Advanced Validation Techniques for XLIFF 2

Soroush Saadatfar and David Filip
Localisation Research Centre, CnGL, University of Limerick, Ireland
soroush.saadatfar@ul.ie, david.filip@ul.ie

Abstract
This paper aims to present an overview of validation issues for the XML Localization Interchange File Format (XLIFF), while XLIFF 1.2 validation is mentioned, the paper concentrates on validation of XLIFF 2.0 and successors (XLIFF 2). The goal is to propose an optimal set of DSDL (Document Schema Definition Languages) (Brown 2010) compliant methods to provide a set of XLIFF TC (and in due course OASIS and ISO) guaranteed standardized machine readable artefacts that would support validation of all XLIFF 2 normative statements. The discussed methods include XSD files, proposed possible schemas in the Relax NG schema language, Schematron rules, and finally a master NVDL (Namespace-based Validation Dispatching Language) file that brings all the necessary and guaranteed methods together. Development of a lightweight demonstrator RESTful web service that wraps all the proposed DSDL methods in one validator platform is also discussed and foreshadowed.

Keywords: XLIFF, validation, Relax NG, XSD, Schematron, NVDL, DSDL, XML, RESTful, web-service, web service

1. Introduction
According to the W3C definition, Extensible Markup Language (XML) is designed to describe data by providing a flexible text format derived from SGML (Standard Generalized Markup Language) (Bray et al. Eds. 2008). As XML is a software- and hardware-independent format for carrying information, it has become an important tool for information exchange among applications. XML is meant to be easily understood by humans, yet – at the same time – machine-readable (w3schools 2014).

XML provides a minimum set of rules for creating user-defined tags to describe the stored and exchanged data. Therefore, XML – as a metalanguage – provides logical space for creating XML vocabularies. In turn, instances of XML vocabularies need to be validated against a specific vocabulary’s schema. An XML schema is a document, which describes the structure of the document and indicates the order of elements with admissible and mandatory attributes and their values. A number of schema languages are available, which can check the XML instances against a variety of rules; from very basic ones – such as hierarchical structure or order of elements –, to complicated conditions, restrictions, and even admissible states in a workflow progression.

XSD (XML Schema Definition) (Thompson et. al. Eds. 2004) is the most popular schema language that is being used widely by XML consumers. In spite of its popularity, XSD is not expressive enough for XLIFF 2 and is not able to target many constraints including complex and advanced rules. Although, the latest version of XSD, 1.1 (Gao et. al. Eds. 2012), can handle many issues omitted in XSD 1.0 (Delima et. al. 2009), it is not broadly implemented and used (Quin et. al. 2012).

Relax NG schema language (Clark et. al. Eds. 2001) can be considered the next step in XML validation. It provides more possibilities for describing complex conditions than XSD. In addition, Relax NG is easier to learn and its expressions and syntax are more intuitive and user-friendly (Quin et. al. 2009; Vlist 2011). This schema has the potential to provide validation solutions to some of the XLIFF 2 constraints that XSD 1.0 cannot express.

Finally, the most powerful and expressive schema language for XML validation is Schematron (Jelliffe Ed. 2002). It can define the most complicated rules and constraints of XML standards and vocabularies. Schematron also provides room for delivering user-defined diagnostics and customized error messages, which can be enriched by detailed information about the objects that failed to comply with any of the validation rules.

All of the mentioned validation techniques are gathered and together make DSDL (Document
Schema Definition Languages). As a multipart ISO standard, DSDL defines a set of schema languages for XML validation (Brown 2010).

2. XLIFF Validation

XLIFF 2.0, as an OASIS standard, is presented in the specification created by the Technical Committee (Comerford, T., Filip, D., Raya, R.M., Savourel, Y.; Eds., 2014). XLIFF contains a large variety of different constraints and rules, which an XLIFF instance must not violate in order to be valid, it also addresses various application conformance targets with its processing requirements. Because of varied requirements for validation expressivity, different techniques for automated validation need to be used to cover the specification in full.

There are a number of tools already developed and available for the purpose of validating different versions of XLIFF. The first tool that addresses XLIFF validation beyond XSD was designed for XLIFF 1.2, it is the XLIFFChecker (Raya, 2012). We are however primarily concerned with XLIFF validation, because XLIFF 2.0 does provide, in its prose, specification statements that allow for more advanced DSDL validation.

The third party validators currently available for the 2.0 edition are XLIFF 2.0 XMarker Checker (Schnabel) and Okapi-lynx (Savourel). Unlike the approach used in this paper, these validators are not DSDL-based, except the basic XSD validation. Their validation methods that go beyond a simple XSD check are implementation dependent in the above. On the plus side, both of the cited XLIFF 2 validators do provide validation for XLIFF 2.0 fragment identifiers, which goes beyond the scope of this paper. This DSDL based approach addresses the wellformedness of an XLIIF fragment identifier only as far as it is required to check attributes of the URI or IRI type.

The main target of this paper is to identify and elaborate validation methods and artefacts on the basis of XML standardised (implementation independent) methods of validation, i.e. DSDL schema languages. DSDL artefacts that cover automated validation for a maximal subset of a XLIFF 2 specification are suitable for becoming XLIFF TC deliverables (as part of the multipart standard product according to OASIS definitions) in XLIFF 2.1 and successor editions.

XSD schema – the first part of DSDL – for XLIFF 2.0 core is already provided by as a part of the XLIFF 2.0 multipart OASIS standard. However, this schema is able to provide only very basic validation and moreover, many patterns that pass XSD validation are in fact violating normative statements of the prose specification. For instance, many attributes which are encoded as optional in the XSD schema, may be in fact required or forbidden conditionally, dependent on specific values of other attributes, availability of specific content or elements etc. These advanced constraints are explained in detail further in this section, as we are tackling them with other DSDL methods.

As the first step, the Relax NG schema for XLIFF 2.0 core was developed (Saadatfar 2014). Relax NG schema for XLIFF validation allows one to conduct more detailed document checks than the original XSD. For instance, Listing 1 illustrates how it is possible to handle the constraint for the subType attribute (used in some of XLIFF inline elements).

The XLIFF specification states:

If the attribute subType is used, the attribute type must be specified as well.

The schema is defining two valid cases (<choice> element allows either of its children to be valid); first if both type and subType attributes are present in the element, and the second case where type appears only. <ref> elements will describe the content of each declared node later in the schema.

Despite its advantages, Relax NG is not expressive enough to describe all the normative content of an XLIFF 2 specification. For instance, it does not

```xml
<optional>
  <choice>
    <attribute name="type">
      <ref name="atttype"/>
    </attribute>
    <attribute name="subType">
      <ref name="attsType"/>
    </attribute>
  </choice>
  <attribute name="type">
    <ref name="atttype"/>
  </attribute>
</optional>

Listing 1: subType dependency in Relax NG
support default values for attributes directly, DTD compatibility annotations must be embedded for this purpose. Definition of some rules may turn into extremely long code which is very hard to read and maintain. For instance, according to the specification, the trgLang attribute of <xlf> element is required if and only if the XLIFF document contains a <target> element. This statement could be targeted in Relax NG by defining two patterns for <xlf>. This approach would duplicate most of the schema and therefore does not present a practical solution.

For such complex constraints, Schematron offers a suitable solution, the rule-based validation. Listing 2 illustrates the rule for the earlier mentioned statement;

```xml
<iso:rule context="xlf:target"
  see="http://docs.oasis-open.org/xliff/xliff-core/v2.0/os/xliff-core-v2.0-os.html#xlf"/>
  <iso:let name="parent-name" value="name(..)/">
  <iso:let name="unit-id" value="ancestor::xlf:unit/@id"/>
  <iso:assert test="ancestor::xlf:target/@trgLang"
    diagnostics="general spec-quote">
    The XLIFF document contains a &lt;iso:name/&gt; element, but the
    trgLang attribute of &lt;xlf&gt; element is missing.
  </iso:assert>
</iso:rule>

Listing 2: trgLang check in Schematron
```

Schematron uses XPath language to access elements inside documents. This approach helps to track errors at all levels.

The rule in listing 3 first seeks <target> elements inside the document (as context attribute defines) and then applies the constraint; i.e. if the file does not contain any <target>, the rule will be ignored as it is not compulsory anymore. After the element was found, at the <assert> element the validation test is conducted (assertion). The expression inside the test attribute will return “true” if <xlf> element has trgLang attribute and otherwise, an error will be raised. In the case of an error, the message placed inside <assert> will be shown as well as a link (inside see attribute) to the XLIFF specification. The <name> element returns the name of the picked node (target in this case). This element becomes very helpful when dealing with several nodes at the same time and it will be discussed later. Finally, beside all that, thanks to <let> elements, which are variables in Schematron, we can save additional information about the error. In this example, we are retrieving the id attribute of the enclosing <unit> element as well as the immediate parent of the <target> element which caused the error (it can be either <segment> or
Listing 3: skeleton “pseudo-solution”, Relax NG comparison between the Relax NG and Schematron approaches is the `<skeleton>` element’s constraint. The `<skeleton>` element must contain either an href attribute and no text (empty) or text without the attribute. Listing 3 shows how this issue seems to have been resolved in the Relax NG schema:

Schema again is giving two valid scenarios; either the element is empty and has the attribute, or it is not empty and no attributes are present.

However, both of the following invalid patterns will pass the validation against this schema; `<skeleton/>` and `<skeleton>`
</skeleton>. The point is that an empty element or a white-space is still considered text in Relax NG, which allows the mentioned patterns to pass although in fact not valid according the normative prose specification.

Listing 3 shows how this constraint can be addressed by a Schematron rule defining all the possible cases.

This rule perfectly targets the `<skeleton>` constraint by describing it within the expression inside the test attribute.

Schematron can cover all remaining XLIFF normative statements including the most important and sophisticated ones, i.e. uniqueness of id attribute values among elements in different uniqueness scopes of an XLIFF document.

ID-uniqueness in XLIFF 2.0 comes in three distinct levels of complexity. First, when children elements (i.e. siblings to each other) must have unique id values within their parent element. This is the case of `<file>` elements within the root `<xliFF>` element. The second level then requires elements grandchildren to have unique identifiers within the entire grandparent scope: the case of `<data>` elements within `<unit>` elements. These first two types are fairly easy to handle, so we will discuss in detail only the third and most complex one.

Unit level in XLIFF is a logical container for a relatively independent portion of content. The extracted text along with codes, represented by XLIFF inline elements, are stored in `<source>` elements, and later the translations will be added to `<target>` elements, both grandchildren of `<unit>`; each pair of the `<source>` and `<target>` siblings is placed within a `<segment>` or an `<ignorable>` element in order to capture segmentation of the content unit.

The constraints and other provisions for the unit level are provided and explained in many different places in the specification, and moreover many of them are conditional to specific governing attribute values.

Listing 4: `<skeleton>` solution, Schematron
The most general, though, is that any id attribute must be unique among all <segment>, <ignore>, <mrk>, <sm>, <pc>, <sc>, <ec>, or <ph> elements within the enclosing <unit> element. But, the inline elements inside <target> must use the duplicate id values of their corresponding inline elements in the <source> sibling, as long as such corresponding elements do exist, given the conditional logic of specific governing attributes.

The trickiest validity conditions to check occur here. The translated text of a content unit may be presented in a different order, compared to the original text. This means that the <target> sibling content does not need to logically or linguistically correspond to the content of its <source> sibling, provided that a different order has been explicitly specifies through the order attribute that is optional at the <target> element.

The order attribute defines the actual position of its <target> element in the sequence of the unit content. Therefore the corresponding inline elements must be searched for inside the relevant <source> that can be a “cousin” (same grandparent, different parent) of the <target>. The value of the order attribute is a positive integer between 1 and N, where N is the number of <segment> and <ignore> elements within the unit. Finally, inline elements that have their canDelete attribute set to ‘no’, must appear in the corresponding <target>, but other elements may be deleted, because the default value of the canDelete attribute is ‘yes’.

As the first step for this task, we will check the outermost elements of the unit; <segment> and <ignore>. The id is optional for these elements, so only those with id attributes will be selected and checked for duplication within the unit. Listing 5 demonstrates this step.

Here, the variables pick the elements which follow the node but still within the same <unit>, its preceding elements and all its descendants. There must not be any node matching this pattern.

As the second step, all inline elements inside <source> elements will be checked for duplication among themselves, which is shown in the Listing 6.

For the demonstration purposes, the rules were shortened for inclusion in this paper; the actual rules provided on the XLIFF TC SVN repository provide detailed error messages that are rigorously based on the normative statements of the specification.

From now on (for the third ID-uniqueness validation step), we are going to assume that the validated XLIFF instance is in the final stage of validation (going to merge back into the original format as the next workflow stage), which means that all rules that apply to the <target> element are enforced, while this element is strictly speaking optional (from the specifications point of view) at any other stage.

This third step consists itself of sub-steps. First, we check whether values (explicit and implicit) of the order attributes are legal (between 1 and N) and then for their uniqueness using the method explained earlier. Then, all inline elements inside the <target> will be verified against all of other inline elements except themselves; they are allowed to have one element duplicating their id in the corresponding <source>, we will look for them a bit later.
Listing 6: Inline id uniqueness check, Schematron

```
  <iso:let name="id-attribute" value="@id"/>
  <iso:let name="parent-unit" value="ancestor::xlf:unit[@id]"/>
  <iso:let name="ph-check" value="preceding::xlf:ph[ancestor::xlf:unit[@id=$parent-unit] and ancestor::xlf:source][@id=$id-attribute]"/>
  <iso:let name="pc-check" value="preceding::xlf:pc[ancestor::xlf:unit[@id=$parent-unit] and ancestor::xlf:source][@id=$id-attribute]"/>
  ...
  <iso:let name="descendant-check" value="descendant::xlf:*[@id=$id-attribute]"/>
  <iso:assert test="count($ph-check)=0">id duplication found!
  </iso:assert>
</iso:rule>
```

Listing 7: order constraints described in Schematron

```
<iso:rule context="xlf:unit">
  <iso:let name="id" value="@id"/>
  <iso:let name="seg-ig-number" value="count(descendant::xlf:segment | descendant::xlf:ignorable)"/>
  <iso:let name="invalid" value="descendant::xlf:target[@order>$seg-ig-number]"/>
  <iso:let name="disordered" value="descendant::xlf:target[@order!-count(preceding::xlf:segment[ancestor::xlf:unit[@id=$id]] | preceding::xlf:ignorable[ancestor::xlf:unit[@id=$id]])+1]@order]/order"/>

  <iso:let name="tester" value="descendant::xlf:target[count(preceding::xlf:segment[ancestor::xlf:unit[@id=$id]] | preceding::xlf:ignorable[ancestor::xlf:unit[@id=$id]])+1=$disordered]@order]"/>
  <iso:assert test="count($invalid)=0">
    The value used for order attribute of &lt;target&gt;: element is greater than number of &lt;segment&gt; and &lt;ignorable&gt;: elements within the enclosing &lt;unit&gt;: element. Number of invalid &lt;target&gt;: elements found: '<iso:value-of select="count($invalid)"/>
  </iso:assert>
  <iso:assert test="count($tester)=count($disordered)">
    Those &lt;target&gt;: elements whose position is taken by order attribute of other &lt;target&gt;: element are missing or do not contain order attribute.
  </iso:assert>
</iso:rule>
```
The next sub-step of the third step is to check if the order attributes are used correctly within units. If a <target> element’s natural order in the unit sequence has been overridden by explicitly specifying a different position via the order attribute, the <target>, which had occurred at the natural position corresponding to the former <target> element’s explicitly set order, the latter has been “displaced”, and therefore has to use its own order attribute to specify another available position within the unit. And so forth, until all target segments have explicitly set available positions, or otherwise occur on positions with their natural order not “displaced” by any other explicitly set order attributes. Listing 7 demonstrates the rule for validating this.

This rule will first identify those order attributes that use an illegal positive integer. The natural positions within the sequence are given by the ordinal numbers designating the positions of the <source> elements within the given unit, so the rule searches for <target> elements, which are not at the natural position (the order attribute may actually match the actual position), and saves the values, then checks for <target> elements at those positions that in turn must have an explicitly set order attribute. This loop continues until the rule verifies that all of the <target> elements are ordered properly or until a violation is identified.

After the corresponding <source> elements have been identified, previously described methods are used to match legal inline ID duplications between target and source content.

3. Web Service

To implement Relax NG and XSD validation in Java, the Java library developed by the Thai Open Source Software Center, Jing (Clark, 2008) can be used. This library, as well as other standard Java libraries for XML processing, provides sufficient tools to develop a Java programme, which can validate XML documents against the given schemas; XSD or RNG by user choice. This programme also returns a list of identified errors in case of a failed validation.

Based on the above, a RESTful web service was developed that consumes (receives) an *.xlf file within an HTTP POST request and produces (returns) the validation result (along with the error report, if any errors or warnings occurred) in JSON format. This web service can be called to validate XLIFF instances against the XSD or RNG schemas or both. The functionality of the web service is illustrated with the screenshots below, with an XLIFF file passing the validation and another failing.

Integrating the Schematron rules is planned as the next step and will give more informative messages in case of a validation failure.

4. Next Steps towards the full advanced validation of XLIFF 2

By using the above demonstrated Schematron methods and techniques, the XLIFF core can be described in full, and hence fully validated with DSDL methods only. The expressivity of Schematron (together with XSD) is sufficient to guarantee the full validation of the XLIFF core and modules including the fragment identifiers. All DSDL based methods of XLIFF validation originating from this research are being proposed as the technical solution (normative
artefacts to become part of the multipart standard) for the Advanced Validation feature of XLIFF 2.1 via the OASIS XLIFF TC.

To be able to check arbitrary XLIFF files including modules (and extensions), against all provided schemas (schems for extensions can be also integrated if provided), constraints and processing requirement, and in order to handle dynamic validation, NVDL (Namespace-based Validation Dispatching Language) needs to be used. This will provide a suitable mapping tool for the advanced validation of different parts of XLIFF document against different namespaces, using mixed DSDL methods and schema languages. Eventually, also NVDL based techniques will be embedded in the validation web service that is being developed based on this research. The web service, however, is not expected to achieve any normative status through the XLIFF TC due to being implementation dependent. Nevertheless, the web service should be made publicly available by the CNGL within the first half of 2015 in order to provide easy access to the advanced validation methods for the standard’s end users.

References


w3schools XML Tutorial [online], available: http://www.w3schools.com/xml/ [accessed 19 Dec 2014].