Making XML document markup international

Yijun Yu1,∗,†, Jianguo Lu2, John Mylopoulos1, Weiwei Sun3, Jing-Hao Xue4 and Erik H. D’Hollander5

1Computer Science Department, University of Toronto, 40 St. George Street, Toronto, Canada M5S 2E4
2University of Windsor, Windsor, Canada N9B 3P4
3Fudan University, Shanghai 200433, People’s Republic of China
4University of Bern, 3010 Bern, Switzerland
5University of Ghent, 9000 Ghent, Belgium

SUMMARY

In name and in practice, the World-Wide Web (hereafter Web) is used around the World beyond English-speaking areas. This creates a tremendous need to internationalize standard terminology used in the technologies that make the Web possible. Existing efforts on XML internationalization (i18n) and localization (i10n) have focused on the content of XML documents instead of the terms used in markup (annotations) such as elements and attributes. The SGML standard ISO 8879 supports the use of Unicode (ISO 10646) throughout a document, including markups. However, most elements and attributes of XML documents are still defined in English, thereby limiting their use among non-English speakers. This paper presents an XSLT-based method that can completely localize the markup of XML documents into different natural languages. We also describe how the proposed technique can be applied to translation problems in programming (e.g. C and Java) or documentation (e.g. LATEX or other formatting languages) so that a program or a document can be converted to and from an XML format. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: eXtensible Markup Language (XML); localization (i10n) and internationalization (i18n); eXtensible Stylesheet Language Transformations (XSLT); programming and documentation languages; markup; transformation

1. INTRODUCTION

The Web has made it possible for information to flow freely across borders and through countries that speak a variety of natural languages. One would expect that all information exchanged could

∗Correspondence to: Yijun Yu, Computer Science Department, University of Toronto, 40 St. George Street, Toronto, Canada M5S 2E4.
†E-mail: yijun@cs.toronto.edu

Received 24 October 2002
Revised 1 September 2003
Accepted 19 April 2004
be expressed in any native language. Unfortunately, although the content can be expressed in native
languages, in most cases the markups in an eXtensible Markup Language (XML) [1] document are still
written in English. This means that in order to develop and maintain XML documents, native speakers
of other languages have to learn much of the XML markup terminology in English. Not surprisingly,
this makes it difficult for non-native English speakers to fully participate in the international Web user
community.

Is there a way to alleviate the learning burden imposed on Web users? Existing efforts on XML
internationalization (i18n) and localization (i10n) focus on the content of XML documents instead
of the terms used in markup (annotations) such as elements and attributes. This paper proposes a
method to make an XML document completely available to non-English speakers through linguistic
localization. Conversely, the method can make a native XML document available to the international
community.

Our proposed method exploits three key techniques as follows.

- Unicode support for markups of SGML languages, including XML. The SGML standard ISO
  8879 [1] enables the use of Unicode (ISO 10646) throughout a document including its markups.
  Our method is feasible thanks to this standard.
- XSLT [2] stylesheet for XML document transformations. Although often used just as a
  presentation stylesheet, XSLT is a functional programming language that supports the
  transformation of a source XML document into a target one.
- XPath expressions for querying a dictionary in XML format. Encoding a dictionary in XML
  format makes it extensible to the translation need of various XML markups. Since the dictionary
  is separate from the localizing XSLT stylesheet, it is possible to choose different source/target
  languages without requiring any change to the static XSLT stylesheet.

We explain the technical details of the method as well as interesting applications on software
developments through illustrative examples.

2. A MOTIVATION EXAMPLE: XHTML CONVERSION

Recently, one of the authors asked a Chinese student to write a Web page in Chinese. The student was
told that she does not need to understand English, since she would use the Chinese version of FrontPage
from Microsoft. Well, this turned out to be only partly true. The menu system and the What You See
Is What You Get (WYSIWYG) display in the tool do hide the English from the user. However, when
she opened the source view to look at the marked-up source code behind the Web page, she found a
mixture of English elements with Chinese content. The moral of the story is that it is hard for a native
user to use Web technology without knowing the meaning of different markups.

This language problem does not only happen with Chinese users. Indeed, every native speaker of
other languages has to learn some English to understand XHTML [3]. In the following example,
we show how our method converts between documents in Chinese, English, Dutch and Greek.
The technique can be extended to other natural languages in a similar fashion.

First we use an XHTML example to show the basic steps in the technique. A typical Web page
written in XHTML looks as follows:
This Web page is written in English. Typically, it has a root element HTML, which in turn has two elements called head and body. In the head element there is a title element, the content of which will be shown as the title in a Web browser. The content of the body element is the Web page displayed inside a Web browser. In this example, we welcome the world with a level one heading element h1.

This document may be localized into Chinese as follows:

```xml
<?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>
```

As we can see, although the English content at the presentation level is localized, the element and attribute names, such as title and xmlns at the source level are still in English.

In order to do the translation on the markups as well, we propose an XSLT solution as a plug-in for the Web-authoring tool. Using this method, a Web-authoring tool such as Chinese FrontPage can plug-in a localizing feature that localize the source markup of the Web page for the end-user when they want to view the source.

To facilitate the solution for the Web page author, the dictionary provider and the localization tool developer come into play. Given a predefined English–Chinese dictionary for the XHTML translations and an XSLT stylesheet independent of the dictionary, the Web page author can still edit the Web page in WYSIWYG fashion, and they can further adjust a Chinese XHTML source to tune the Web page without manual effort. It is even better if the dictionary is provided either by a third party or by the standard community.

Such a scenario applies for any XML-based editor that requires markup translations.

### 3. XSLT Transformations

An XSLT transformation converts an XML document into another XML or text document through a stylesheet. In our case, the input to the stylesheet is an XML document and the output is a localized document where all element types and attributes (markups) are mapped one-to-one into names in another natural language.

The translation is done by a localizing XSLT, as shown in Figure 1. The localizing XSLT engine uses a dictionary to translate the markups in the source XML document into the target XML document. In this section, we first explain the dictionary structure, then explore the localizing XSLT step by step.

#### 3.1. Dictionary

The dictionary defines a mapping between the names for XML markups, i.e. elements and attributes. The following XML document shows the structure of such a multilingual dictionary for translation between languages in English, Chinese, Dutch and Greek.
Figure 1. The markup in an XML document is localized by XSLT from a source language to a target language according to an auxiliary dictionary document.

<dictionary>
  <entry type="element"><en_US>html</en_US><zh_CN>超文本</zh_CN>
    <nl_BE>html</nl_BE><el_GR>html</el_GR></entry>
  <entry type="element"><en_US>head</en_US><zh_CN>文本头</zh_CN>
    <nl_BE>hoofding</nl_BE><el_GR>κεφάλι</el_GR></entry>
  <entry type="element"><en_US>body</en_US><zh_CN>文本体</zh_CN>
    <nl_BE>tekst</nl_BE><el_GR>περιεχόμενα</el_GR></entry>
  <entry type="element"><en_US>title</en_US><zh_CN>标题</zh_CN>
    <nl_BE>titel</nl_BE><el_GR>τίτλος</el_GR></entry>
  <entry type="element"><en_US>h1</en_US><zh_CN>一级标题</zh_CN>
    <nl_BE>kop1</nl_BE><el_GR>τίτλος 1ος</el_GR></entry>
  <entry type="attribute"><en_US>size</en_US><zh_CN>尺寸</zh_CN>
    <nl_BE>grootte</nl_BE><el_GR>μέγεθος</el_GR></entry>
  ……
</dictionary>

The XML document defines the dictionary as a list of entries for markups. Each entry has at least two children, one of which is used for the source language (e.g. English, abbreviated as “en_US”), while the others are used for target languages (e.g. Chinese, abbreviated as “zh_CN”). For each language, we use its two-letter language code in ISO 639 together with its two-letter country code in ISO 3166-1.
Figure 2. The localizing XSLT traverses an XML document top-down. The markup is 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.

For each markup, the translator looks up the name in this dictionary and replaces it with the corresponding name in the target language. The result of this translation for our XHTML example is shown as follows:

```xml
```

3.2. The localizing XSLT

The localizing XSLT stylesheet consists of several snippets corresponding to the eight steps explained below. Figure 2 shows the structure of the stylesheet.

1. The first processing instruction tells an XML processor that the stylesheet is also an XML document:

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

2. The following snippet declares the entities to be used as macros for constant strings in the stylesheet. In our example, three entities are declared, one for the source language name, one for the target language name, and one for locating the auxiliary dictionary file. These entities will be used extensively in the following parts of the stylesheet. Once an entity is declared, we can use an entity reference such as &source; to denote “en_US”:

```xml
<!DOCTYPE stylesheet [<!ENTITY source "en_US"> <!ENTITY target "zh_CN"> <!ENTITY dictionary "document\{\'dictionary.xml\'}\dictionary/entry">]>
```
3. The ‘stylesheet’ element is the root of the stylesheet. It has a ‘version’ attribute and several XML name-space attributes. The default XML name-space as indicated by ‘xmlns’ is the same as the name-space xmlns:xsl, so that we do not need to put an ‘xsl:’ prefix before any element of the stylesheet. The other two name-spaces are the Universal Resource Names (URNs) for the source and target languages. An URN identifies the document type to an XML parser. Although a URN may contain a reference to a remote site (e.g. ‘http://www.w3.org/1999/XSL/Transform’), that reference need not to be fetched for validating a stylesheet document. Technically, an URN is a Unicode string, including English.

```xml
<stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:en_US="urn:&source;" xmlns:zh_CN="urn:&target;">
...
</stylesheet>
```

4. The ‘output’ element specifies that the output is in an indented XML format:

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

5. The following ‘template’ element matches the root element of the input XML document. It invokes the other templates that match for individual elements in the source XML document.

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

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

```xml
<template match="&source;::*">
 <variable name="node" select=".="/>
 <variable name="tag" select="local-name()="/>
 <for-each select="&dictionary;[@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>
</template>
```

The first two children of the above template denote the current element as $node variable and its name as $tag variable. For each entry in the dictionary, if $tag is the same as the value of the source
language child, then it will be replaced with the corresponding target language child. The attributes of
the element are processed by invoking a `translate-element` template, as explained in step 7.
The difference between `call-template` and `apply-templates` is that the former does not
change the context element but the latter does. Therefore, in the `translate-element` template, the
context element `.` will be a dictionary entry instead of an element `$node` in the source document.
To avoid losing the reference to the source XML document, we pass `$node` as a parameter to the
callee template using a `with-param` child.

When none of the entries in the dictionary matches `$tag`, as tested by comparing the count of
matched elements with zero, the name of element `$node` remains being `$tag` while its 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, we need to look up the dictionary to replace the
names in the source language with their counterparts in the target language. This is done in a similar
fashion to the translation of the element’s name. The major difference is in the XPath expression in the
for-each select condition in comparison with the template match condition: to query the attribute
through `"@*"` rather than query the name being queried through `"&source;::*"`.

```xml
<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 predefined function
names. In our XHTML example, few predefined functions are used in the attributes, so we can simplify
the filter by copying the attribute value.
As a result of the above-mentioned steps, an XHTML document can be localized. Figure 2 illustrates the top-down traversal in the stylesheet.

### 3.3. Localizing the XSLT stylesheet

A seemingly more ambitious task is to localize the XSLT stylesheet document by the XSLT itself. This task can easily be fulfilled by inserting the markups of XSL into the dictionary. In order to translate the predefined key words in the XPath axes such as ‘parent::’, the last step of the translation is declared in the following templates:

```xml
<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 the code above, 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 until no match is found anymore.

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stylesheet[<!ENTITY source "en_US"><!ENTITY target "zh_CN"><!ENTITY diction "document('dictionary.xml')/dictionary/entry">]>

<stylesheet xmlns:xsl="urn:xsl1" target="zh_CN">
  <xsl:template name="filter">
    <xsl:param name="attrib"/>
    <call-template name="filter-string">
      <with-param name="str" select="$attr"/>
    </call-template>
  </template>

  <template name="filter-string">
    <xsl: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>
</stylesheet>
```
Using this example, we have shown that not only can a Web page author benefit from a localization, but so can a stylesheet designer. The conversion can be done automatically using the stylesheet, which can be localized itself, thereby removing the English burden for a local stylesheet designer.

The dictionary presented here confines itself to the markups of HTML and XSL merely to illustrate the concepts involved. Extending the dictionary contents, any XML markups can be translated by the same XSLT, as we have shown by adding XSL markups into the dictionary for HTML markups.

Moreover, the localizing stylesheet serves as a highly reusable program for different purposes. For example, simply by inverting the source and target entities, the stylesheet can also be used to internationalize XML markups. The following sections explore its applications further.

4. APPLICATIONS

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

4.1. Programs

Like HTML pages, programs are mostly written in English. To a non-English speaker, the key words of a program are not easy to understand or remember.

To localize a program in a programming language other than XSL or XSLT, we need to XML-ize the program first. Previous efforts, such as ret4j, can be used for the task of converting a Java program into an XML document [4]. Similarly, yaxx [5] does the same for programs specified by any YACC grammar. Such an XML document is a good candidate for our localization process. Since the grammar rules for a programming language are fixed, one can feed the localizing XSLT engine with a single translation dictionary to translate the XML equivalent of a program automatically.

For example, the following C program prints ‘Hello, world’:

```c
void main() {
    printf("Hello, World!");
}
```
The example program is first translated into an XML document by yaxx, reusing an ANSI-C grammar in YACC [6]. YACC is a compiler-compiler that takes as an input the grammar of a programming language and generates a parser that constructs a syntax tree of any program in that language [7]. Our extension to YACC is called yaxx, which adds XML-processing into the YACC-generated parser. The generated parser from yaxx can output the syntax tree structure in XML format after parsing a C program. For our example, the generated XML syntax tree is shortened here to present the essential elements:

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

  </function_definition></external_definition> </file>
```

Next the XML document is converted into a localized document using the XSLT presented in this paper:

```
<文件>
  <函数定义> <函数声明>  
    <类型描述> void </类型描述> 
    <说明> void </说明> 
    <左括号> / </左括号>
  </函数声明>
  
  <复合语句>  
    <语句表>  
      <表达式语句>  
        <表达式语句>  
          <表达式语句>  
            "Hello, World!"
          </表达式语句>
        </表达式语句>
      </表达式语句>
    </语句表>
  </复合语句>
</文件>
```

The localized XML document is supplied to a code generation XSLT to regenerate the localized code as follows:

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

Although this localized code cannot be parsed by a common C compiler, it can be kept as documents accompanying the original program.

The grammar for the YACC specification language is also a YACC grammar (see the implementation of bison [8], an open source variant of YACC), therefore the YACC grammar can also be
Localizing XSLT A program in a non-XML language \( L \), e.g., a C program, or translated C A YACC grammar of a language \( L \), e.g., ANSI C, YACC, … Parser for \( L \) YACC YAXX An XML syntax tree 
Localizing XSLT Translated XML syntax tree 
Code generation XSLT A program in translated \( L \) 

Figure 3. Using a YACC grammar for the programming language \( L \), yaxx outputs the syntax tree of a program in \( L \) as an XML document. A localizing XSLT stylesheet translates the document markups into the target language and a code generation XSLT transforms the localized XML document into a localized program.

automatically translated using our XSLT stylesheet and a dictionary for YACC. The localized YACC with Unicode support can accept the localized program as if it were the original C program. The combined use of these tools is illustrated in Figure 3.

4.2. Documents

DocBook offers a standard for representing books, articles and technical reports uniformly in XML [9], which is in a similar vein to what Flynn has proposed for typesetting through SGML and HTML [10]. The element names of DocBook are written in English as well. Therefore, our XSLT approach can be used to localize a DocBook document.

Existing tools that translate a DocBook into \( \LaTeX \) [11,12] can be extended to translate a localized DocBook document into a \( \LaTeX \) document. The \( \LaTeX \) document can be further rendered using packages that are aware of the local language. An example of such an application is the preparation of this paper. Figure 4 summarizes the use of the localizing XSLT in preparing the \( \LaTeX \) source for this paper. Specifically, this paper was prepared as follows.

1. Initially the paper was prepared as a localized DocBook XML document, all in Chinese.
2. A reverse translation converted it into a DocBook document with English tags and Chinese content.
3. The DB2LaTeX tool [12] was used to convert the DocBook XML document into \LaTeX. We made minor changes to allow the use of the CJK (Chinese Japanese Korean) \LaTeX package [13] in the \LaTeX output.
4. The CJK package was used to render the \LaTeX output mixed with Chinese to produce a good-quality DVI and PDF document.

Apart from \LaTeX and DocBook, Microsoft Word is also widely used for documentation. In fact, Microsoft Word 2003 outputs XML documents and its wide adoption for XML support of documentation is just a matter of time. We have given a solid demonstration here that a technical paper can be localized (although partially) by XSLT.

5. CONCLUSIONS AND FUTURE WORK

This paper has proposed an approach to the automated translation of XML markups back and forth between different natural languages. Through concrete examples we have shown how to automatically translate XHTML and XSL markups. Our proposed method can be used for any XML markup translation. Combined with yaxxx or DB2LaTeX, the same XSLT can localize a non-XML program or a \LaTeX document. We view our proposal as a way to extend the English-speaking community of Web users into a truly world-wide community.
Many obvious extensions are within easy reach as follows.

- **Developing a multi-lingual dictionary for multiple communities and multiple markup standards.** Our technique maintains a list of multiple dictionary entries, which allows for word-by-word multilingual translation between English, Chinese, Dutch and Greek, etc. Since each new XML sub-language proposes new markups and existing XML standards are changing and growing, a community must agree on an evolving dictionary and an evolution process.

- **Extending markups with multimedia annotations.** An extension to the dictionary might annotate a word with its pronunciation or its illustration. Technically this requires treating each source/target of an entry as an XML subtree instead of a plain tag. Since our XPath query fetches subtree nodes, such an extension is not difficult to adapt. Thus XML standards for sound, such as VoiceXML, and for graphics, such as SVG, are perfect candidates for this multimedia extension.

- **Standardizing terminology for different dialects.** In our XML-based dictionary, one meaning in one source language is denoted by one expression in the target language. Extending this one-to-one dictionary mapping to a \( N\)-to-one mapping (that is, having just one meaning for more than one expressions) is helpful for standardizing the usage of a language. For example, various dialect terms in Chinese can be translated automatically to a standard mandarin term. This standardization makes it much easier to communicate between speakers of different dialects as well as speakers of different languages.

- **Parsing localized programs.** Currently, localized programs are used only as accompanying documents to the original program because of limited Unicode support in YACC-like parsers. Once a Unicode-enabled YACC parser is available, programs in local languages can be considered as real programs just like the internationalized versions.

- **Localizing or internationalizing Word documents.** Documents in MSWord 2003 can be saved in XML format. Thus the localization of DocBook documents can be extended to work on Word XML documents.

**ACKNOWLEDGEMENTS**

We are grateful for the encouragement from the editor Andy Wellings and for useful feedback from the anonymous referees. We also thank Greg McArthur for proof-reading the final draft.

**REFERENCES**