/
      |          |-- codemappingtordm.xml
      |          └-- codemappingfromrdm.xml
      |-- MySpec2 with version 4.3/
      └-- MySpec3 with version 2.0/
```

Reference data model

As the reference model the D16B SCRDM (Subset) CII coupled or uncoupled XML schemas are used. All specifications will be mapped to this reference model for both directions. The model itself can be downloaded at the UN/CEFACT website[4].

Message specifications

In this document “message specifications” reflect a specific standard for an electronic invoice like ZUGFERD, PEPPOL, e-SENS and others.

To store the structural information of each standard the original XSD files are used. This assures to use the same basic design model (e.g. venetian blind) and the ability to verify the compliance of SCOBDO output files to the standard specific requirements.

[4] See https://www.unece.org/cefact/xml_schemas/index

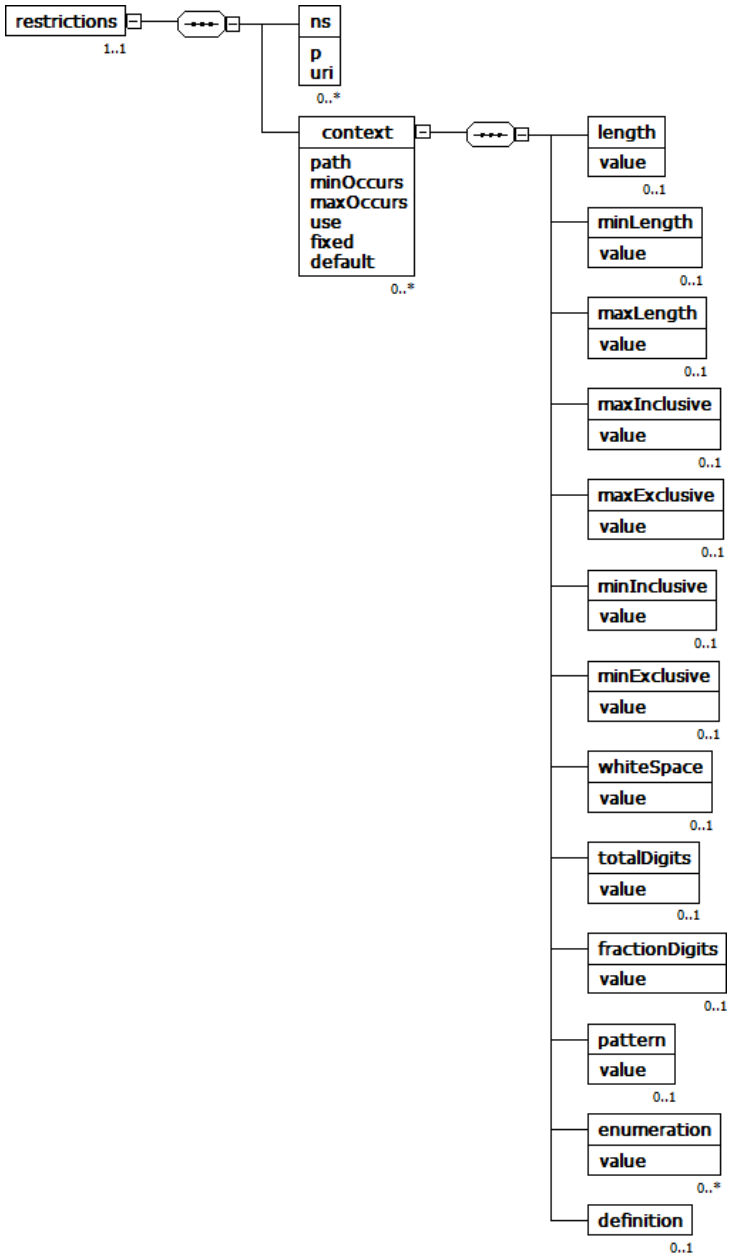
In practice a standard is often profiled. This means that context specific restrictions (e.g. business rules) are applied on the original structure. In some standards they are only defined verbally (e.g. in a user specification), in others they are realized by checking the instance files (for instance by applying schematron files). As a third category there may exist modified schema files that reflect the applied business rules. As within SCOBDO it is neither possible neither to predict which version is used nor to prescribe a specific method, is a common approach for all those kinds of implementation needed. The SCOBDO answer to this is to save those as a separate XML file, that eases implementation by external solution providers. At the same time intellectual property rights (IPR) of the original XSDs is not violated by any modification. To simplify additional implementations, the structure of the profile.xml is defined as close as possible to the capabilities of XSD schemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<restrictions xmlns="urn:scobdo:restrictions">
  <ns prefix="p" uri="example"/>
  <context path="Name" minOccurs="3" maxOccurs="3" default="AAAAA">
    <minLength value="5" />

    <!-- all other xsd facets -->

    <enumeration value="AAAAA"/>
    <enumeration value="BBBBB"/>

    <definition>The scenario or setting of a Cross Industry Invoice
    (CII) exchanged document, such as its business process
    application context.</definition>

  </context>
  <context path="CityName" maxOccurs ="0"/>
  <context path="Test/p:SubElement/@id" use ="required"/>
</restrictions>
```

The most important element is „context“. That element is repeatable. It has a path to a specific schema context and shows all restrictions made at this context. The path is relative to the root element. If attribute „maxOccurs“ is set to „0“ no further restrictions are within the „context“ element.

In addition there is an element „ns“. It is also repeatable and lists all prefixes and namespace that are used in the „path“ attribute of element „context“.

The structure for this is defined in „restrictions.xsd“. It can be used for validation of this structure. The corresponding diagram is shown as follows:


restrictions
1..1
ns
p
uri
0..*
context
path
minOccurs
maxOccurs
use
fixed
default
0..*
length
value
0..1
minLength
value
0..1
maxLength
value
0..1
maxInclusive
value
0..1
maxExclusive
value
0..1
minInclusive
value
0..1
minExclusive
value
0..1
whiteSpace
value
0..1
totalDigits
value
0..1
fractionDigits
value
0..1
pattern
value
0..1
enumeration
value
0..*
definition
0..1

Comparison of restriction.xsd and XSD-Types

The following table shows how the different facets are designed in a schema file and how they are implemented within the restriction.xsd. It can easily be seen that the basic structure is equal. This makes implementation more efficient.

Standard schema		**Restriction.xsd**		
Facet or equal	Element / Attribute	Path	Element / Attribute	Type
Default	Attribute	default	Attribute	AnySimpleTypeValue
annotation/documentation	Elements	definition	Element	xs:string
enumeration	Element	enumeration	Element	AnySimpleTypeValue
Fixed	Attribute	fixed	Attribute	AnySimpleTypeValue
fractionDigits	Element	fractionDigits	Element	NonNegativeValue
length	Element	length	Element	NonNegativeValue
maxExclusive	Element	maxExclusive	Element	DecimalValue
maxInclusive	Element	maxInclusive	Element	DecimalValue
maxLength	Element	maxLength	Element	NonNegativeValue
minExclusive	Element	minExclusive	Element	DecimalValue
minInclusive	Element	minInclusive	Element	DecimalValue
minLength	Element	minLength	Element	NonNegativeValue
pattern	Element	pattern	Element	StringValue
totalDigits	Element	totalDigits	Element	NonNegativeValue
use	Attribute	use	Attribute	Use
whiteSpace	Element	whiteSpace	Element	WhiteSpaceValue

Mapping from specification elements to the reference model

To store the mappings there are two files necessary. The first file defines the mapping from the specification to the model. The second file defines the mapping from the model to the specification.

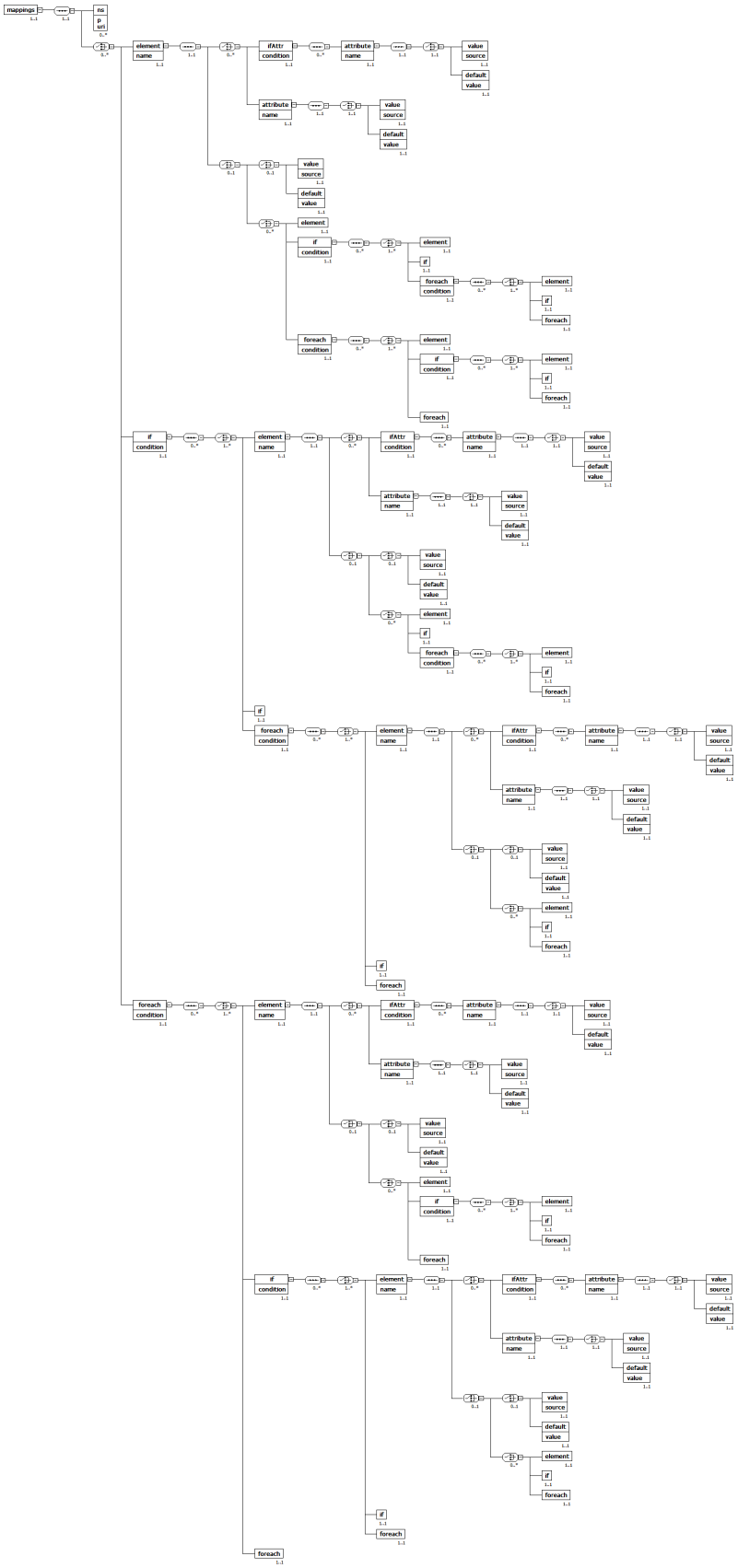
The following figure shows a sample of a mapping definition file.

```
<mappings>
  <ns p="fn" uri="http://www.w3.org/2005/xpath-functions"/>
  <ns p="fncm" uri="http://www.gefeg.com/xslt/codemapping"/>
  <element name="Person">
    <element name="Name">
      <attribute name="Abbr">
        <ifAttr condition="Person/Stature">
          <value source="Person/Stature/NameS/@AbbrS"/>
        </ifAttr>
      </attribute>
      <attribute name="MaidenName">
        <default value="FRANZI"/>
      </attribute>
      <if condition="Person/Stature/@Height=&apos;Tall&apos;">
        <value source="Person/Stature/NameS/text()"/>
      </if>
    </element>
    <element name="Address">
      <value source="Person/Stature/AddressS/text()[fn:upper-case($value)]"/>
    </element>
    <if condition="Person/Stature/@Height">
      <foreach condition="Person/Stature/ChildS">
        <element name="Kind">
          <attribute name="Sex">
            <value source="@SexS[fncm:sex($value)]"/>
          </attribute>
          <element name="Vorname">
            <if condition="@SexS=&apos;W&apos;">
              <value source="GivenNameS/text()[upper-case($value)]"/>
            </if>
            <if condition="@SexS=&apos;M&apos;">
              <value source="GivenNameS/text()[lower-case($value)]"/>
            </if>
          </element>
        </element>
      </foreach>
    </if>
  </element>
</mappings>
```

The principle is a mapping defined from the target perspective (The place where things are mapped to). For this the basic structure represents elements named `element` and `attribute`. They are used to defines the target elements or attributes to be mapped to. If an element or attribute is filled with a distinctive value a sub-element `value` can be defined. This is the standard case when a specific source value should be selected for output. The corresponding Xpath is defined in the `source` attribute of this element. Alternatively, `default` can be chosen for example to represent fixed values where no additional condition is applied.
It is possible to express conditions by using the elements `if`, `foreach` and `ifAttr`. By adding these constructs if and for-each constructs are supported. The corresponding rules are stated in the `condition` attribute.

The same way as it is defined for restrictions with the `ns` element namespaces and their prefixes can be defined. This list shall be complete according to the namespaces and prefixes defined in the corresponding mapping sources and target

The schema for this xml file is stored in “mappings.xsd”. The following diagram shows the general structure:


mappings
ns
p
uri
element
name
ifAttr
condition
attribute
name
value
source
default
value
if
condition
foreach
condition

Codes

As code list mappings are needed for more than just two specifications, they are stored and managed on a separate level. Code list mappings are always described in addition to the mappings between elements and attributes. For a complete mapping both directions need to be defined, which leads again into storing of two separate files - one for each direction.

```
<codeListMapping>
  <sourceCodeList agency="EDIFACT"
      name="UNITS OF MEASUREMENT"
      version="D11A" />
  <targetCodeList agency="Dutch Customs"
      name="UNITS OF MEASUREMENT"
      version="4711" />
  <item source="CM" target="cm" />
</codeListMapping>
```

“sourceCodeList” and “targetCodeList” give the information about the properties of the code lists involved. Element “item” is unbounded and has in its “source” attribute a code from the source code list and in the “target” attribute the code value that is needed in the target.

A schema for that is provided in “codelistmapping.xsd”. The general structure is displayed in that diagram: