Best Practice for DSDL-based Validation

Soroush Saadatfar

ADAPT Centre

<Soroush.Saadatfar@ul.ie>

David Filip

ADAPT Centre

<David.Filip@adaptcentre.ie>

Abstract

This paper proposes a best practice guide to apply Document Schema Definition Languages (DSDL) for validation of an arbitrary industry vocabulary. The research is based mainly on a practical case study of creating such an optimized set of DSDL validation artefacts for XLIFF 2, a complex industry vocabulary. Available schema languages have advanced functionality, enhanced expressivity and can be used in concert if needed. This advantage, on the other hand, makes the creation of a stable and robust set of validation artefacts hard, because there would usually be more than one way to describe the same Functional Dependencies or Integrity Constraints and various validation tasks can be solved by more than one schema language.

Keywords: DSDL, validation, expressivity, progressive validation, constraints, functional dependencies, XLIFF, Schematron, NVDL

1. Introduction

Validation is a key component of processing vocabularies based on XML (eXtensible Markup Language) [1]. This nontrivial task has been approached by a number of various initiatives according to the needs of specific target data models. Initially, DTD (Document Type Definition) [2] was widely used to define structure of XML documents and usually combined with programmatic approaches to validate the constraints that were not expressible in DTD. Several schema languages followed since DTD to enhance the expressivity for different XML constraints, however, the programmatic approach to tackle advanced contsraints validation had not been fully superseded. Although, the first validation technique has advantage of expressivity and trancparency being standardized.

Our focus in this paper will remain on non-programmatic, standards driven, trasparent approaches based on machine readable implementation independent artefacts. We aim to illustrate the potential DSDL methods and schema languages have to replace ad hoc programmatic validation approaches based on our experience with XLIFF (XML Localization Interchange File Format) [3], an OASIS standard which has been widely adopted in the localisation industry since its inception.

XLIFF has a multimodal structure, comrising a core namespace and several module namespaces; a complex data model designed to fulfill current and future needs of the industry. It is intended to complete a round-trip in the localisation workflow and to be dynamically *Modified* and/or *Enriched* by different *Agents* who manipulate the XLIFF data in accordance with their specialized or competing functionality. This XML vocabulary does not follow the normal XML behaviour when it comes to usage of `ID` attributes and it defines a number of scopes for NMTOKEN [4] keys (that are called IDs in XLIFF context) instead of the usual XML convention where the `ID` attributes are required to be unique throughout the document. The task of internal fragment referencing therefore cannot be implemented using native XML `IDREF` attributes either. The standard specifies several levels of date driven structures for NMTOKEN keys as well as introducing the *XLIFF Fragment Identification Mechanism* to replace the XNL `ID` and `IDREF` concepts respectively. These approaches have been chosen with needs of the industry in mind and will be discussed in detail in the following section. Also some other relational dependencies that XLIFF sets, have a complex nature and logic and some of these we will also be covered in the next section of this paper. As XLIFF files are meant to circulate in arbitrary workflows, it is vital - for purposes of lossless data exchange - that all workflow token instances conform to the XLIFF Specification completely

doi:10.14337/XMLLondon16.Saadatfar01

and do not violate any of its advanced constraints. Therefore comprehensive validation should be applied after any modification. Programmatic validation, mainly applied to its earlier version, XLIFF 1.2, led to misinterpretations of the standard in many cases and developers never implemented the full list of constraints specified by XLIFF 1.2. These issues motivated the XLIFF TC and the authors of this paper to find an exhaustive solution for validation of XLIFF 2.x (XLIFF 2.0 and its backwards compatible successors) to provide a unified and transparent platform for validating XLIFF instances against the full specification. Research revealed that the DSDL (Document Schema Definition Languages) [5] framework is capable of providing such a solution. This attempt succeeded and brought Advanced Validation Techniques for XLIFF 2 [6] to be part of the standard, starting from the 2.1 version, scheduled to release in 2016. In the following sections, we will try to generalize this work to provide mapping for XML constraints and appropriate DSDL method to use. We believe that transparent and standardized validation is a prerequisite for achieving interoperability in workflows and that the DSDL framework provides enough expressivity for producing machine readable validation artefacts for arbitrary XML industry vocabularies and data models.

2. Analysis of the XML Data Model

XML has a very simple, but at the same time universal and generalized nature as a data model: (a) it is composed of only two objects: XML nodes and their values (b) data model expressed through value and structure of nodes (c) it defines a minimum set of rules in terms of syntax. These properties enable XML to deliver its main task- extensibility. Each XML-based vocabulary represents the target data model by specifying valid scenarios of structure and values of declared nodes.

In this section, we will discuss constraints of XLIFF 2 from general XML point of view. We will first have a brief look at the structure and purpose of XLIFF. We review the literature for available definitions and notations of XML constraints in the next step and then apply them to XLIFF. This section will also include *Processing Requirements* that XLIFF specifies for different types of users (*Agents*) as they modify XLIFF instances. This type of constraints require progressive (dynamic) validation as they are based on comparison of files before and after Agents perform changes.

2.1. XLIFF Structure

The main purpose of XLIFF is to store and exchange localizable data. Typically, a localisation workflow contains processes like extraction, segmentation (process of breaking down the text into the smallest possibly translatable linguistic portions, usually sentences), metadata enrichment (like entering suggestions based on previous similar content, marking up terminology etc.), translation (editing of target content portions corresponding to source content portions) and merging the payload (the translated content) back to the original format. An XLIFF document can facilitate in its progressing instances all the aforementioned tasks. The extracted segmented content will be placed in the `source` element children of `segment`. The translated content is kep aligned in `target` siblings of the `source` elements; it can be added later in the process. In other words, the flow of original text transforms into sequence of `segment` elements within a `unit` element, the logical container of translatable data. Parts of the content which are not meant to be translated can be stored in the `ignorable` siblings of the `segment` elements. `unit` elements can be optionally structured using a `group` parent (ancestor) recursively. Finally, a `file` elements will wrap the possibly recursive structure of `group` and `unit` elements. An XLIFF instance (an XML document with the root `xliff`) can have one or more `file` elements. In order to preserve metadata and markup within text (e.g. the HTML `<b>` tag), XLIFF has 8 *inline elements* , (e.g. `pc` element in "Listing 1" for well formed paired codes) some recursive, which may appear along with text in `source/target` pairs. The native codes may be stored in `originalData` elements, which are then referenced from inline elements. To avoid confusion, we present a simplified notion of XLIFF and skip many other structural parts. "Listing 1" shows a sample XLIFF file.

Listing 1 - Sample XLIFF instance

```
<xliff version="2.0" srcLang="en" trgLang="fr">
  <file id="f1">
    <unit id="u1">
      <originalData>
        <data id="d1">&lt;b&gt;</data>
        <data id="d2">&lt;/b&gt;</data>
      </originalData>
      <segment>
        <source>
          Some <pc id="pc1" dataRefStart="d1"
          dataRefEnd="d2">Important</pc>text.
        </source>
        <target>
          Un texte <pc id="pc1" dataRefStart="d1"
          dataRefEnd="d2">important</pc> de.
        </target>
      </segment>
      <ignorable>
        <source>Non-translatable text</source>
      </ignorable>
      <segment>
        <source>Second sentence.</source>
        <target>Deuxième phrase</target>
      </segment>
    </unit>
  </file>
</xliff>
```

2.2. XML Constraints

The structural relation of XML nodes is the first step of shaping the *tree* of the targeted data model. All other constraints can be specified only afterwards. Generally, studies for defining XML constraint have proposed keys [7], foreign keys [8] and functional dependencies [9], [10]. These papers represent each type of constraint through mathematical expressions which is out of scope of this paper as our goal is to match every category with appropriate schema language in practice. Therefore we will consider general types as well as some of their special cases.

2.2.1. Keys and foreign keys

The concept of `ID` and `IDREF`, introduced by DTD, cover the category of keys and foreign keys respectively. However, these attributes implement only special cases of these types, *absolute keys/foreign keys* , setting the scope of keys to the root element, i.e. absolute path. In XLIFF, for instance, only `file` elements define absolute keys whilst keys are specified at 14 points of XLIFF core elements. Relative keys can be of various complexity depending on steps of relativity they designate (starting from zero steps, i.e. absolute keys). For example for `unit` elements, with only one step of relativity (from the root element), scope of key uniqueness is the parent `file` element. Number of steps for one key can be a variable if the corresponding element is allowed at different places of the XML tree, like `data` elements with unique keys in the scope of `originalData`, where the latter element might occur at several levels of the hierarchy. These variables, however, have a minimum value stating how close the element can be to the tree root, in the case of `data` the minimum number of steps is 3. In a more complicated scenario keys might be shared among nodes of distinct types and distinct parents. Keys for all inline elements, some of which can be recursive, are set at the `unit` level, but only for those appearing in the source text whereas elements of the target must duplicate their corresponding key in source. This type of content, where an element can have a mix of text and other elements, is usually being dropped in attempts of generalizing Keys for XML as an assumption [11] and therefore not well researched.

XLIFF, when it comes to foreign keys, has all of its referencing attributes relative (e.g. `dataRefStart` attribute in "Listing 1"). Even though cross-referencing is allowed only at the unit level, the standard defines format of IRIs pointing to XLIFF documents through Fragment Identification. This mechanism allows one to specify a unified path to any node with identifier. Therefore the notion `<pc id="1" dataRefStart="#/f=f1/u=u1/d=d1" ...` is an alternative valid value which specifies the referenced node by an absolute path instead of the shorter form used in Listing 1, where identifier is given relatively (within the enclosing unit).

2.2.2. Functional Dependencies

Attempts of generalizing this nontrivial category for XML has been limited so far and define only some variations of Functional Dependencies (*FD*) [12]. A significant progress has been made though, by applying mappings of paths in XML document to relational nodes. Basically, FDs specify functional relations of XML nodes and play an important role in data models being the most complex of XML Integrity Constraints. Co-occurrence constraints and value restrictions are some usual variations of FDs. XLIFF has various FDs, some of which we review in this paper. The `target` element has an optional `xml:lang` attribute, but when present must match `trgLang` attribute of the root element. The latter attribute, on the other hand, is initially optional (at early

stages of the localization process), but must be present when the document contains `target` elements. Another interesting and complex FD in XLIFF relates to the optional `order` attribute of `target`. As was mentioned earlier, the sequence of `segment` and `ignorable` elements specifies the flow of text at the unit level, but sentences of a translated paragraph may have different order than in the source language. In this case `target` elements must specify their actual "deviating" position respective to the default sequence. This value then points to the corresponding `source` element for ID uniqueness constraints over inline elements "Listing 2" and "Listing 3" show a paragraph with three sentences in different order, represented in HTML and XLIFF respectively.

Listing 2 - Paragraph with disordered sentences after translation

```
<p lang='en'>Sentence A. Sentence B. Sentence C.</p>
<p lang='fr'>Phrase B. Phrase C. Phrase A.</p>
```

Listing 3 - Usage of `order` attribute in XLIFF

```
<unit id="1">
 <segment id="1">
  <source>Sentence A.</source>
  <target order="5">Phrase A.</target>
 </segment>
 <ignorable>
  <source> </source>
 </ignorable>
 <segment id="2">
  <source>Sentence B.</source>
  <target order="1">Phrase B.</target>
 </segment>
 <ignorable>
  <source> </source>
 </ignorable>
 <segment id="3">
  <source>Sentence C.</source>
  <target order="3">Phrase C.</target>
 </segment>
</unit>
```

The XLIFF file in Listing 3 is valid, in terms of `order` constraints, as (a) values are between 1 and 5 (segments and ignorables combined) (b) each `target` element occupying a different position than its natural explicitly declares its order.

2.2.3. Data Types

Constraints of this category apply various rules on values that can be assigned to XML nodes. The concept of Simple and Complex data types was introduced by W3C XML Schema [13] and provides a solid library that enables one to build custom data types through manipulating simple data types and specifying restrictions for them. A number of libraries have been developed after that to target specific needs. Allowed values can be defined in many different ways including set of fixed values, default values, forbidden values/characters, mathematical restrictions for numeral values, specific or user-defined format etc. We will return to this topic in the following sections.

2.2.4. Progressive Constraints

Some data models are designed to perform in different stages of their life cycle and therefore might need to be validated against different set of constraints according to the stage they are at. *Initially* optional `trgLang` attribute of XLIFF, which was mentioned earlier, serves as a good example for this case. But some advanced constraints, like XLIFF Processing Requirements, focus on the applied modifications in documents and perform validation based on comparison of data before and after changes were made. For example, if value of one attribute has been changed by the last user, value of some other nodes must be changed as well. Technically, this type can be considered as a cross-document functional dependency, but as XML vocabularies are being often used for exchange purposes, this might grow into a category on its own in the future. XLIFF classifies its users (Agents) based on the type of general task they carry out (e.g. Extracting, Enriching etc.) and assigns different sets of Processing Requirements to each group.

In the following section we will examine expressivity of popular XML schema languages against each of the aforementioned types.

3. Implementing XML Constraints

After specifying XML Integrity Constraints, we now will explore schema languages which can implement the constraint types in the previous section. W3C XML Schema is the schema language to define the structure of XML trees with the widest industry adoption. The DSDL framework, on the other hand, is a multipart ISO standard containing various languages for different validation tasks and broad domain of constraint types.

Schema languages often perform across constraint types and DSDL provides the framework for mapping and using them together. In the current section we aim to highlight *the task* each language can handle the best. Following such guidance will contribute to optimizing performance level of the validation process.

3.1. XML Schema

This schema language is useful for defining the XML tree nodes and their relations in the XML tree. XML Schema uses XML syntax and Data types [4], the second part of the language, introduces an advanced library of data types which is widely used and referenced by other schema languages. Users can apply different restrictions to values of elements or attributes. XML Schema supports only absolute Keys and foreign Keys using a limited implementation of XPath [14], a syntax for regular expressions in XML. The concept of key in XML Schema presumes that the attribute must always be present and thus cannot be applied to optional attributes. Finally, XML Schema cannot target Functional Dependencies of any level of complexity.

3.2. DSDL framework

Some parts of DSDL, like RelaxNG [15] and Schematron [16], were standalone projects initially that were subsequently standardized as part of the framework. For the goals of this paper, we only review 3 parts of the framework that together enable full expressivity for XML Integrity Constraints.

3.2.1. RelaxNG

This schema language describes structure and content of information items in an XML document through a tree grammar. The grammar-based validation RelaxNG offers is an easy and convenient approach, although it keeps the expressivity power of this language close to the level of XML Schema. RelaxNG has some basic built-in data types so other libraries (e.g. XML Schema Data types) should be used for advanced requirements in this type of constraints. Although it does not support any variations of Keys and foreign Keys, RelaxNG is able to cover some basic Functional Dependencies like co-occurrence constraints based on values or presence of XML nodes. Defining such constraints in RelaxNG, however, might wind up not pragmatic for vocabularies with a large number of nodes. such as XLIFF. For instance, the constraint on `trgLang` attribute in XLIFF, which was mentioned earlier, could, theoretically, be expressed in RelaxNG by defining two possible valid grammars, but this would unfortunately effectively double the volume of the schema. Overall, RelaxNG is a simple language to use compared to XML Schema. Its focus on targeting only one problem has made RelaxNG an efficient schema language [17] and therefore more and more industry vocabularies tend to pick this language for validation tasks. We developed an experimental RelaxNG schema for XLIFF 2, yet this was not adopted as part of the advanced validation feature for XLIFF 2.1 by the XLIFF TC due to a wide overlap with the XML Schema, which already was a normative part of the standard and needs to be kept for backwards compatibility reasons.

3.2.2. Schematron

The rule-based validation allows Schematron to catch Functional Dependencies violations. This schema language provides full support for both XPath regular expressions and functions which enables users to define comprehensive sets of XML paths to express an arbitrary Functional Dependency, key or foreign key. Each Schematron rule describes permitted relationships between document components by specifying a context, i.e. the XML node(s) where the current rule applies and then conducting the user-defined test. Because XPath integration provides a powerful mechanism for navigating through XML trees, Functional Dependencies often may be expressed in multiple ways. It is a convenient approach to first define the subject and the object of a Functional Dependency as well as whether the path, through which the subject *affects* the object, is relative or absolute. Consider the Functional Dependency for the `order` attribute, where the subject node is any `target` element whith explicitly specified order. The object then would be such a `target` element that occupies the natural position of the subject. The subject and object are related at the unit level (the common `unit` ancestor), therefore a relative regular expression needs to be applied. "Listing 4" illustrates implementation of this constraint using our convention.

Listing 4 - XLIFF Functional Dependency for the `order` attribute expressed in Schematron

```
<iso:rule context="xlf:target[@order]">
  <iso:let name="actual-pos" value="count
    (../preceding-sibling::xlf:segment|
    ../preceding-sibling::xlf:ignorable)+1"/>
  <iso:assert test="ancestor::xlf:unit//xlf:target
  [@order=$actual-pos])">
    Invalid use of order attribute.
  </iso:assert>
</iso:rule>
```

Schematron also introduces a phasing mechanism that can be used to group constraints in various phases so that rules are applied only when the relevant phase is *active* . Using this feature, alongside with `document()` function of XPath enables cross-document rules and *progressive validation* consequently. An XLIFF Processing Requirement, forbidding any changes in `skeleton` element (which stores the original data), is represented in Schematron in "Listing".

Listing 5 - XLIFF constraint on `skeleton` element expressed in Schematron

```
<iso:let name="original-xliff"
 value="document('before.xlf')"/>
  <iso:rule context="xlf:skeleton">
    <iso:assert test=
     "current()=$original-xliff//xlf:skeleton">
       Structure and content of skeleton element
       must not be changed.
  </iso:assert>
</iso:rule>
```

Schematron also offers some other useful features like variables (used in "Listing 5") and several tools to produce customized and informative error reports. The full adoption of XPath has made Schematron the most expressive schema language in the DSDL framework that is capable of handling the most complex Functional Dependencies, Keys and foreign Keys. Many of XLIFF constraints and Processing Requirements have been implemented in Schematron for the Advanced Validation feature to be avialable as of XLIFF 2.1.

3.2.3. NVDL

Namespace-based Validation Dispatching Language [18], NVDL, provides a schema language for selecting elements and attributes in specific namespaces within a document that are to be validated by a specified schema. NVDL is especially useful for muyltimodal XML vocabularies, such as XLIFF, that may contain different namespaces within a single document instance. NVDL can handle the task of mapping namespaces and assign appropriate schema artefacts for effcetive validation. The Advanced Validation feaure for XLIFF 2.1 and successors uses NVDL to compartmentalize the validation task for any potential XLIFF Document and run XML Schema and Schematron artefacts to validate static and dynamic usage of XLIFF Core and Module namespaces. Although Schematron rules can be embedded in both XML Schema and RelaxNG, it is generally advisable to use NVDL for this purpose, even in cases where the XML document declares only one namespace, as the former approach would require additional extraction and processing where NVDL is supported by various tools and libraries and provides simpler syntax for the task.

4. Conclusion

In this paper we reviewed generalized forms of XML Integrity Constraint types used to represent a data model in XML. We demonstrated, on the examples from the XLIFF industry vocabulary, that various types of Keys, foreign Keys and Functional Dependencies can be of different complexity depending on how advanced the required regular expressions are. We then explored a number of XML schema languages - focusing on the DSDL framework - in terms of their capacity to target different types of XML constraints and functional dependencies. The comparison revealed that Schematron, mainly due to its full adoption of XPath, can provide the highest expresivity for all types of constraints and fucntional dependencies among the tested schema languages. It is an industry proven best practice to validate the static structure of XML instances first, e.g. using RelaxNG or XML Schema, and to apply other advanced constraints or functional dependencies only afterwards, programmatically or by appropriate advanced schema languages, as paths towards the tested nodes must be established first and only then can be examined against any such advanced constraints. We also provided a convention to simplify and optimize defining Schematron rules through the concept of *subject/object* of Functional Dependencies.

Although the DSDL multipart standard (especially Schematron and NVDL) has resolved many issues in the XML validation domain, some of its aspects could be still improved. For instance, Schematron leaves implementation of some of its features, optional for processors, which significantly affects the functionality when using different processors or invoking Schematron

rules from an NVDL schema. These features usually presented via attributes like `subject, role` would generally enhance the error reporting, if more widely adopted.

Applying the methods presented in this paper to other industry vocabularies than XLIFF is proposed as future work. Similar investigations of DSDL applications on various vocabularies would provide a valuable set of artefacts for further theoretical research and study of XML Integrity Constraints and Functional Dependencies on the basis of emerging industry needs.

Bibliography

[1] *T. Bray, J. Paoli and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C, Nov. 2008..*

[2] *T. Bray, J. Paoli and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 W3C, 1998..*

[3] *T. Comerford, D. Filip, R.M. Raya and Y. Savourel. XLIFF Version 2.0, OASIS Standard, OASIS, 2014.*

[4] *H. Thompson et al. XML Schema, Part 2: Datatypes. W3C Recommendation, Oct. 2004. .*

[5] *International Standards Organization, (2001). ISO/IEC JTC 1/SC 34, DSDL Part 0, Overview. ISO..*

[6] *S. Saadatfar and D. Filip, "Advanced Validation Techniques for XLIFF 2," Localisation Focus, vol. 14, no. 1, pp. 43–50, 2014..*

[7] *P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan, "Keys for XML," Computer Networks, vol. 39, no. 5, pp. 473–487, 2002..*

[8] *M. Arenas, W. Fan and L. Libkin, On verifying consistency of XML specifications. In PODS, 2002..*

[9] *M. Vincent, J. Liu, and C. Liu, "Strong functional dependencies and their application to normal forms in XML," TODS, vol. 29, pp. 445–462, 2004..*

[10] *A. Deutsch and V. Tannen. MARS: A system for publishing XML from mixed and redundant storage. In VLDB, 2003..*

[11] *M. Arenas and L. Libkin, "A Normal Form for XML Documents," ACM Transactions on Database Systems, vol. 29, no. 1, pp. 195–232, Mar. 2004..*

[12] *w. Fan. XML Constraints: Specificaion, Analysis and Aplications. In DEXA, 2005..*

[13] *H. Thompson et al. XML Schema. W3C Recommendation, Oct. 2004..*

[14] *J. Clark and S. DeRose. XML Path Language (XPath). W3C Recommendation, Sep. 2015..*

[15] *International Standards Organization, (2003). ISO/IEC 19757-2:2003(E), DSDL Part 2, Regular-grammar-based validation — RELAX NG. ISO..*

[16] *International Standards Organization, (2004a). ISO/IEC 19757-3, DSDL Part 3: Rule-Based Validation — Schematron. ISO..*

[17] *E. van der Vlist, RELAXNG. O'Reilly Media, Inc., 2003..*

[18] *International Standards Organization, (2004b). ISO/IEC 19757-4, DSDL Part 4: Namespace-based Validation Dispatching Language— NVDL. ISO..*