The ITB Journal

Volume 5 | Issue 1 Article 32

2004

Auto Generation of XLIFF Translation Documents from Proprietary File Formats

Kieran O'Connor
Institute of Technology Blanchardstown, Dublin 15, Ireland., kieran.o'connor@itb.ie

Geraldine Gray
Institute of Technology Blanchardstown, Dublin 15, Ireland.

Auto Generation of XLIFF Translation Documents from Proprietary File Formats

Kieran O'Connor & Geraldine Gray
kieran.o'connor@itb.ie
Institute of Technology Blanchardstown, Dublin 15, Ireland

Abstract

The handling of proprietary documents by localisation vendors is time consuming and error prone, and represents a significant challenge to localisation projects. Vendors with many customers, each with its own set of document formats, must potentially support a document format set numbering in the hundreds or thousands. This paper describes an approach to automating the extraction of translatable text from a variety of file formats. The solution is based on XLIFF, language parsers, and XML transformations.

Keywords: Localisation, grammar, regular expression, JavaCC, JJTree, XLIFF, XSLT, XML, XML Schema, transformation, translation, language, parse.

1 Introduction

Localisation is the process of adapting text to specific target audiences in specific geographic locations [WorldLingo, 2004, section: Glossary of Terms: Localization]. It is a fundamental process for many industries expanding or product selling into foreign regions. Hence there is a large market for vendors who provide translation services. Making the translation process as efficient and streamlined as possible provides benefits both to the document owner and to the vendor. The owner realises a quicker translation turnaround, and the vendor leverages its efficient process in acquiring and retaining business. [Rubric 2000].

Current challenges in the localisation process include insufficient operability between tools; lack of support for an overall localisation workflow; the number of file formats localisation developers have to deal with; and the large number of proprietary intermediate formats. [8th Annual International Localisation Conference 2003]

To address some of these challenges, and OASIS technical committee, whose objective was to address the lack of standards in the localisation industry, has developed an XML based standard called XLIFF - XML Localisation Interchange File Format. The specification provides the ability to mark up and capture localizable data and interoperate with different processes or phases without loss of information. [XLIFF 2004] Version 1.1 of the standard was submitted to the Oasis standards review process in November 2003. The significance of XLIFF is that translation tools can now be developed which accept input in a common format, i.e. XLIFF, enhancing operability between tools.

The research documented in this paper was done in collaboration with a local IT company, who are developing a product that implements a localisation workflow based on the XLIFF standard. However as stated above, one of the major challenges of localisation is the large number of file formats presented by

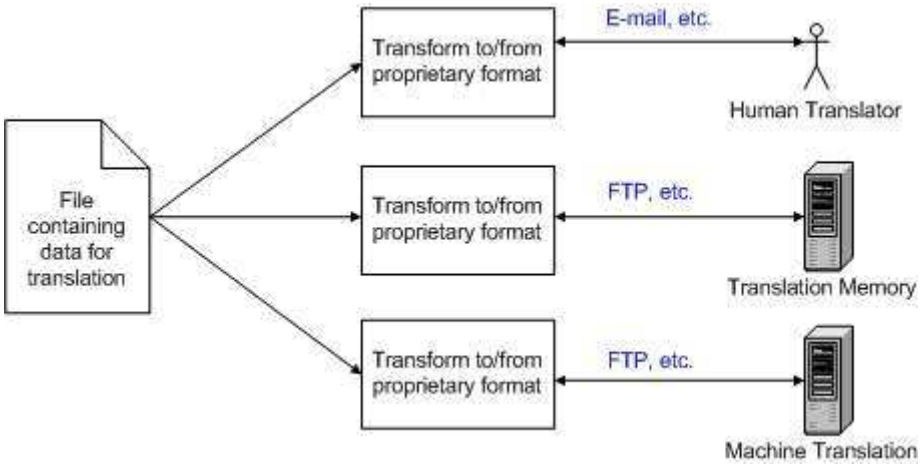
customers for translation. Therefore an XLIFF based localisation workflow requires pre-process of a proprietary file format to extract translatable text, and insert it into an XLIFF document. Once the workflow is complete, and the translated text has been added to the XLIFF document, post processing is required to insert this translated text back into the propriety file.

The focus of the work documented in this paper was to investigate if it is possible to automate these pre-processing and post-processing steps, given the variety of file formats presented for translation. At the time of writing, there was no documented work in this area. It is out belief that ITB, in conjunction with a local technology company, are the first to publish a technique for automating these pre-processing and post-processing steps.

2 State of the Art

Currently the typical translation process consists of a proprietary document submission to a localisation vendor for processing. This could be done, for example, online or through the use of physical storage devices. The document is manually examined to determine its type and to identify the text contained within which is translatable. Therefore there is a requirement for the examiner to understand the document contents. The translatable text is then stored in a file, such as a name/value pair file. The file is then passed through various tools, such as machine translation and translation memory, until as close to 100% translation as possible is achieved. Specialised human translators may also be used when necessary. The tools used for the machine translation and translation memory tasks each require the input data to be of a particular format, depending on the implementation. Thus file transformations may be required at each step. When the translation process is complete the original document must be re-constituted complete with the new translations.

The diagram below illustrates the basic translation process where proprietary file and exchange formats are used. The file passed to each translation component must be in a format understandable by receiver. Thus it must be transformed.



▪ Figure 1 – Proprietary document localisation

The transformation of a file from the proprietary formats supplied by the customer, to the proprietary format required by the translation tool, can be a costly and complicated task. When new file formats are introduced so also must a new transformation rule set be developed.

3 Methodology

This research has identified two categories of input file formats that cover the majority of input files presented for translation. The first category is documents that can be described using a language grammar. The second category is documents in XML format and hence can be described by an XML Schema.

3.1 Grammar

3.1.1 Backus-Naur Form (BNF) & Extended Backus-Naur Form (EBNF)

Language rules, manifested by grammars, are a method of describing the expected lexical and syntactic structure of a document. Backus-Naur Form (BNF) and Extended Backus-Naur Form (EBNF) are standard ways of describing complex languages using grammars. Particularly, they are a standard means of describing languages that comprise a large number of possible valid expressions. The following is an example of a simple EBNF grammar that describes how integers may be combined to form a valid expression e.g., 5+8.

```
simpleLanguage ::= integerLiteral (“+” | “-“ | “/” | “*”) integerLiteral
integerLiteral ::= [0-9]+
```

3.1.2 JavaCC Grammar

This research utilises the JavaCC grammar format, as specified for the JavaCC parser generation tool (detailed in section 4). This format is specific to JavaCC but is closely related to EBNF. There is a grammar repository of existing JavaCC grammars available at the following address: http://www.cobase.cs.ucla.edu/pub/javacc/.

3.2 XML Schema

XML documents are described using an XML Schema. The XML schema defines the contents of an XML document as a valid tree structure of elements and their children. The following shows the definition of a complex element that contains two child elements.

```
<xsd:complexType name="PolicySyncRq">
      <xsd:sequence>
                  <xsd:element ref="RqUID" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="PolicyNumber" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
</xsd:complexType>
```

4 Technologies

Various technologies are utilised in this research. Some are directly related to localisation such as XLIFF, others are more general such as JavaCC and XSLT. This section gives a brief overview of each technology used.

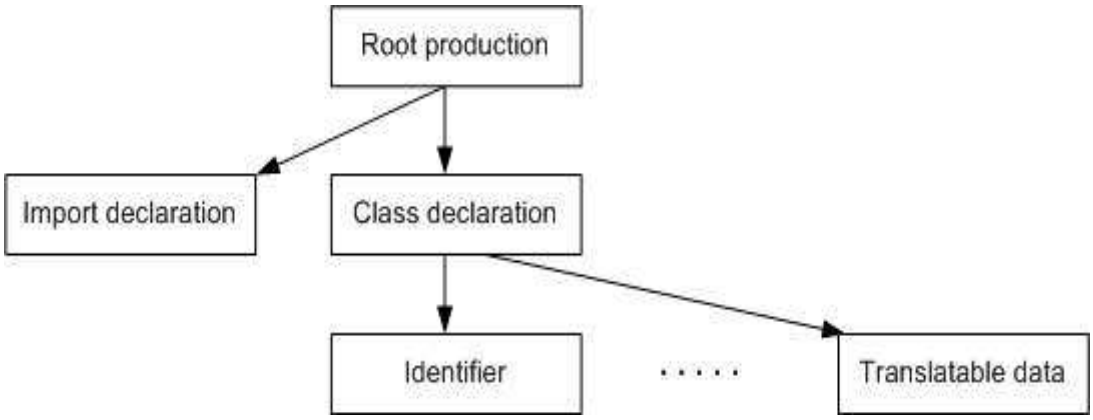
4.1 XML Localisation Interchange File Format (XLIFF)

XLIFF is an emerging XML specification aimed at the localisation industry. It has been adopted, and is currently under review by the standards organisation OASIS. The purpose of XLIFF is to standardise the format in which translation information is held in transmission between applications and/or organisations. It allows the mark-up of localisable data into a common, globally understandable format that enhances the larger translation process by providing data format transparency between interoperating parties. The XLIFF specification is available online at: http://www.oasis-open.org/committees/xliff/documents/cs-xliff-core-1.1-20031031.htm. This research will use XLIFF as the standard for representing translation information.

4.2 Java Compiler Compiler (JavaCC) & JJTree

JavaCC is a tool that takes as input a grammar and creates a generic parser that, when given a source document, will evaluate the syntactic correctness of the same. JJTree is a pre-processor tool accompaniment to JavaCC. JJTree creates a tree representation of a source document parse when used in conjunction with JavaCC. The tree can then be traversed and evaluated node by node at runtime. As nodes are encountered that correspond to translatable information they can be handled. The tree can also be unparsed, i.e., written out to a file node by node, hence recreating the original document. This allows for intelligent document alteration at runtime.

Below is a diagram that outlines how a Java programming language document might be represented as a parse tree (a full parse would probably contain hundreds of nodes). The root production is present in all JavaCC grammars. Child elements of this production essentially represent the contents of the document.



▪ Figure 2 – Parse tree

4.3 XSL Transformations (XSLT)

XSLT is one part of the Extensible Stylesheet Language (XSL) family of specifications for XML document transformation and presentation. The other members are: XML Path Language (XPATH) and XSL Formatting Objects (XSL-FO). XSLT is a language for transforming one XML document into another XML document [XSL Transformations, 1999, section: Abstract], or into another format completely, such as HTML, PDF, or SQL statements. XSLT files contain templates which are matched to elements using patterns (expressed in XPATH). These templates define change rules to apply to the source XML tree (matched using XPATH) such as renaming and reordering XML elements. A transformed XML file can be completely unrecognisable from the original depending on the transformations used. Below is a simple XML file pre and post transformation with the XSLT file that was used to facilitate the change. This is a simple example but XSLT contains many directives and functions to create more complex transformations.

```
Original:    <?xml version=”1.0” encoding=”UTF-8”?>
             <prescription>
                   <medicalPrescription>
                         <id>123456789</id>
                         <name>John Doe>
                         <address>57, Nowhere Avenue</address>
                   </medicalPrescription>
             </prescription>

Transformed: <?xml version=”1.0” encoding=”UTF-8”?>
             <prescription>
                   <medicalPrescription>
                         <id>123456789</id>
                         <name>John Doe>
                         <address>HELLO</address>
                   </medicalPrescription>
             </prescription>

Transform:   <?xml version="1.0" encoding="iso-8859-1"?>
             <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
                   <xsl:output method="xml" indent="yes"></xsl:output>
```

```
                  <xsl:template match="/"><xsl:apply-
templates/></xsl:template>
                  <xsl:template match="//*">
                        <xsl:copy>
                              <xsl:apply-templates/>
                        </xsl:copy>
                  </xsl:template>
                  <xsl:template match="//address">
                  <xsl:element name="address">
                        <xsl:value-of select="HELLO"/>
                  </xsl:element>
                  </xsl:template>
            </xsl:stylesheet>
```

Instruction for the XSLT processor to find all elements named 'address' (denoted by the XPATH statement //address).

Instructs the processor to set the value of the contents of the elements found (<address> elements) to 'HELLO'.

5 Case 1 – Grammar Based Parser Generator

5.1 Objective

The objective is to automatically generate a parser from a given grammar. The purpose of using parser technology is to allow the document type to be described as a grammar, create the parser once, and use the same parser to process all documents of that type. The parser created must be capable of the following:

- Examine a source document to identify the data contained within which is translatable.
- Create an XLIFF document with the translatable text contained.
- Create a skeleton document. Post translation, the translated text must be merged with this skeleton document to recreate the format of the original. The skeleton file is similar to the original except that, in place of the translatable text extracted to the XLIFF document, there are markers inserted that are used to merge with the translated text.

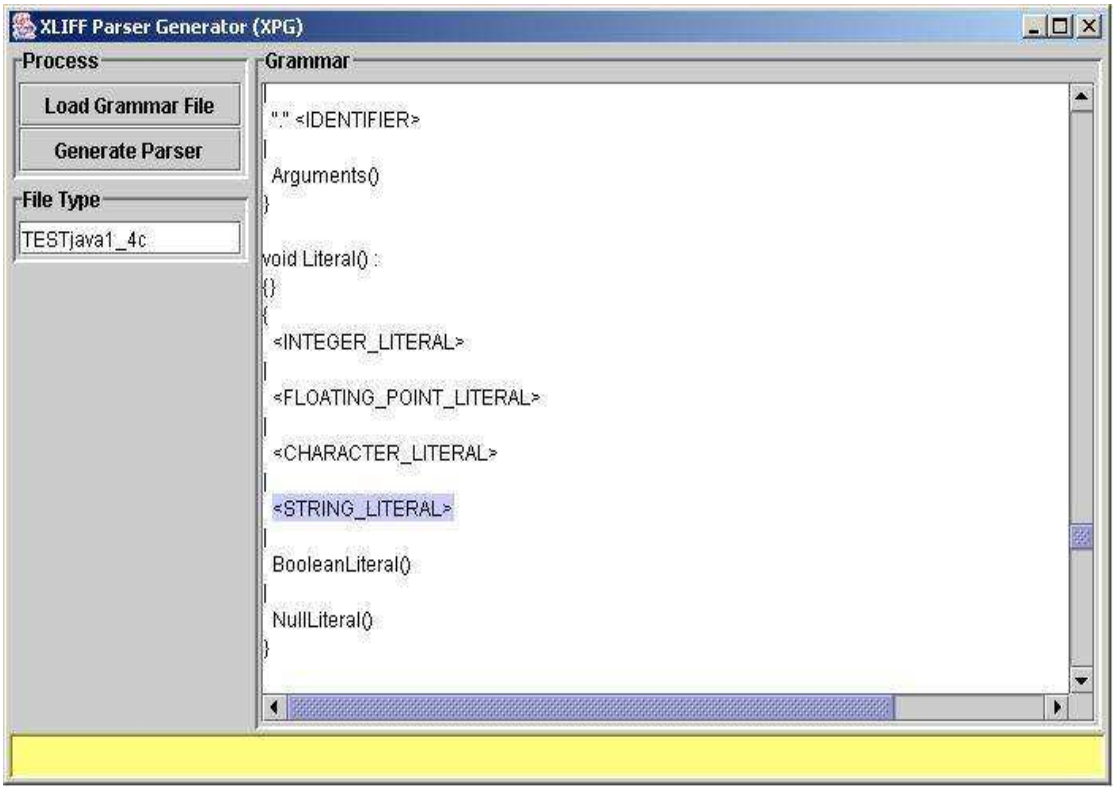
5.2 Parser Creation

Creating a parser for a specific grammar involves the steps outlined below.

1. Identify, or create a grammar representation of the document type in question.
2. Identify, within the grammar, the token that defines the data that will need to be extracted for translation. For example, the `'<STRING_LITERAL>'` declaration within the `'literal()'` production represents a translatable string for a normal programming language. JJTree directives and processing instructions are inserted into the grammar based on the selection. This creates a JJTree grammar.
3. The pre-processor JJTree is run against the new grammar to create tree handling classes. These generated classes are used for such purposes as representing tokens encountered in the tree traversal as nodes of the parse tree. Also outputted by the JJTree tool is a regular JavaCC grammar file.
4. JavaCC is run against the new JavaCC grammar file from the previous step to produce the remaining files needed for the parser. The combination of classes now available constitutes the parser.

5.2.1 JJTree Directives and Processing Instructions

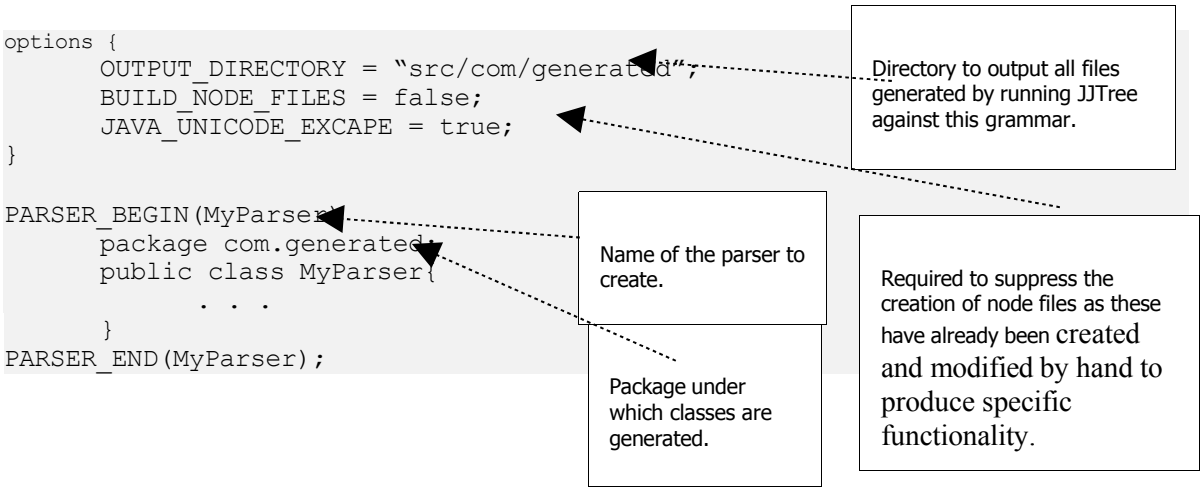
In step 2 above, JJTree directives are inserted into a grammar based on user input. These form the processing instructions that the generated parser uses to create the parse tree, identify translatable text, and consequently output the XLIFF and skeleton files. Using a graphical user interface the user loads a grammar and selects from it the token that represents translatable text. Figure 3 shows the user interface used, with a grammar loaded and the `'<STRING_LITERAL>'` token highlighted.



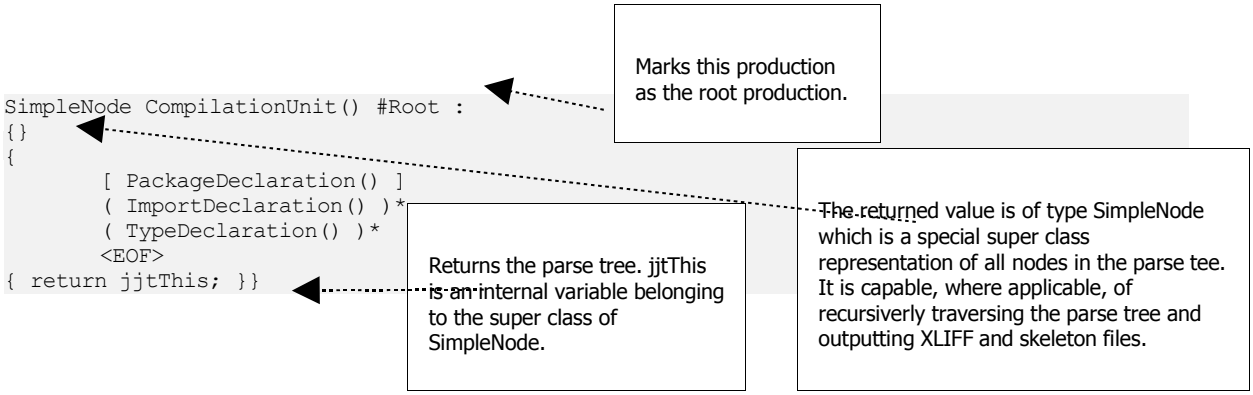
▪ Figure 3 – Parser Generator User Interface

With this selection made by the user the parser can be generated by hitting the `'Generate Parser'` button. This action inserts the required JJTree directives into the original grammar and processes this new grammar as described in the following paragraphs.

The options section, at the beginning of the grammar, is updated with processing instructions. The parser code section is updated with the name of the parser to create and the package to use for the generated classes.



The root production in a grammar is the production from which all other language productions hang (as illustrated by figure 2). JJTree directives must be inserted at the root production of the grammar to allow for the creation of the parse tree. The following code snippet shows a root production called `'CompilationUnit'`.



The SimpleNode class is a node super class that contains utility methods used when processing the parse tree node by node. Say for example that the user selected the `'<STRING_LITERAL>'` token as representing translatable text as illustrated in the user interface diagram above. The following code is inserted within the production that the selection belongs to.

```
void literal() :
{ Token tok; }
{
      <INTEGER_LITERAL>
      |
      <CHARACTER_LITERAL>
      |
      tok=<STRING_LITERAL>
           { jjtThis.setValue(tok.image); }
           { jjThis.setOffsets(tok.beginLine, tok.beginColumn, tok.endLine,
tok.endColumn); }
      |
      BooleanLiteral()
      |
      NullLiteral()
}
```

Allocates space for the tokens that are encountered as the parse tree is traversed.

Assigns the tok variable to the contents of the <STRING_LITERAL> token.

Set the value of this node.

Record exactly where in the document this token was encountered.

5.2.2 The SimpleNode Class

The SimpleNode class itself, which maintains node information (and hence translatable text information), and outputs the XLIFF and skeleton files, is described in the following code snippet.

```
public class SimpleNode implements Node{
      private String myValue;
      static{
            . . .
            ConstructXLIFFDocument.createElemType_body("body");
            . . .
      }

      public void setValue(String val){
            this.myValue = val;
      }

      public void setOffsets(int bgnLine, int bgnCol, int eLine, int
            . . .
      }

      public void dumpXLIFF(String outFile){
            . . .
      }

      public void dumpSkeleton(String outFile){
            . . .
      }
```

Node is an interface automatically generated by JJTree.

Statically construct the core pf the XLIFF document.

If this node represents translatable text then record the fact in the myValue variable.

Record acactly where in a document translatable text was encountered.

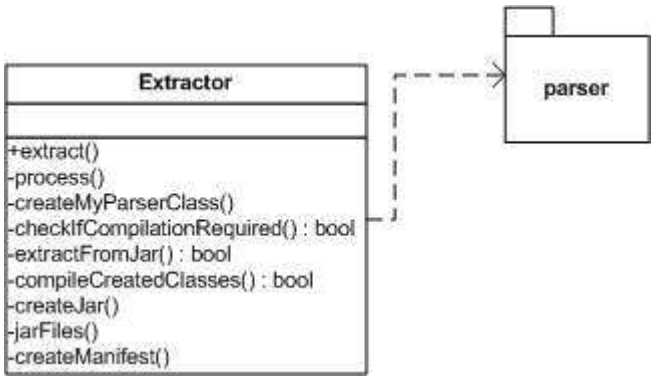
Prints tree stack to output. Recursively examines each node in the tree to determine if myValue != null, i.e., translatable text stored.

Output a skeleton file with XLIFF <trans-unit> IDs inserted.

5.3 Using the Parser

The parser is used by an extractor class whose purpose is to provide functionality for creating XLIFF and skeleton documents. The extractor class uses the parser to recursively traverse the parse tree created for the inputted document and outputs the relevant XLIFF & skeleton files.

The extractor class is described by the following diagram. It contains one publicly available method, extract. This method takes as parameters the URL of the source document, and the name of the parser to use. It then uses the private methods at its disposal to process the document.



▪ Figure 4 – Extractor class

The result of running the extractor class against a file using a previously create parser is a valid XLIFF document with translatable data suitably inserted, and a skeleton file.

```
XLIFF:      <trans-unit id="123456789">
                  <source>my translatable text</source>
            <trans-unit id="98734">
                  <source>yet more translatable text</source>
            </trans-unit>

Skeleton:   public static void main(String args[]){
                  String s1 = "<trans-unit id=123456789>";
                  System.out.println("<trans-unit id=98734>");
            }
```

5.4 Post translation

A typical XLIFF document will go through phases of translation until the data contained is fully translated. At the end of this process there will be an XLIFF document that contains original data and the data's resultant translation within the specified XML element '`<trans-unit>`'. The following is an example of a translated XLIFF document.

```
<trans-unit id="123456789">
      <source>hello</source>
      <target>bonjour</source>
</trans-unit>
<trans-unit id="436554">
      <source>how many</source>
      <target>combien</target>
</trans-unit>
```

A merge class takes the XLIFF document, and specifically the translations, and merges them with the skeleton file created during the extraction process, to produce a final translated document. This process is accomplished using XML processing technologies to process the XLIFF document and regular expression matching to process the skeleton document

6 Case 2 – XML Schema Based Transformer Generator

6.1 Objective

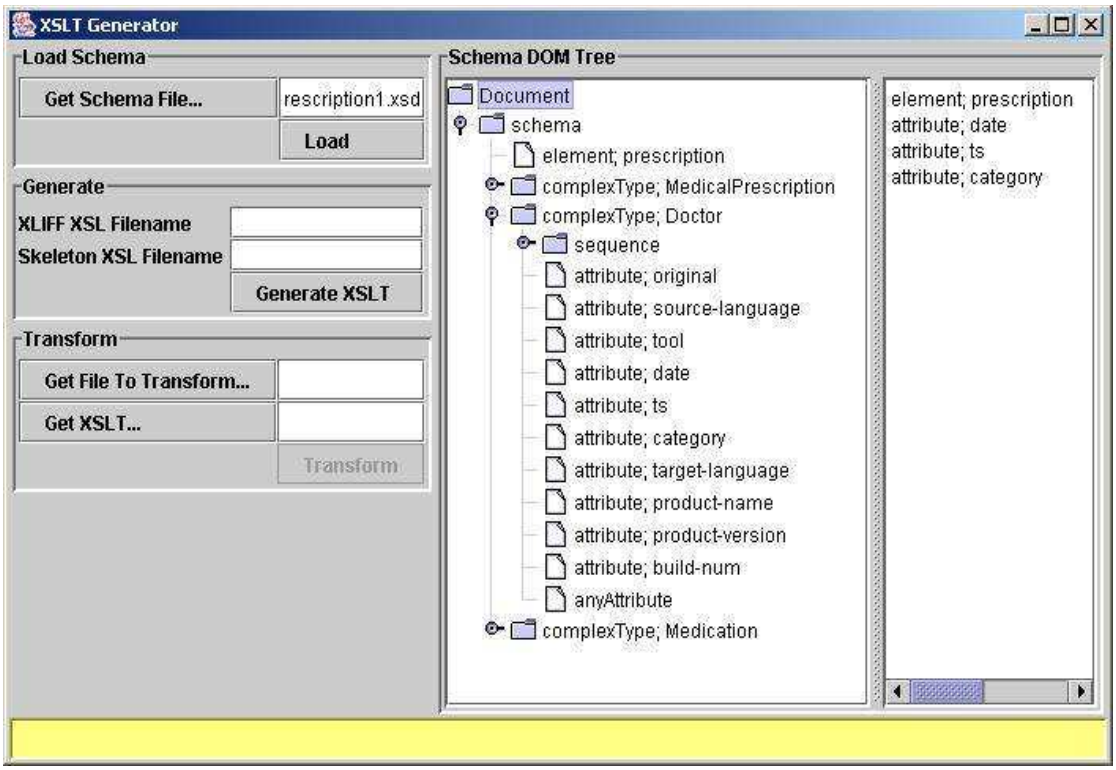
In the XML Schema based approach the objective is to identify XML elements and attributes, from an XML Schema, that are considered to contain translatable text. From this information two XSLT files are created. The first converts an XML document into an XLIFF document complete with translatable text suitably inserted. The second converts the original XML document into a skeleton used for merging.

6.2 Transformer Creation

Creating the two files used to process an XML document for localisation follows the following steps:

1. Identify within an XML Schema the elements and attributes that can contain translatable text.
2. Generate the necessary XSLT files based on these selections from the step above.

The task of identifying the elements and attributes within an XML Schema which can contain translatable text is simplified by using a graphical user interface. The user will load an XML Schema into the interface. This creates a tree structure of the information. The user can then navigate to and select the required elements and attributes. The following diagram shows the user interface.



▪ Figure 5 – Transformer Generator User Interface

6.2.1 XSLT Directives and Processing Instructions

Based on the selections made in the user interface, two transformation files are created; one for creating the XLIFF, the other for creating the skeleton file. The following code snippet shows the XSLT generated for converting an XML file conforming to the format described by a given XML Schema into XLIFF based on the user selections. The XSLT generated for creating the skeleton file will be similar.

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.0">
      <xsl:output method="xml" indent="yes"></xsl:output>
      <xsl:template match="/">
            <xlf:xliff version="1.1" . . . >
                  <xlf:file datatype="rtf">
                        <xlf:header/>
                        <xlf:body>
                              <xsl:for-each select="//Doctor/@category">
                                    <xlf:trans-unit id="{generate-id()}">
                                    <xlf:source>
                                          <xsl:value-of select="."/>
                                    </xlf:source>
                              </xsl:for-each>
                        </xlf:body>
                  </xlf:file>
            <xlf:xliff>
      </xsl:template>
</xsl:stylesheet>
```

Create this XLIFF construct.

Examine each element called 'Doctor' with attribute 'category'.

Insert the value into this XLIFF construct.

6.3 Using the Transformations

1. Run the first XSLT file against the original XML document to create an XLIFF file.
2. Run the second XSLT file against the original XML document to create a skeleton file.

Having created the initial stylesheets, any XML document conforming to the XML Schema in question can be transformed to create the necessary XLIFF and skeleton files.

6.4 Post Translation

See section 5.3. The merger application for this is the same as that for the grammar based approach.

7 Conclusion

The prototype developed by this research has successfully translated both programs of a recognisable grammar, and XML files from a previously processed XML schema. Using language parsers and XSLT transformations to enable efficient and automatic handling of multiple document types by localisation vendors improves the translation process efficiency. No longer must vendors spend time examining and extracting data from every document that requires translation. This research shows that documents can be processed by type rather than individually. Also, as a larger set of parsers and/or transformation files is built up over time, the less likely it is that the vendor will encounter an unsupported format. This is particularly true for proprietary file formats that are unlikely to change over time.

Reference

WorldLingo (2004)
Glossary of Terms, Retrieved March 2004, from WorldLingo website
http://www.worldlingo.com/resources/glossary.html

XSL Transformations (1999)
XSL Transformations (XSLT), Version 1.0. Retrieved March 2004 from W3C website
http://www.w3.org/TR/xslt

Rubric (2000)
R*egina Born, Computer Associates Tom Shapiro, Rubric,* Streamline The Localization Process, Software Business, March 2000

8th Annual International Localisation Conference (2003)
Tony Jewtushenko, Principle Product Manager, Oracle and chair OASIS TC XLIFF, "XLIFF – the XML based standard for Localisation File Format", 8th Annual International Localisation Conference, Localisation Research Centre, University of Limerick, 2003

XLIFF (2004)
http://www.xliff.org