SYSTRAN Translation Stylesheets: Machine Translation driven by XSLT

Pierre Senellart, SYSTRAN S.A.INRIA Futurs

Jean Senellart, SYSTRAN S.A.

Abstract

We present the SYSTRAN Translation Stylesheets (STS) used by the SYSTRAN machine translation engines that power the most prominent online translation services on the Web today (including Google, Yahoo!, AltaVista's BabelFish, and AOL). SYSTRAN develops and markets the world's most widely used machine translation software which powers products and solutions for the desktop, network infrastructures and the Internet that facilitate communication in 40 language combinations and in 20 specialized domains. The choice of leading global corporations, search engines and governments, SYSTRAN's software is applied across diverse best-practice solutions for intra-company communications, content management, online customer support, eCommerce, email systems, chat, and more. SYSTRAN's expertise spans over three decades of building customized translation solutions through open and robust architectures.

XSL Transformation stylesheets are usually used to transform a document described in an XML formalism into another XML formalism, to modify an XML document, or to publish content stored into an XML document to a publishing format (XSL-FO, (X)HTML…). SYSTRAN Translation Stylesheets (STS) use XSLT to drive and control the machine translation of XML documents (native XML document formats or XML representations — such as XLIFF — of other kinds of document formats).

STS do not only provide a simple way to indicate which part of the document text is to be translated, but also enables the fine-tuning of translation, especially by using the structure of the document to help disambiguate natural language semantics and determine proper context. For instance, the phrase “Access From Front Door” is to be analyzed as “The access from front door” within a title, and as “Do access (something) from front door” in the text body. In that case, the STS would pass a `title` option to the translation engine. The stylesheet can activate specialized domain dictionaries for some parts of the document and can mark some expressions as not to be translated, in the same manner.

Another key application of STS is to consider machine translation as part of the authoring and publishing process: source documents can be annotated with natural language markup produced by the author, markup which will be processed by STS to improve the quality of translation, the gateway to the automatic publishing of a multilingual website from a monolingual (annotated) source. This composition, inside the publishing process, is a real breakthrough for Web content translation. Traditionally, machine translation is applied after publishing and does not have access to the original structure of the document, but only to its HTML representation.

The mechanism is implemented through XSLT extension functions. In particular, the stylesheet uses a `systran:translate` function to translate an XML fragment, and `systran:getValue`/`systran:pushValue`/`systran:popValue` functions for consulting and for setting linguistics options in the translation engine. Proper management of character properties is also provided so that, for instance, the translation of a phrase in bold font will appear in bold font, even if the phrase has moved within the translated sentence.

This process is highly customizable by the addition of new templates into the stylesheets. Because the translation is driven by the document structure, it is easier to leverage this structure during the translation process and to keep it in the translated document, than with traditional document filters, which process the entire document linearly.

STS are part of SYSTRAN's commercial version 5.0 product line and are also used for and by specific corporate-customers.

Table of Contents

Introduction

The continuous expansion of the content available on the Web, combined with the internationalization of exchanges that the Web allows, increase the need for multilingual publishing. However, it is impossible to handle the voluminous amounts of short-life and specialized content (such as technical support documentation, news feeds…) in the traditional human localization workflow. Alternatively, machine translation (MT) provides on-demand translation with relatively low cost. MT usage has become common at different customization levels: from “raw” translation available for free on the Web's most popular portals, to highly customized translation for specific applications.

One facet of the “how to achieve quality translation” dilemma is how MT is integrated in the “multilingual publishing” workflow. Traditionally, this is done at a very late stage of the process, where MT engines need to cope with poorly structured content. We describe in this paper how the use of SYSTRAN Translation Stylesheets (STS) leverages document structure in an MT process: XSLT exposes the semantics underlying the document structure to the MT engine, subsequently improving the accuracy of MT rule selection, as well as facilitating the interaction with the author.

We first provide a description of the SYSTRAN Translation Stylesheet and compare its advantages over classical document filters. We will also show how this new technology naturally introduces MT in the multilingual authoring and publishing workflow.

The SYSTRAN Translation Stylesheet

To understand the way the Translation Stylesheet operates, one needs first to understand how “traditional” document filtering functions and the means through which the translation process can be parameterized. The next section breifly describes this information, and is followed by how Translation Stylesheets are defined. We conclude this section with short examples that illustrate the power of the Translation Stylesheet driving the translation process.

Traditional document filters and translation customization

With regard to MT engines, “filters” are in charge of extracting sentences from the source document and for rebuilding a target document with translated sentences. Once extracted, the sentences are sequentially processed “out of context” — in particular, it is impossible to know if a given sentence belongs to a specific part of the original document (as represented in Figure 1, “Traditional document filter: sentences are extracted from the source document, translated with unique global parameters, and re-injected in the target document”). Therefore, no local parameterization can be performed.

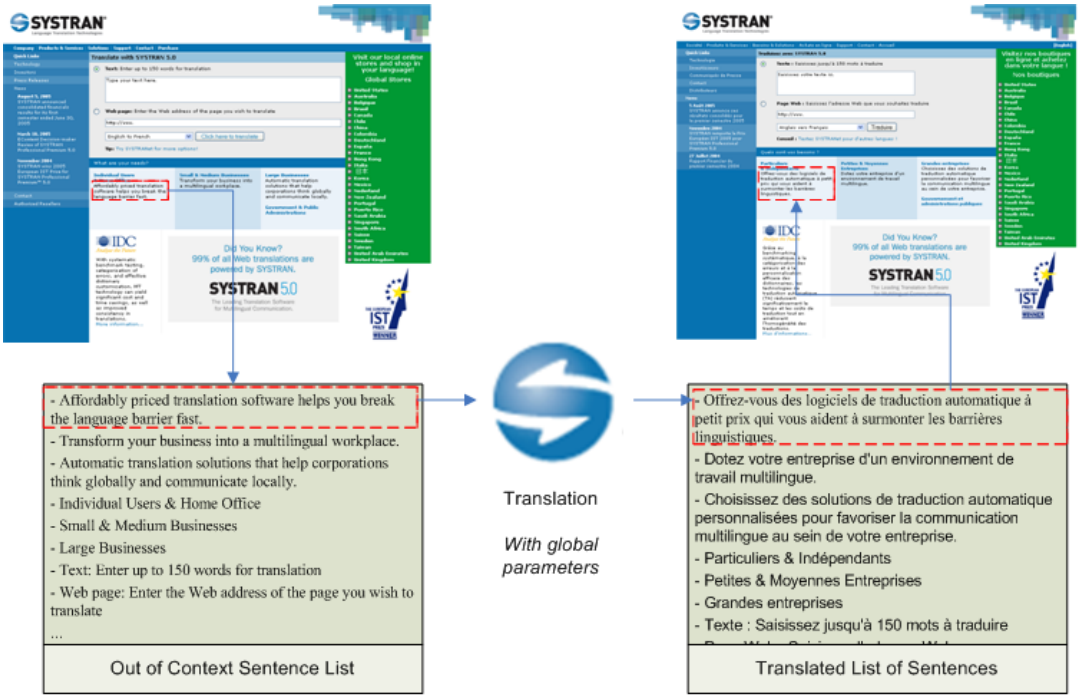

Figure 1. Traditional document filter: sentences are extracted from the source document, translated with unique global parameters, and re-injected in the target document

In such a workflow, the panel of available *translation parameters*, i.e. the different means through which one can customize the translation, include the following:

- selection of user-defined dictionaries — changes the meaning of an expression according to a specific terminology.
- modification of linguistic parameters — for instance, the way the imperative should be translated, the gender of a specific pronoun… Such options are represented by name/value pairs.

Additionally, the user can define DNT (*Do Not Translate*) expressions but this requires specific markup on the input text by the user.

It is important to note that the translation parameters in traditional document filters are *global* — each sentence of the corpus is translated with the same parameters since there is no way to differentiate sentences once they are extracted. Primary benefits from Translation Stylesheets are to use the XSLT language in order to modify locally translation options, to define dynamically DNT expressions and to introduce new types of parameterization.

STS technical description

Extension functions

A SYSTRAN Translation Stylesheet (STS) is a standard XSLT 1.0 stylesheet [1] which uses the following extension functions (in a SYSTRAN namespace):

[1] Actually, the EXSLT [EXSLT] `node-set` extension function is needed; this requirement would be removed when switching to XSLT 2.0.

node-set translate(*node-set*)

This function returns a translation of the node-set. The node-set must be a set of `par` (for *paragraph*) trees verifying the following DTD (attributes are not shown):

```
<!ELEMENT par (#PCDATA|mark|typo|tag)*>
<!ELEMENT mark (#PCDATA|mark|typo|tag)*>
<!ELEMENT typo (#PCDATA|mark|typo|tag)*>
<!ELEMENT tag ANY>
```

- `mark` tags are used to provide additional local information (in input) to or to get additional feedback (in output) from translation engines.
- `typo` tags are used to indicate local character properties (e.g. bold font, hyperlinks…) so that these properties are reinserted in the translation result, at the proper location (following reordering of words).
- Everything inside a `tag` tag remains as is in the translation result.
- Every text content not inside a `tag` node is translated.

Example 1, “Sample Source Tree” displays a sample tree passed to the `translate` function, and Example 2, “Translation from English to French of tree from Example 1, “Sample Source Tree” ”, the corresponding translation from English to French. Note that the ambiguous “boot” is not translated as a shoe, but as a technical term due to the `mark` set on the word.

Example 1. Sample Source Tree

```
<par>
  <typo type="bold">Quick</typo>
  <mark
     action="set" type="domain"
     value="information_technology">boot</mark>!

  <tag>This <zorglub>will not be</zorglub> translated.</tag>
</par>
```

Example 2. Translation from English to French of tree from Example 1, “Sample Source Tree”

```
<par>
  Démarrage <typo type="bold">rapide</typo> !
  <tag>This <foobar>will not be</foobar> translated.</tag>
</par>
```

string getValue(*string*)

Queries the value of the translation engine options. These can be general options (like the current source and target languages), global linguistic options (such as if “you” in English should be translated as informal “tu” or polite “vous” in French) or even stylesheet-specific options set by the user.

void pushValue(*string* , *string*)

Sends local options to the translation engine, such as requiring that, inside a particular tag, every noun/verb ambiguity should be resolved as noun.

void popValue()

popValue is the counterpart of pushValue: a stack of options is maintained in the translation engine; the latest element added to the stack is removed on the call of popValue. This mechanism allows for a very general management of the span life of options sent to the translation engine.

Default behavior and utility templates

To expedite the process of writing STS, utility templates are defined in a tools.xsl imported stylesheet. Additionally, tools.xsl defines the needed templates in order to use the following approach for writing STS stylesheets, by importing tools.xsl and benefiting of the natural *cascading* mechanism of XSLT. The next section presents present illustrations of these somewhat abstract rules.

- By default, nodes are browsed in document order and textual nodes are preserved and not translated.
- Nodes whose direct content should be translated should call a translate_par template (or one of its variants).
- Inline elements (those which are descendants of nodes whose content will be translated) must define templates with the *preprocess* mode which, should produce mark, typo, tag according to the wanted behavior.
- *postprocess* mode should be defined for mark, typo, tag if some processing needs to be done to recover original structures.

Examples

Generic STS

Example 3, “Generic STS for an XML data format with no mixed elements” shows a very simple STS which translates every text node of the XML document as a separate paragraph. This is enough to translate an XML document with no mixed elements. This stylesheet can be refined to handle some kinds of tags in a special way.

Example 3. Generic STS for an XML data format with no mixed elements

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:import href="tools.xsl"/>

<xsl:template match="text()">
  <xsl:call-template name="translate_par">
    <xsl:with-param name="source" select="."/>
  </xsl:call-template>
</xsl:template>

</xsl:stylesheet>
```

XHTML STS

Example 4, “Partial STS for XHTML” shows a simplified and partial STS for translating XHMTL files. The following features are illustrated:

- `b` and `strong` tags are converted to `typo` tags during preprocess. Bold character properties are converted back to `strong` tags. There is an additional way to restore the exact original tags, which is not detailed here.
- The content of `p`, `h1`, `h2`… is translated.
- An option is sent to the translation engine for a special handling of `title` tags.
- `alt` and `title` are translated, except in special cases (we have the full expressive power of XPath and XSLT to refine the conditions).
- For block quotations, both the source and the translation are kept in document translation.

Note that this illustration only scratches the surface of possibilities able to be expressed via STS.

Example 4. Partial STS for XHTML

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:h="http://www.w3.org/1999/xhtml"
  xmlns:systran="http://www.systransoft.com/xslfunctions"
  extension-element-prefixes="systran">

<xsl:import href="tools.xsl"/>

<!-- Character properties -->
<xsl:template match="h:b|h:strong" mode="preprocess">
  <typo bold="1">
    <xsl:apply-templates mode="preprocess"/>
  </typo>
</xsl:template>

<xsl:template match="typo[@bold='1']" mode="postprocess">
  <xsl:element name="strong"
                          namespace="http://www.w3.org/1999/xhtml">
    <xsl:apply-templates mode="postprocess"/>
  </xsl:element>
</xsl:template>

<!-- Tags with content to translate -->
<xsl:template match="h:p|h:li|h:h1|h:h2|h:h3|h:h4|h:h5|h:h6">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:call-template name="translate_par">
      <xsl:with-param name="source" select="node()"/>
    </xsl:call-template>
  </xsl:copy>
</xsl:template>

<!-- Translate title, resolving verb/nouns ambiguities to nouns -->
<xsl:template match="title">
  <xsl:value-of select="systran:pushValue('TITLE','1')" />
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:call-template name="translate_par">
      <xsl:with-param name="source" select="text()"/>
    </xsl:call-template>
  </xsl:copy>
  <xsl:value-of select="systran:popValue()" />
</xsl:template>

<!-- @title and @alt should be translated… -->
<xsl:template match="@title|@alt">
  <xsl:call-template name="translate_par">
```

```
    <xsl:with-param name="source" select="."/>
  </xsl:call-template>
</xsl:template>

<!-- … except for acronyms or abbreviations -->
<xsl:template match="@title[name(..)='abbr' or name(..)='acronym']">
  <xsl:copy/>
</xsl:template>

<!-- Keep both source and translation in result for blockquote -->
<xsl:template match="blockquote">
    <xsl:copy-of select="." />
    <xsl:copy>
      <xsl:apply-templates select="@*"/>
      <xsl:call-template name="translate_par">
        <xsl:with-param name="source" select="node()"/>
      </xsl:call-template>
    </xsl:copy>
 </xsl:template>

</xsl:stylesheet>
```

Multilingual publishing workflows overview

The millions of web pages machine translated[2] per day through search engines result pages have the following characteristic: machine translation directly deals with the publishing format – most of the time poorly structured HTML – of a content that was not originally meant to be translated. Localization is triggered "on-demand" by the user and there is no connection between the publishing workflow and this localization workflow. The translation engine must deal with the publishing format and guess the structure beyond the format. A good example of this can be seen by the use of separators (like |, •…) on webpages – they are used sometimes as item separators, sometimes as formatting symbols, sometimes as list bullets… without any algorithmic way to differentiate these uses.

More over, the translation engine is using a "tolerant" (similar to *quirks* mode of common browsers) HTML filter to parse the content and to restore the translated text in the original format. But the formatting of the result can be imperfect since text length is not preserved and might break "visual adjustments" performed by stylesheets or manually by webmasters.

Finally, the user, using Machine Translation (MT) as a *gisting* tool, is most of the time not aware of the content of the page before translation; and he would not be able to parameterize the translation process accordingly. As a result of all of this, the translation quality is generally lower than possible — the translation process is too far from the document structure to deal with its semantic.

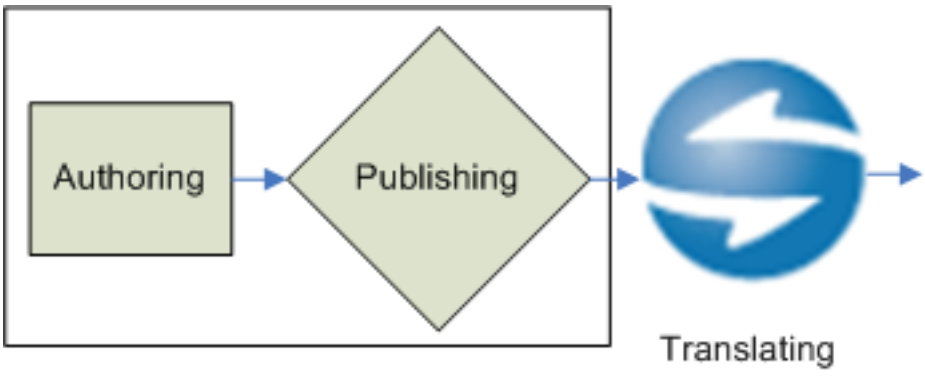

Figure 2. Traditional MT Publishing Workflow: Authoring, Publishing, MT Localization

[2] In early 2005, the estimated number of pages translated by SYSTRAN translation engines was about 20,000,000 pages a day.

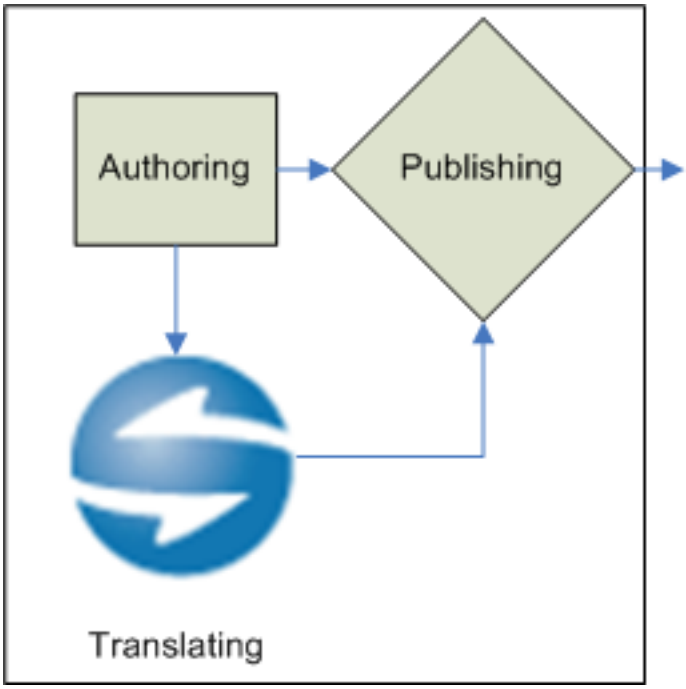

Figure 3. Better MT Publishing Workflow: Authoring, MT Localization, Publishing

Figure 2, “Traditional MT Publishing Workflow: Authoring, Publishing, MT Localization” represents this first usage. A direct improvement of this workflow is to introduce localization before publishing (see Figure 3, “Better MT Publishing Workflow: Authoring, MT Localization, Publishing”), on the authoring format (ideally, an XML format) with the following direct benefits:

- The content might be more structured and is not impacted with publishing effects.
- Localization is performed by the content provider, who is aware of the semantic domain of the content, and the translation process can be customized accordingly.

To achieve this, translation process must deal directly with the user own XML structure and turn document syntax into linguistic information used by the engine to provide a customized translation. This is performed through the use of Translation Stylesheet as described in previous section.

The workflow can be improved further by introducing a interaction between the author and the machine translation through the STS (see Figure 4, “Ideal MT Publishing Workflow: Authoring with MT in mind, MT Localization, Publishing”). In this workflow, the author is notified during the editing of the content, similarly to an active spellcheck, that some expressions are ambiguous, that some structure is too complex and might result in a bad translation, that some word is not known by the translation engine… The author can then choose to adjust the content or to annotate the document with linguistic markup which will be processed by the STS. This added information will improve the quality of the translation for any target language, without requiring the author to be familiar with this language.

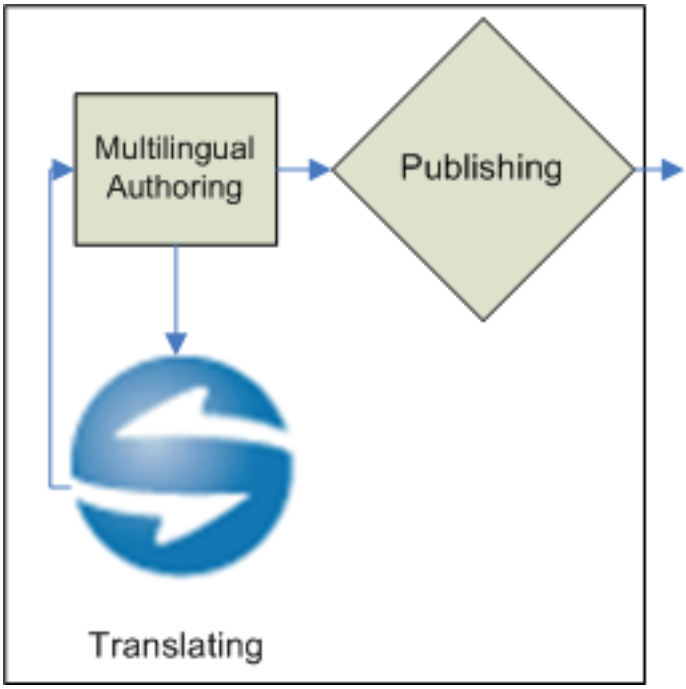

Figure 4. Ideal MT Publishing Workflow: Authoring with MT in mind, MT Localization, Publishing

Applying STS to structured but non XML content

The use of STS presented above is restricted to the translation of XML documents[3]. However, it would be nice to be able to handle non XML content (e.g. Microsoft Word, LaTeX, PDF, e-mails…) with all the benefits of STS, since these formats are still widely used, both for authoring and publishing. A way to achieve this is to have converters to and from an XML format. We present here how to use OASIS XLIFF [XLIFF] for this purpose. XLIFF is the XML Localization Interchange File Format designed by a group of software providers, localization service providers, and localization tools providers. This format allows to extract text content from any document format, together with normalized information about document structure and character properties. After localization, the original format can be reconstructed (*merge* process) based on the translated strings.

To get benefit from the STS and allows same level of translation fine-tuning that STS provide for translating XML format, the filter converting source format into XLIFF should preserve as much structure from the source document as possible. For instance, through the use of the XLIFF `group` tag, high level document formating such as tables or lists can be represented. Use of `bpt` tags can reflect character properties in the source format… The semantics of those tags is then described within an STS and allows the translation engine to deal with any type of document property. Figure 5, “STS translation of non XML document through XLIFF interchange format” describes the complete workflow.

[3]Actually, the use of libxml2/libxslt libraries [LIBXML] allows us to handle HTML documents with STS as well.

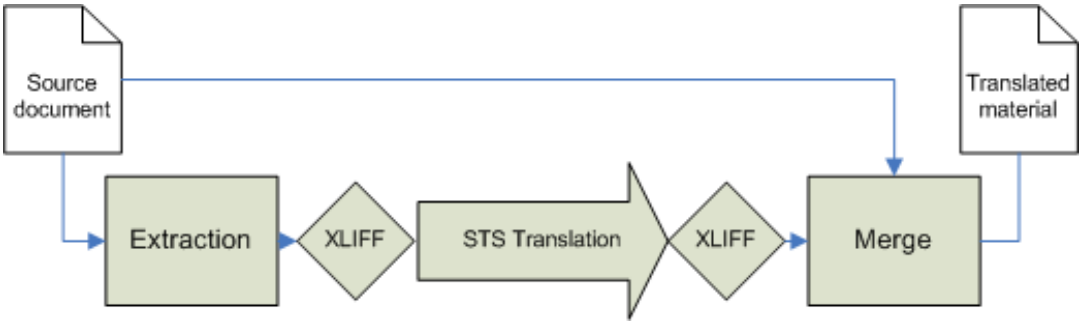

Figure 5. STS translation of non XML document through XLIFF interchange format

Conclusion

We have presented an innovative application of XSLT that is used to drive a complex external process with parameters: a Machine Translation system. This mechanism binds the deep structure of the source document to the translation engine options. This renders a correlation between the document syntax and its underlying linguistic structure. Conversely, one can enrich the document structure with linguistic information in order to improve the quality of machine translation results. In an optimal workflow, the author and the translation engine interact with a feedback - structure enrichment cycle.

This tool is the first step towards an authoring process that is cognizant of the increasing demands for multilingual content.

Bibliography

[EXSLT] *Extended XSLT*. http://www.exslt.org/.

[LIBXML] *The XML C parser and toolkit of Gnome*. Available under the MIT license at http:/ / www.xmlsoft.org/.

[XLIFF] *XML Localization Interchange File Format*. White-paper available at http://www.oasis-open.org/apps/group_public/download.php/3110/XLIFF-core-whitepaper_1.1-cs.pdf .