The Dos and Don’ts of XML document localization

Andrzej Zydroń
CTO XTM International

ABSTRACT

XML is now ubiquitous: from Microsoft Office to XHTML and Web Services it is at the core of electronic data communications. The separation of form and content, which is inherent within the concept of XML, makes XML documents easier to localize than those created with traditional proprietary text processing or composition systems.

Nevertheless, decisions made during the creation of the XML structure and authoring of documents can have a significant effect on the ease with which the source language text can be localized. For example, the inappropriate use of syntactical tools can have a profound effect on translatability and cost. It may even require complete re-authoring of documents in order to make them translatable.

This presentation highlights the potential pitfalls in XML document design regarding ease of translation and provides concrete guidance on how to avoid them.

1. Introduction

The adoption of XML as a standard for the storage, retrieval and delivery of information has meant that many enterprises have large corpora in this format. Very often information components in these corpora require translation. Normally, such enterprises have enjoyed all of the benefits of XML on the information creation side, but very often, fail to maximize all the benefits that XML based translation can provide.

The separation of form and content which is inherent within the concept of XML makes XML document easier to localize than traditional proprietary text processing or composition systems. Nevertheless decisions made during the creation of the XML structure and authoring of documents can have a significant effect on the ease with which the source language text can be localized into other languages. The difficulties introduced into XML documents through inappropriate use of syntactical tools can have a profound effect on translatability and cost. It may even require complete re-authoring of documents in order to make them translatable. This is worth noting as a very high proportion of XML documents are candidates for translation into other languages.

A key concept in the treatment of translatable text within XML documents is that of the "text unit". A text unit is defined as being the content of an XML element, or the subdivision thereof into recognizable sentences that are linguistically complete as far as translation is concerned.

2. Designing XML documents for translation

It is very important to consider the implications for localization when designing an XML document. Wrong decisions can cause considerable problems for the translation process thus increasing costs. All of the following examples assume that the text to be translated is to be extracted into an intermediate form such as XLIFF (XML Localization Interchange File Format). Anyone planning to provide an XML document directly to translators will soon be disabused of this idea after the first attempt. The intermediate format protects the original file format and guarantees that you get back an equivalent target language document to that of the original source. An additional concept which is important regarding the localization of XML documents is that of the 'inline' element. Inline elements are those that can exist within normal text (PCDATA - Parsable Character DATA). They do not cause a linguistic or structural break in the text being extracted, but are part of the PCDATA content.

The following is a list of guidelines based on (often bitter) experience. Most of the problems are caused by not following the fundamental principles of XML and good XML practice. It is nevertheless surprising how often you can come across instances of the following type of problem. Please note that this is not a proscriptive list, there may be special circumstances where the proposed rules may have to be broken:

2.1. Avoid the use of specially defined entity references

Although entity references can look like a 'slick' technique for substituting variable text such as a model name or feature in a publication, they can cause more problems than they resolve.

```
<para>Use a &tool; to release the catch.</para>
```

Example 1: Incorrect use of Entity References

Entities can cause the following problems:

- **Grammatical difficulties**

  If the entity represents a noun or noun phrase this will potentially cause serious problems for languages in which nouns are strongly inflected, such as many Slavonic and Germanic languages. What appears fine as an entity substitution in English can cause insurmountable problems in inflected languages.

  The solution is to resolve all entities in the serialized version of the XML document prior to translation.

- **Parsing difficulties**

  During the translation process the text will typically be transformed into different XML based translation formats such as XLIFF where the entity will cause a parsing error.

- **Problems with leveraged memories**

  The use of specially defined entity references can also cause problems with leveraged memories. The leveraged memory may contain entities not declared in the current document.

It is generally better to use alternative techniques rather that entity references:

```
<para>Use a <tool id="a1098">claw hammer</tool> to release the CPU
retention catch.</para>
```

Example 2: Proposed solution

One area where entities CAN be used to great effect is that of boilerplate text. The technique here is to use parameter entities to store the text. The text must always be linguistically complete in that it cannot rely on positional dependencies with regard to other entities etc. Boiler plate text is used solely within a DTD. There need to be parallel target language versions of the DTD for this technique to be used which can add to the maintenance cost, although judicious use of INCLUDE directives and DTD design can mitigate this.

2.2. Avoid translatable attributes

Translatable attributes can also look like a smart way of embedding variable information in an element.

```
<para>Use a <tool id="a1098" name="claw hammer"> to release the CPU
retention catch.</para>
```

Example 3: Incorrect use of translatable attributes

Unfortunately, they present the translation process with the following difficulties:

- Grammatical difficulties
  The same problems can arise as with entity references. If you want to use the text for indexing etc., then you cannot rely on the contents of translatable attributes to be consistent for inflected languages.
- Flow of text difficulties:
  With translatable attributes there are two possibilities regarding the flow of text:
  - The text is part of the logical text flow.
  - The text should be treated outside of the text flow.

If the text is to be part of the text flow then the translatable attribute causes the insertion of extra inline elements in translatable format (typically XLIFF format) of the file. If it is to be translated separately, then the translatable attribute forms a new text unit. The translator then needs to know if it is to be translated within the context of the original text unit or in isolation. With extra inline elements the burden is on the translator to preserve the encapsulating encoding, baring in mind that there may be significant changes in the sequence of such attribute text in the target language. Translation may often require that the position of the various components of a text unit are significantly rearranged.

```
<para>Use a <tool id="a1098">claw hammer</tool> to release the CPU
retention catch.</para>
```

Example 4: Proposed solution

There is a good rough rule of thumb that if text has more than one word then it should not be used in attributes. As a syntactical instrument attributes are much more limited than elements. For a start you can only have one attribute of a given name. The use of attributes should be reserved for single "word" values that qualify in a meaningful way an aspect of their element.

2.3. Avoid using CDATA sections that may contain translatable text

CDATA sections are typically used as a means of escaping multiple '<' and '&' characters. Unfortunately they pose particular problems for tools that are extracting such text. The problem is not one of the escaped characters, but how to treat the CDATA text.

```
<TEMPLATE><![CDATA[<p>Please refer to the <em>index page      </em> page
for further information</p>]]></TEMPLATE>
```

Example 5: CDATA section problems

The problem is a similar one to that posed by translatable attributes. Is the text to be treated as 'inline' to the surrounding text? What of the escaped characters. Are they to be replaced on translation with the appropriate characters that were originally escaped, or are they to be left in their escaped form. How is the software to know?

I have come across whole XML documents being embedded as CDATA within an encompassing XML document. This poses significant problems regarding the treatment of the CDATA text. It must first be extracted and then re-parsed before it can be extracted for translation.

Unless the text within CDATA sections is specifically never to be translated, please avoid using CDATA sections and use the standard built in character references to escape the text.

```
<TEMPLATE>&lt;p>Please refer to the &lt;em>index page        &lt;/em>
page for further information&lt;/p></TEMPLATE>
```

Example 6: Proposed solution

```
<TEMPLATE       xlink="ftp://ftp.xml-intl.com/resources/examples/inline-
cdata/ex1.xml"/>
```

Example 7: Or alternatively us a link to an external resources

2.4. Avoid the use of infinite naming schemes

Do not use the following type of element elm001, elm002, elm003 in well formed documents.

```
<?xml version="1.0" ?>

<resources xml:lang="en">

        <err001>Cannot open file $1.</err001>

        <hint001>Hint: does file $1 exist.</hint001>

        <err002>Incorrect value.</err002>

        <hint002>Hint: value must be between $1 and $2.</hint002>

        <err003>Connection timeout.</err999>

</resources>
```

Example 8: Example of infinite naming scheme usage

This presents problems for extraction programs and is not regarded as good XML practice. A much better way of doing this is to use the ID and IDREF attribute mechanisms to link elements together.

```
<?xml version="1.0" ?>

<resources xml:lang="en">

        <err id="001">Cannot open file $1.</err>

        <hint id="001">Does file $1 exist.</hint>

        <err id="002">Incorrect value.</err>

        <hint id="002">Value must be between $1 and $2.</hint002>

        <err id="003">Connection timeout.</err>          .          .
</resources>
```

Example 9: Proposed solution

2.5. Avoid placing translatable text in Processing Instructions (Pls)

Processing instructions are a very 'weak' syntactical instrument in XML. There is no built in mechanism in XML to assist syntactically in the preservation of Processing Instructions. Similarly any extraction tools will need to have special knowledge of the structure of the data in processing instructions.

```
<para>Use a <?tool name="claw hammer"?> to release the CPU retention
catch.</para>
```

Example 10: Incorrect use of translatable text in Pls

```
<para>Use a <tool id="a1098">claw hammer</tool> to release the CPU
retention catch.</para>
```

Example 11: Proposed solutions

It is generally not a good idea to have any processing instructions present within translatable text. The main reason is that there is no guarantee that they will survive the translation process. It is better to strip out any PIs prior to translation.

2.6. Avoid the use of text in bitmap graphics

There should be no excuse with the existence of SVGs to use bitmapped graphics. They pose particular problems in that the original bitmap will need to recreated for the target language with the translated text. This is usually a very costly and error prone process and requires appropriate target language knowledge of the person that is editing the graphics.

2.7. Never make any assumptions about text length sizes in your design

Always allow for the fact that the target language text may be significantly longer than the source. For example "Welcome" becomes "шчыра запрашаем" in Belarusian and "maligayang pugdatíng" in Tagalog. Design your output with flexibility in mind.

2.8. Always use UTF-8 (or alternatively UTF-16) encoding

With English source we can often get tempted to use 7 bit ASCII or ISO 8859/1 encoding. As soon as you find that you are required to translate into a language that is not covered by ISO 8859/1 you will find that trying to maintain documents in different encoding schemes a real problem. Always use UTF-8 from the start. It gives you immediate access to commonly used punctuation characters such as 'm-dash' and 'n-dash' etc. It also significantly simplifies your document processing. All XML parsing tools have to be able to cope with UTF-8 and UTF-16. UTF-8 is more economical in terms of space usage for most European Languages whose scripts are based on the Latin alphabet.

2.9. Never break a linguistically complete text unit

Never start a sentence in one non-inline element and continue it in another. You cannot rely on the translated text being in the same word sequence in terms of the sentence content as the target. It also makes the job of translation much more difficult as the translator does not see the whole sentence.

```
<para>

   <line>This text should not be</line>

   <line>broken this way – the translated text may well be in a
different order.</line>

</para>
```

Example 12: Example of a sentence broken over more than one element

2.10. Avoid the use of “typographical” elements

Use logical elements instead that encompass the text.

```
<para><b>Do not use</b> '<br/>' type      elements.</para>
```

Example 13: Example of typographical element usage

Use `emph` instead of `bold`. Encompass any text that requires to be on a line with `line` elements.

```
<para>
    <emph>Do not use</emph> 'br' type elements.
</para>
```

Example 14: Suggested correct usage

Avoid at all cost introducing any line breaks into the text stream. You can unconditionally guarantee that this will cause problems in some if not all of the target languages.

2.11. Do not mix translatable and non-translatable text in the same elements

Keep non-translatable PCDATA in different elements than translatable PCDATA.

```
<data-items>
       <data id="class">com.xmlintl.data.dataDefinition</item>
       <data id="text">Replace generic data definitions with specific instances.</item>
      </data-items>
```

Example 15: Example of mixed PCDATA

Most XML translation tools will have problems with this type of construct. It is only when inspecting the 'id' attribute that a decision can be made as to whether the PCDATA should be extracted or not.

```
<data-items>
       <class id="com.xmlintl.data.dataDefinition">
        <text>Replace generic data definitions with specific instances.</text>
       </class>
      </data-items>
```

Example 16: Suggested solution

2.12. Avoid holding Source and target PCDATA in the same document

This can cause all manner of problems for processing and extraction tools.

```
<para>
   <text xml:lang="en">My hovercraft is full of eels.</text>
   <text xml:lang="fr">Mon aéroglisseur est plein d'anguilles.</text>
   <text xml:lang="hu">Légpárnás hajóm tele van angolnákkal.</text>
```

```
    <text xml:lang="ja">私のホバークラフトは鰻で一杯です。</text>
    <text xml:lang="pl">Mój poduszkowiec jest pełen węgorzy.</text>
    <text xml:lang="es">Mi aerodeslizador está lleno de anguilas.</text>
    <text xml:lang="zh-CH">我隻氣墊船裝滿晒鱔. </text>
</para>
```

Example 17: Example of mixed source and target PCDATA

Unless your document requires mixed language content use a separate document instance to store each target language version. If you store both source and target data in the same document it will become unwieldy, overly large and cumbersome to process.

2.13. Clearly define text that requires translation

Keep any PCDATA that requires translation in different elements from PCDATA that does not require translation. Use special elements for text within PCDATA that is specifically not to be translated.

```
<para>The following part of this sentence should <notrans>not be
translated</notrans> at all.</para>
```