Proceedings of the 21st IASTED International Conference
APPLIED INFORMATICS
February 10-13, 2003, Innsbruck, Austria

Localizing XML documents through XSLT

Yijun Yu
Electrical Engineering Department
Gent University, Belgium
yijun@elis.rug.ac.be

Jianguo Lu
School of Computer Scinece
University of Windsor, Canada
jlu@cs.uwindsor.ca

Jinghao Xue
Muller Institute for Biomechanics
University of Bern, Switzerland
jxue@memot.unibe.ch

Yi Zhang
Information Technology Department
Gent University, Belgium
yi.zhang@intec.rug.ac.be

Weiwei Sun
Computer Science Department
Fudan University, China
wwsun@online.sh.cn

Abstract

Existing efforts on XML internationalization and localization have been focusing on the contents of XML documents instead of on the meta presentations such as tags and attributes. Although XML standard allows the use of unicode throughout a document including the meta level presentations, most tags and attributes of XML documents are still defined in English, which makes it difficult for native people to use.

This paper presents a pure XSLT stylesheet to completely translate and localize XML documents into different natural languages. Furthermore, we also describe how this technique can be applied to translation problems in programming (e.g. C and Java) or documentation (e.g. LaTeX) languages when the program or the document can be converted to and from XML documents.

Keywords XML localization, XSLT, XHTML, DocBook, program localization

1 Introduction

The world wide web (WWW) drives the information to freely flow across borders into different countries using a variety of natural languages. One would expect that all the exchanged information can be expressed in a local language. However, the standards for exchanging information in the web are mostly written in English. Hyper Text Markup Language (HTML) is such an example, where all tags are either English words or English abbreviations. Another example is the Extensible Stylesheet Language (XSL), the language to express transformations between the Extensible Markup Language (XML) [1] documents. People who can not understand English may suffer from a "culture shock" when reading the source of a document if he can not understand English.

We had asked a Chinese student to write a web page about his country. We told him that everything he needed would be Chinese using Chinese Front Page from Microsoft. Well, it is partially true. The menu system and the What You See Is What You Get (WYSIWYG) display in the tool do hide the English from the document. But curiously, he opened the source view to look at the code behind the web page: it is a mixture of English tags with Chinese characters. It is hard to go on without explaining the magic behind these tags. Such a language problem does not occur to Chinese only. Indeed, every native speaker of other languages has to learn quite a little English to understand the HTML.

Is there a way to alleviate the learning burden imposed on these people? This paper tries to explain a new way of using XSL to make the language accessible to them. The technique for the attempt is nothing else but still XSL [3] Transformations, the powerful transformer between XML documents that allow for unicode to be used in tags and attributes [1].

2 Motivation

Throughout the examples in the paper, we convert between documents in English and Chinese. The technique can be extended to other natural languages in a similar way.

First we use an HTML(hypertext markup language) example to show the basic steps in the treatment. A typical web page written in HTML looks like

```
<html>
  <head><title>A HTML document</title></head>
  <body><h1>Hello, world!</h1></body>
</html>
```

The document is written in English. Typically, a web page has a root tag `HTML` enclosed by two brackets. It has two subtrees which have `head` and `body` tags as the root respectively. The `title` tag in the header will normally be shown as the title of the web browser. The body part is the web page displayed inside the web browser. In this example, we welcome the world with a level one heading `h1`.

The HTML code of the localized document is shown as follows:

```
<?xml version="1.0" encoding="GB2312"?>
<html xmlns="urn:html">
  <head><title>HTML文档</title></head>
  <body><h1>你好，世界！</h1></body>
</html>
```

Although the English contents are localized, the formatting tags are still in English. In order to convert them also into Chinese, we use an XSLT transformation.

3 Approach

An XSLT transformation converts an XML document into another XML or a text document through a stylesheet. In our case, in order to use XSLT, we first assume the HTML is a strict XML document using the HTML tags, i.e. XHTML [9]. Thus the input for the stylesheet is an XHTML document and the output we expect is a localized document in HTML-like language where all tags are mapped one-to-one into keywords in another natural language.

The mapping between the two sets of keywords is first placed in the following XML document.

```
<?xml version="1.0" encoding="GB2312"?>
<dictionary>
 <entry type="element">
   <html>html</html><chtml>超文本</chtml></entry>
 <entry type="element">
   <html>head</html><chtml>文本首</chtml></entry>
 <entry type="element">
   <html>body</html><chtml>文本体</chtml></entry>
 <entry type="element">
   <html>title</html><chtml>标题</chtml></entry>
 <entry type="element">
   <html>h1</html><chtml>一级标题</chtml></entry>
 <entry type="attribtute">
   <html>size</html><chtml>尺寸</chtml></entry>
  ... </dictionary>
```

The document declares a dictionary which is a list of two types of entries corresponding to elements or to tags and to attributes respectively. Each entry has two children, one for the source language (e.g. XHTML) and the other for the localized language (e.g. Chinese HTML).

Using the dictionary, the translator needs to look up the keyword of each element or attribute and to replace it with the corresponding keyword in the new language. The target of such transformation for our XHTML example is shown as follows:

```
<?xml version="1.0" encoding="GB2312"?>
<超文本>
  <文本首><标题>HTML文档</标题></文本首>
  <文本体><一级标题>你好，世界！</一级标题>
  </文本体></超文本>
```

The translation is done by an XSLT, as shown in figure 1.

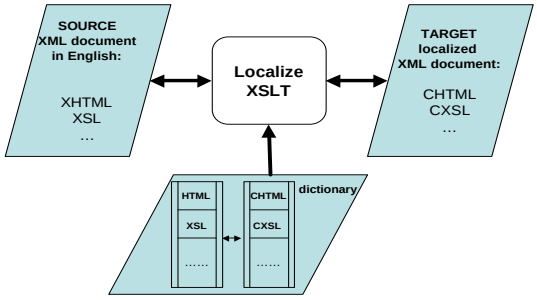

Figure 1. An XML document is localized by an XSLT stylesheet which translates it from a source language to a target one according to an auxiliary dictionary document.

In figure 2, we dissect the stylesheet into several understandable steps:

1. Declarations of the stylesheet: the first processing instruction tells that the stylesheet is also an XML document and its encoding is `GB2312` for simplified Chinese.

   ```
   <?xml version="1.0" encoding="GB2312"?>
   ```

   For other natural languages, one could use different encodings.

2. Declarations of the entities used in the stylesheet: an entity defines a macro replacement for a constant string throughout the stylesheet, and the use of an entity `ent` is `&ent;`. The defined entities will be replaced with the corresponding strings by an XML parser. Here in our example three entities are defined for names of the source and target language and the auxiliary dictionary file. They will be used extensively in the following stylesheet.

   ```
   <!DOCTYPE stylesheet [
   <!ENTITY source "html">
   <!ENTITY target "chtml">
    <!ENTITY diction "document('dictionary.xml')
   /dictionary/entry">]>
   ```

3. The following tag is the root of the XSL stylesheet, it has a version attribute and several name space attributes. The default name space is the same as name space `xsl`, and two other name spaces are respectively the URN (universal resource name) of the source and target languages.

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

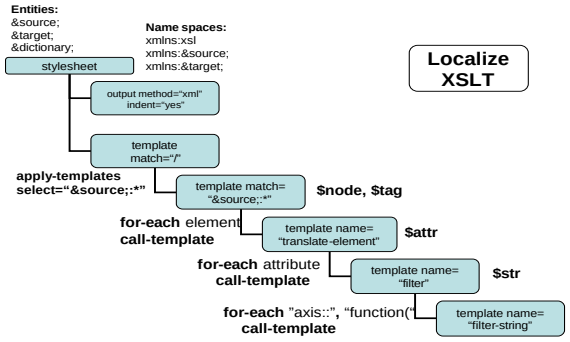

Figure 2. The localizing XSLT stylesheet traverses an XML document hierarchy top-down. The tags and attributes are translated from a source language to a target language. Predefined XPath axes and functions in the string value of attributes are replaced with their target language counterparts.

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:html="urn:&source;"
xmlns:chtml="urn:&target;">
       ...
</stylesheet>
```

4. The following tag specifies that the form of output is still XML and it will be indented automatically.

```
<output method="xml" indent="yes"/>
```

5. The top level template matches the root tag of the XML document in the source language and invokes the other templates for individual elements.

```
<template match="/">
 <apply-templates select="child::&source;:*"/>
</template>
```

6. The next template translates all the elements in the source language.

```
<template match="&source;:*">
  <variable name="node" select="."/>
  <variable name="tag" select="local-name()"/>
  <for-each select="&diction;[@type='element']">
   <if test="$tag=./&source;">
    <element name="./&target;">
     <call-template name="translate-element">
      <with-param name="node" select="$node"/>
     </call-template>
    </element>
   </if>
  </for-each>
  <if test="count(&entry;[@type='element'
and ./&source; = $tag])=0">
   <element name="$tag">
    <call-template name="translate-element">
     <with-param name="node" select="$node"/>
    </call-template>
   </element>
  </if>
</template>
```

The first two children of above template denote the current element as `$node` and its tag name as `$tag`. For each entry in the dictionary, if `$node` is the same as the value of a source language term, then it will be replaced with the corresponding target language term, and the attributes are processed by invoking another `translate-element` template. The difference between `call-template` and `apply-templates` is that the former does not changes the context element but the latter does. Therefore in the `translate-element` template, the context element " `.` " will be a dictionary entry instead of an element `$node` in the source document. To avoid losing the reference to the source XML document, we need to pass `$node` as a parameter to the invoked template using a `with-param` child.

When none of the entries in the dictionary matches `$tag`, as tested by comparing the count of matching elements with zero, the tag name of element `$node` remains `$tag` while the attributes and values will be translated by the same sub-template.

7. In the `translate-element` template, the current element in the source document is passed as a parameter. Then for each attribute name and value, one needs to look up the dictionary to replace the keywords in the source language with their counterparts in the target language. This is done similarly to the translation of the element's tag name. The major difference is in the XPath expression in the `for-each` select condition: to query the attribute through `"@*"` instead of to query the tag name through `"&source;:*"`.

```
<template name="translate-element">
 <param name="node"/>
 <for-each select="$node/@*">
  <variable name="attr"
select="local-name()"/>
  <variable name="anode" select="."/>
  <for-each select="&entry;[@type='attribute']">
   <if test="$attr=./&source;">
    <attribute name="./&target;">
     <call-template name="filter">
      <with-param name="attr" select="$anode"/>
     </call-template>
    </attribute>
   </if>
  </for-each>
  <if test="count(&entry;[@type='attribute'
and ./&source;=$attr])=0">
```

```
    <attribute name="$attr">
     <call-template name="filter">
      <with-param name="attr" select="$anode"/>
     </call-template>
    </attribute>
   </if>
  </for-each>
  <value-of select="$node/text()"/>
  <for-each select="$node/child::&source;:*">
    <apply-templates select="."/>
  </for-each>
</template>
```

8. For the value of an attribute, the `filter` template is invoked to translate the predefined function names. In our XHTML example, few predefined functions are used in the attributes, so we just simplify the filter with copying the attribute value.

```
<template name="filter"><param name="attr"/>
     <value-of select="$attr"/>
</template>
```

As result of above-mentioned steps, the XHTML document can be localized. This approach has two advantages:

- Swapping the string values of source and target entities, a localized document can also be translated back to an XHTML document. That is even more useful because people can use their mother tongues without knowing anything about English terminology for XHTML.
- The vocabulary of the dictionary can be augmented whenever new keywords are established. Putting them in a separate dictionary allows flexible translation without changing the translator itself.

A more "ambitious" attempt is to localize the XSL stylesheet using the stylesheet itself. Such an attempt is fulfilled easily by inserting terminology of XSL and CXSL into the source and target languages. Given different name spaces, a common term can be represented differently into different languages, for example, an entry of the dictionary may be

```
<entry>
    <lang1>A</lang1>
    <lang2>B</lang2>
    <lang3>C</lang3>
</entry>.
```

where a common term may have three different words in three different languages. In this way, a multilingual dictionary is more concise than a number of bi-lingual dictionaries.

Meanwhile, in order to translate the predefined keywords like axes in XPath expressions, the last step of the XSLT is modified into the following code:

```
<template name="filter">
 <param name="attrib"/>
 <call-template name="filter-string">
   <with-param name="str" select="$attr"/>
 </call-template>
</template>
<template name="filter-string">
 <param name="str"/>
 <for-each select="&entry;[@type='axis']">
  <variable name="axis"
select="concat(./&source;, '::')"/>
  <variable name="to_axis"
select="concat(./&target;, '::')"/>
  <if test="contains($str,$axis)
and starts-with($str, $axis)">
   <call-template name="filter-string">
     <with-param name="str" select=
"concat($to_axis, substring-after($str,$axis))"/>
   </call-template>
  </if>
 </for-each>
 <if test="count(&entry;[@type='axis'
and contains($str,$axis)])=0">
    <value-of select="$str"/>
 </if>
</template>
```

In above code, the first template `filter` calls the second template `filter-string` to replace every occurrence of a pattern ("axis::" or "function(") with their counterparts in the target language. Note that `filter-string` is a recursive template that keeps doing the translation of one such pattern until no match is found anymore.

At the end of this section, we list part of the localized results:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stylesheet [
 <!ENTITY source "xsl"><!ENTITY target "cxsl">
 <!ENTITY diction "document('dictionary.xml')/
         dictionary/entry"> ]>
<样式表单 版本="1.0" xmlns="urn:cxsl">
 <输出 方法="xml" 缩进="yes"/>
 <模板 匹配="/">
   <应用模板 选择="子女::xsl:*"/> </模板>
 <模板 匹配="xsl:*">
  <变量 名字="node" 选择="."/>
   <变量 名字="tag" 选择="局部名()"/>
   <每个 选择="&entry;[@type='element']">
   <如果 满足条件="$tag=./xsl">
    <元素 名字="{./cxsl}">
     <调用模板 名字="translate-element">
      <带参数 名字="node" 选择="$node"/>
     </调用模板> </元素> </如果> </每个>
  <如果 满足条件="计数(&entry;[@type='element'
     并且 ./xsl = $tag])=0">
   <元素 名字="{$tag}">
    <调用模板 名字="translate-element">
     <带参数 名字="node" 选择="$node"/>
    </调用模板> </元素> </如果> </模板>
  ...
</样式表单>
```

4 Applications

In this section we discuss the use of the localization through XSLT in two domains, i.e. the programming languages and the documenting languages.

4.1 Programs

Programming languages are mostly in English. To a non-English speaker, the keywords of a program are not easy to understand and remember.

The XSL can be regarded as a programming language, but in general, a program is not an XML document.

Previous efforts like `ret4j` have been made to enable converting a Java program into an XML document [8]. Likewise `yaxx` [12] can output the syntax of a C or Fortran program into XML according to the YACC grammar [6]. Such an XML document is a good candidate for the localization. Since the grammar rules for a programming language is fixed, one can use a single translation dictionary that translates the program. The translation can be done automatically with the help of `yaxx` or `ret4j`.

For example, the following C program prints "Hello, world":

```
void main() {
  printf("Hello, World!");
}
```

YACC is a compiler-compiler which takes the grammar of a programming language like C as the input and it generates a parser that constructs a syntax tree of the C program. Our extension to YACC is called `yaxx`, which output the internal syntax tree structure while parsing the C program.

The example program is first output into an XML document by `yaxx` reusing an ANSI-C grammar [10]. It is shortened here to present the essential elements.

```
<file>
 <external_definition><function_definition>
   <declaration_specifiers>
    <type_specifier>void</type_specifier>
    <declarator><identifier>main</identifier>
      <PUNCT_LPAR/><PUNCT_RPAR/></declarator>
   </declaration_specifiers>
   <function_body>
    <compound_statement><PUNCT_LBRACE/>
     <statement_list>
       <expression_statement><primary_expr>
         <identifier>printf</identifier>
        </primary_expr><PUNCT_LPAR/>
        <primary_expr>"Hello, World!"
        </primary_expr><PUNCT_RPAR/>
       </expression_statement></statement_list>
    <PUNCT_RBRACE/></compound_statement>
   </function_body>
  </function_definition></external_definition>
</file>
```

Then the XML document is converted into a localized document by the XSLT presented in this paper:

```
<文件>
 <外部定义><函数定义><函数说明>
    <类型描述>空</类型描述>
    <说明><标识符>主程序</标识符>
      <左括号/><右括号/></说明>
   </函数说明><函数体>
    <复合语句><左花括号/>
     <语句表><表达式语句><表达式语句>
          <标识符>格式打印</标识符>
        </表达式语句><左括号/>
        <表达式语句>"你好，世界！"
        </表达式语句><右括号/>
       </表达式语句></语句表>
    <右花括号/></复合语句>
   </函数体></函数定义></外部定义>
</文件>
```

A localized XML document can be supplied to a code generation XSLT to regenerate the localized code as follows:

```
空 主程序() {
  格式打印("你好，世界！");
}
```

This localized code can not be parsed by a common C compiler, but it can be kept as a documentation accompanying with the original program.

The grammar for the language of YACC grammar is also a YACC grammar (see the implementation of `bison` [4], an open source variant of YACC), therefore the YACC grammar can also be automatically translated through the use of our XSLT and a certain dictionary. The localized YACC with Unicode support can accept the localized program as if it is the original C program. The relationship of these tools are illustrated in figure 3.

4.2 Documentation

DocBook is a standard way of representing books, articles and technical reports uniformly in XML [11]. The tag names of DocBook are "unfortunately" in English too. Therefore to localize a DocBook can also be assisted with the XSLT approaches in the paper.

Existing tools that translates a DocBook into LaTeX [5, 2] can be extended to translate the localized DocBook into a LaTeX document. The LaTeX document can be further rendered using the packages aware of the local language. An example of such application is the presentation of this paper. the localizing XSLT in preparing the LaTeX source of this paper. The paper is prepared in following steps:

1. Initially the paper was prepared as a localized DocBook XML document;

2. A reverse translation converted it into an English DocBook XML document;

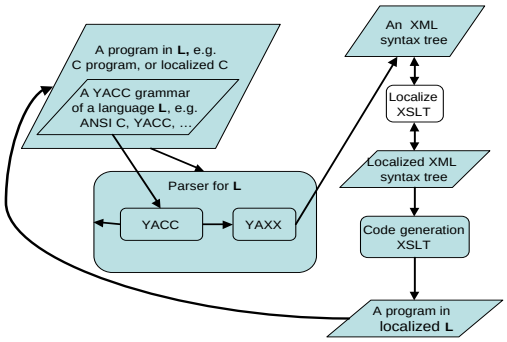

Figure 3. Using a YACC grammar of the programming language L, `yaxx` outputs the syntax tree of a program into an XML document. A localizing XSLT translates the document into the target language and a code generation XSLT transforms the localized XML document into a localized program.

3. The `DB2LaTeX` tool [2] was used to convert the DocBook XML document into LaTeX. We made minor changes to allow the use of CJK (Chinese Japanese Korean) LaTeX package [7] in the LaTeX output.

4. CJK package was used to render the LaTeX output mixed with Chinese and to produce good quality DVI and PDF document.

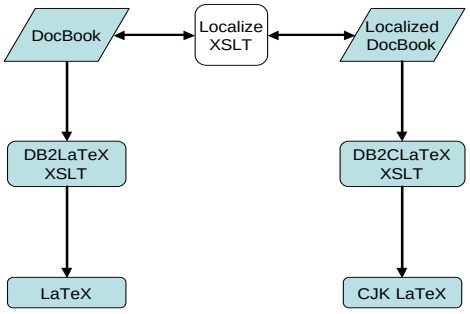

Figure 4. The DocBook document for this paper can be localized using XSLT. The DB2LaTeX XSLT stylesheets are extended in order to produce a localized document using the CJK LaTeX package.

5 Conclusion

This paper discusses an automatic approach to localize any XML document in English to other natural languages and to convert the localized document back to an English document. This approach makes the localization of XHTML and XSL documents painlessly. Using the same XSLT together with other tools like `yaxx`, a program can be localized; using the same XSLT together with DB2LaTeX, documents written in DocBook can also be localized to CJK LaTeX. We see this approach as a way to extend the English community to a multilingual one.

References

[1] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible Markup Language (XML). *The World Wide Web Journal*, 2(4):29–66, 1997.

[2] R. Casellas. DocBook to LaTeX2e converson with XSLT, 2002.

[3] J. Clark and S. Deach. Extensible Stylesheet Language (XSL), Version 1.0. World Wide Web Consortium Working Draft, Aug 1998.

[4] C. Donnelly and R. Stallman. *Bison Manual: Using the YACC-compatible Parser Generator, for Version 1.29*. Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA, 2000.

[5] M. Goosens and S. Rahtz. *The LaTeX Web Companion*. Addison Wesley: Reading, Massachusetts, 1999.

[6] I. E. Gorman. Lex and yacc: Compiler-construction techniques for the everyday programmer. *Dr. Dobb's Journal of Software Tools*, 21(2):86–97, feb 1996.

[7] W. Lemberg. The CJK package for LaTeX2e — multilingual support beyond `babel`. *j-TUGboat*, 18(3):214–224, sep 1997.

[8] E. Mamas, G. Baron, and K. Kontogiannis. Reengineering tool kit for java. Technical report, 2001.

[9] M. Sauers and R. A. Wyke. *XHTML essentials*. Wiley, New York, NY, USA, 2001.

[10] A. standard. ANSI C: Standard x3.159, 1989.

[11] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. O'Reilly and Associates, Inc., 1999.

[12] Y. Yu and E. H. D'Hollander. YACC extension to XML. Technical report, Ghent University, 2002.