Amazigh Converter based on WordprocessingML

Fadoua Ataa Allah Jamal Frain

CEISIC, IRCAM, Morocco
{ataaallah, frain}@ ircam.ma

Abstract

Since the creation of the Royal Institute of Amazigh Culture, the Amazigh language is undergoing a process of standardization and integration into information and communication technologies. This process is passing through several stages, after the writing system stabilization, the encoding stage and the development of appropriate standards for the keyboard layout; the stage of computational linguistics is undertaking. Thus, in the aim to save the Amazigh cultural heritage, many converters, allowing the Tifinaghe ANSI-Unicode transition and Arabic-Latin-Tifinaghe transliteration, have been developed. However, these converters could not assure the file layout and the processing of all documents' parts. To overcome these limitations, the new WordprocessingML technology has been used.

Keywords: Amazigh language, transcoding, transliteration, WordprocessingML technology, ANSI, Unicode, Tifinaghe, Arabic and Latin scripts.

1. Introduction

The integration of the Amazigh language in Information Technology and Communication (ICT) has become a necessity to promote the Amazigh language. However, this integration was confronted by several challenges, including those related to standardization and language planning.

To let the Amazigh language supporting and conveying knowledge, firstly, a writing form and an alphabetic system have been established. Secondly, based on a linguistic description of the most widely spoken varieties of the Amazigh language, a spelling system has been stabilized (Ameur *et al.*, 2004). Then, a stage of character encoding has been undertaken. However, the difficulty in these steps is to achieve generic solutions in limited time to allow the integration of Amazigh into the Moroccan educational system in 2003. Thus, the native Amazigh writing system encoding went through two steps: ANSI then Unicode encoding (Moukhlis and Ouniam, 2006).

Promoting Amazigh culture involves the maintenance and preservation of literary heritage and the dissemination of writing Tifinaghe on all media. To this end, Amazigh converters have been developed:

- A command line transcoder for ANSI-Unicode encodage and an Arabic-Latin-Tifinaghe transliterator, providing only the treatment of .txt files (Ataa Allah and Boulaknadel, 2011).
- A transcoder and transliterator with graphical interfaces based on interob API for processing the files .txt, .rtf, .doc and .docx (Ataa Allah *et al.*, 2013).

However, all these converters have known technical limitations that can be summarized in the following points:

- Complexity of an automatic identification of the used fonts, which is necessary required for transcoding Tifinaghe script from ANSI to Unicode, especially in multilingual documents.
- Simple search without combination of free content criteria.
- The loss of the document format.
- The difficulty of processing headers, footers, footnotes at the bottom of pages.
- Crash or slow processing of long files.

To overcome these limitations, we have decided to develop a new converter based on WordProcessingML technology that will handle files particularly Word format in more efficient and simpler way than interob API technology. Furthermore, we propose new code for processing files' header, footer and footnote.

The remainder of this paper consists of four sections. In Section 2, we present a historical overview and the writing systems of the Amazigh language. Then, we introduce, in Section 3, the WordprocessingML technology, especially the needed structures. Finally, before to conclude in Section 5, we describe, in Section 4, the elaborated tool.

2. Amazigh language

2.1. Amazigh language history

Amazigh is the native language of North Africa. It is also known by the name of "Berber" and the local name "Tamazight". This language is present from Morocco to Egypt passing through Algeria, Tunisia, Niger and Mali. It is spoken by tens of millions of people as non-standardized dialects.

In Morocco, there are three main varieties of the Amazigh language: Tarifite in the North; Tamazight in the Center, the Middle Atlas and a part of High Atlas; and Tachelhite in the South, South-west of High Atlas, the Anti-Atlas and Sous. These varieties were primarily employed in oral communication. However, in order to preserve the Amazigh language, it is important to transit from orality to literacy and to upgrade the language, then to integrate the Amazigh language into the information and communication technologies.

Although the Amazigh language was primarily an oral tradition, the Amazigh language has, since antiquity, its own writing system called "Libyco-Berber" (Tifinaghe in Amazigh). This system dates back more than 40 centuries (Hachid, 2000; Skounti et *al.*, 2003). However, the appearance form of its signs has been undergoing many

modifications: since its inception "the Libyan" to the neo-Tifinaghe in the late sixties and Tifinaghe IRCAM-in 2001 (Ameur *et al.*, 2004).

2.2. Tifinaghe-IRCAM graphical system

Since February, 11th, 2003, Tifinaghe-IRCAM has become the official graphic system for writing Amazigh in Morocco (Ameur *et al.*, 2004). This system contains:

- 27 consonants including: ⴹ, ⵅ, ⵅⵯ, ⵃ, ⴻ, ⵀ, ⴽ, ⴽⵯ, ⵁ, ⵄ, ⵄ, ⵅ, ⵇ, ⵉ, ⵍ, ⵎ, ⵏ, ⵓ, ⵔ, ⵚ, ⵜ, ⵟ, ⵠ, ⵣ;
- 2 semi-consonants: ⵢ and ⵡ;
- 4 vowels: three full vowels ⴰ, ⴷ, ⵄ and neutral vowel (or schwa) ⵄ which has a rather special status in Amazigh phonology.

No particular punctuation is known for Tifinaghe. IRCAM has recommended the use of the international symbols.

2.3. Amazigh encoding

To allow the Amazigh people to communicate in their own language and follow at the same time the technological evolution, it is necessary to ensure a wide diffusion and a linguistic analysis of digital document content. Thus, a digital transcription system has been created, and several efforts have been undertaken to encode Tifinaghe in Unicode/ISO 106461 (Moukhlis and Ouniam, 2006). Nevertheless, this process has taken a long time than what the integration of the Amazigh language into the Moroccan education system was allowed. To resolve this problem, the ANSI encoding supported by fonts has been used (for more details about Tifinaghe ANSI and Unicode encoding the reader can refer to (Ataa Allah *et al.*, 2013).

3. WordprocessingML

Unlike its predecessors, the highly anticipated Microsoft Office 2007document format 'OpenXML', is a fully open standard. It is based on open specifications and uses open standards like ZIP, XML, XMLSchema, PNG, ..., which make it particularly interesting for developers since these standards are supported by most technologies such as .NET, Java, PHP, and C. Moreover, OpenXML adopts the separation concept of content and presentation: each kind of data (text, image, multimedia, etc.) is stored independently of the styles and presentation information.

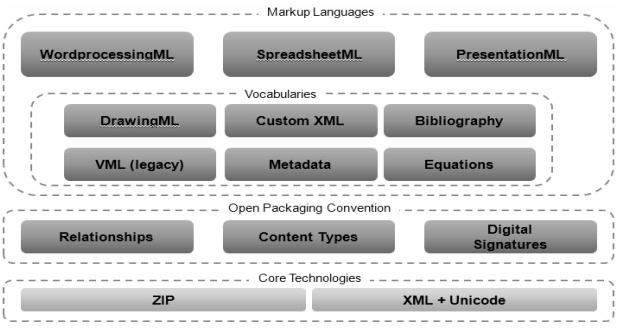

Fig. 1: Open XML specification

Furthermore based on WordprocessingML and XML technologies, Word file creation and manipulation became easier and do not require the use of Interop, which means that the user does not need to install the Word software to manipulate the document.

In the remaining of this section, we describe the WordML document structure, and explain the operations of reading, writing and updating a Word document[1].

3.1. Presentation

The introduction of XML was marked as a native format of Word documents. Any Word document can be opened in Word and saved as XML. This new format, called WordML, offers many opportunities to generate and process Word documents through the WordprocessingML markup.

The WordML package has the form of a zip archive containing a collection of compressed files, where each file is a part of the package. To display their contents, the .docx files can be renamed by .zip, or use Zip utility to open them.

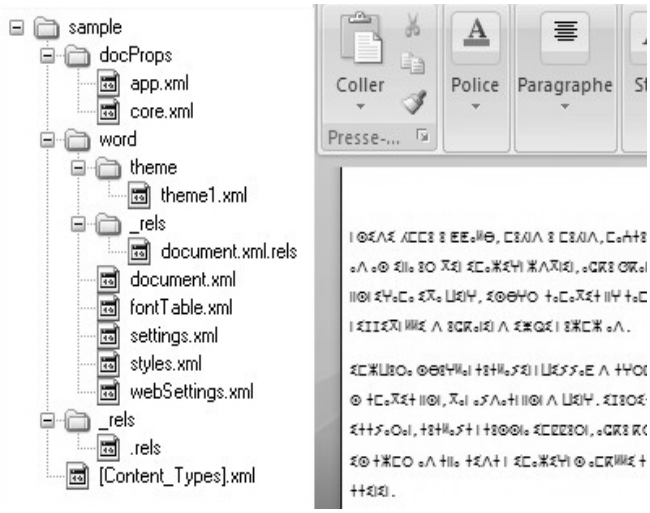

Fig. 2: WordprocessingML structure

To be effectual, a document must contain at least the following parts:

- word / document.xml: the main part of the file (content).
- _rels / rels.xml: file of the main relations.
- [Content_Types].xml: Types of files in the package.

3.2. Environment configuration

To read and write packages (files in OpenXML format), the SDK (Software Development Kit) OpenXML can be used. Thus, two references should be added in the project assembly or the solution, which are:

- Microsoft.Office.DocumentFormat.OpenXml;
- WindowsBase.

However, the following assembly directives are also required to compile the source code:

- DocumentFormat.OpenXml;
- DocumentFormat.OpenXml.Packaging;
- DocumentFormat.OpenXml.Wordprocessing.

[1] http://msdn.microsoft.com/en-us/library/office/cc850833.aspx, consulted on September 2013.

The SDK (Software Development Kit) provides a main advantage in simplifying the access to different parts of OpenXML documents.

3.3. File writing

3.3.1. WordprocessingDocument Object Creation

In the Open XML SDK, the WordprocessingDocument class represents a Word document package. To create a Word document, first a WordprocessingDocument class instance need to be created and populated with parts, then the Create (String, WordprocessingDocumentType) method should be called. Once, the Word document package is created the user can add parts to it. To add the main document part, the AddMainDocumentPart() method of the WordprocessingDocument class is called. Having done that, the document structure and text could be added.

```
// Create a document by supplying the filepath.
using (WordprocessingDocument wordDocument =
WordprocessingDocument.Create(filepath,
WordprocessingDocumentType.Document)){
  // Add a main document part.
  MainDocumentPart mainPart =
  wordDocument.AddMainDocumentPart();
  // Create the document structure and add some text.
  mainPart.Document = new Document();
  Body body = mainPart.Document.AppendChild(new
  Body());
  Paragraph para = body.AppendChild(new
  Paragraph());
  Run run = para.AppendChild(new Run());
  run.AppendChild(new Text("ⵜⴰⵎⵔⴰⴳⵜ "));
}
```

Fig. 3: Sample code of WordprocessingDocument Object Creation

3.3.2. WordProcessingML Document Structure

The basic structure of a WordProcessingML document consists of the document and the body element, followed by one or more block level elements such as p, which represents a paragraph. This later contains one or more r elements. The r stands for run, which is a region of text with a common set of properties, such as formatting. A run includes one or more t elements that contain a range of text.

3.4. Font structure

The following text from the ISO/IEC 29500 specification can be useful when working with rFonts element. This element specifies the fonts which shall be used to display the text contents of this run. Within a single run, there may be up to four types of attributes (ASCII, High ANSI, Complex Script “cs”, East Asian). Each one of them allows using a unique font.

Although it is in the same run, the contents could be in different font faces by specifying a different font for different attributes, as shown in fig. 4.

```
<w:r>
  <w:rPr>
  <w:rFonts w:ascii="Times"w:hansi="Tifinaghe-
   IRCAM"/>
  </w:rPr>
  <w:t> Merci : </w:t>
  <w:t> ⵜⴰⵎⵔⴰⴳⵜ </w:t>
</w:r>
```

Fig. 4: Example of rFont attributes

3.5. Search a text

OpenXML SDK allows looking for a text very simply. Once the package (docx file) is opened, the document contents are reloaded, according to the XML Stream, as follow:

```
using (WordprocessingDocument wdDoc =
WordprocessingDocument.Open(url, true)) {
    const string wordmlNamespace =
     "http://schemas.openxmlformats.org/wordprocessing
     ml/2006/main";
     NameTable nt = new NameTable();
     XmlNamespaceManager nsManager = new
     System.Xml.XmlNamespaceManager(nt);
     nsManager.AddNamespace("w", wordmlNamespace);
     XmlDocument xdoc = new XmlDocument(nt);
    XmlDocument xdocheader = new XmlDocument(nt);
    xdoc.Load(wdDoc.MainDocumentPart.GetStream());
    .....
```

Fig. 5: XML Stream of document contents

Then, we can retrieve a document portion with a specific font by using an XPath query as illustrated in fig. 6.

```
XmlNodeList hiddenNodes =
xdoc.SelectNodes("//w:r/w:rPr/w:rFonts[@w:ascii='Font
ici']/../../w:t", nsManager);//requete xpath
int i = 0;
string font = "";
foreach (System.Xml.XmlNode hiddenNode in
hiddenNodes){
   Console.WriteLine(hiddenNode.InnerText);  }
```

Fig. 6: Sample code for retrieving a specific font

3.6. Font change

To change the paragraph font that is stored in the **rFonts** properties of the **pPr** element, represented by the RunProperties class in Open XML SDK, the following code is used:

```
// Open a Wordprocessing document for editing.
 using (WordprocessingDocument package =
WordprocessingDocument.Open(fileName, true)) {
  List<RunProperties> runProps =
  package.MainDocumentPart.Document.Descendants<R
  unProperties>().ToList();
    foreach (RunProperties rp in runProps) {
      if (rp.RunFonts.HighAnsi == "Tifinaghe-IRCAM"){
        rp.RunFonts = new RunFonts(){
          ComplexScript="Tifinaghe-Ircam Unicode"}; }}
   package.MainDocumentPart.Document.Save(); }
```

Fig. 7: Sample code for changing WordprocessingDocument font

3.7. Header, footer and footnote change

To modify either a header, footer or a footnote part of a document the WordprocessingML documentation advises to delete the part than replace it by a new one containing the desired updates. However, this method does not allow keeping the part layout. To overcome this limitation, we propose the new approach illustrated by fig. 8, and structured as follow:

- Get the specified part from MainDocumentPart object.
- Load the XML scheme of this part into an XmlDocument instance.
- Search the specified text to update by using the XPATH query.
- Replace the returned object by the desired text.
- Save the XML scheme.

```
FootnotesPart footnotesPart =
wdDoc.MainDocumentPart.FootnotesPart;
 if (footnotesPart != null)  {
  Hashtable liste_footnote = new Hashtable();
  IEnumerable<Footnote> footnotes =
  footnotesPart.Footnotes.Elements<Footnote>();
    foreach (Footnote footnote in footnotes)
   {
     String txtXml = footnote.InnerXml;
     byte[] byteArray =
    Encoding.BigEndianUnicode.GetBytes(txtXml);
    MemoryStream stream = new
    MemoryStream(byteArray);
    xdocheader.Load(stream);
    XmlNodeList footerNodes =
    xdocheader.SelectNodes("//w:r/w:n[@w:ascii='" +
    font_sel + "']/../../w:t", nsManager);
      foreach (System.Xml.XmlNode hiddenNode1 in
      footerNodes) {
      try{
      Console.WriteLine( hiddenNode1.InnerText);
      ……
      }catch (Exception c){ }}}
```

Fig. 8: Sample code for changing WordprocessingDocument header, footer and footnote

4. Amazigh Converter

Through its existence, the Amazigh language has known different forms of writing: Latin supported by the International Phonetic Alphabet, Arabic script, and Tifinaghe character based on ANSI and Unicode encoding. In the aim of allowing users to read or write in a suitable form, and to save the Amazigh literature heritage in a standard unique form, a command-line converter has been developed (Ataa Allah and Boulaknadel, 2013). This tool ensures an automatic conversion from one form to another. However, it has some limitations such as do not having menu-driven and graphical user interfaces, do not processing rich text format, and do not dealing with multilingual text, especially when the Amazigh language is written by ANSI encoding which makes the distinction between Tifinaghe script and other scripts relies on fonts. To overcome these shortcomings a first version of a desktop converter based on the interob API was developed (Ataa Allah *et al.*, 2013). However, this later could not process the file header, bottom, or footnotes nor preserve the file layout. Thus, we have opted to integrate the WordprocessingML technology to deal with these limitations.

4.1. Technical architecture

The technical architecture of the desktop converter is based on the .Net technology and an implementation of the Model-View-Controller (MVC) pattern that allows to separate application data model and user interface views into different components.

4.2. Functional architecture

As summarized in Fig. 9, the desktop converter consists of two processes: Transcoder and transliterator.

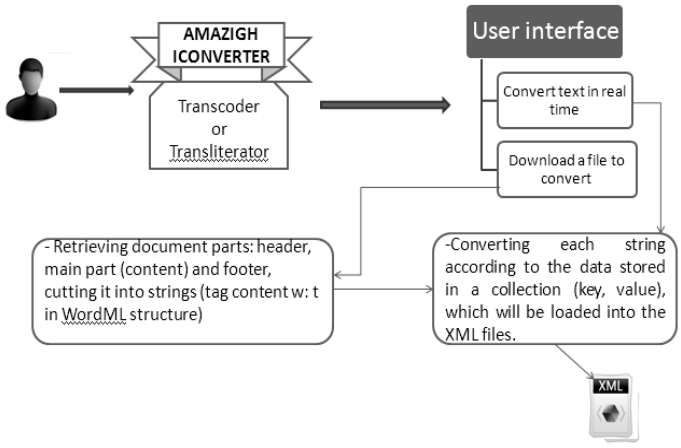

Fig. 9: Amazigh converter architecture

4.2.1. Transcoder

This process allows shifting the ANSI representation of Tifinaghe to Unicode representation for the content of a file in one of the following format: .txt, .rtf, .doc, and .docx.

In the case of a file conversion, based on the WordprocessingML technology, the system detects the used fonts in this file. Then, converts all the Tifinaghe characters written on ANSI encoding that are defined by the Amazigh ANSI fonts, such as Tarommit and Tifinaghe-IRCAM, to Unicode.

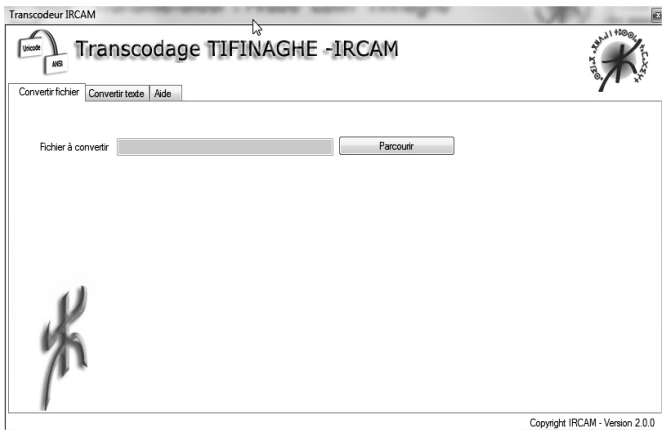

Fig. 10: Transcoder interface for converting a file

In the case of converting the content of a text area, the transcoder enables a real time encoding conversion. The input content can be pasted or typed, and the output can be copied or saved into a text file.

4.2.2. Transliterator

The transliterator process aims to substitute the script of a text to another, while conserving the phonetic pronunciation of its words. This process is based on direct mapping between the pairs of scripts (Latin, Tifinaghe Unicode) and (Arabic, Tifinaghe Unicode). In the Latin - Tifinaghe Unicode mapping the IRCAM correspondences are used by default (Ameur *et al.*, 2004). While, the phonetic and Stroomer correspondences (Stroomer, 2001) are also available. In the Arabic - Tifinaghe Unicode mapping, there are more constrained rules to use the IRCAM correspondences. These constraints depend mainly on the cursiveness of the Arabic language, the phonetic pronunciation, and the use of Amazigh and Arabic vowels. Thus, some Arabic - Tifinaghe correspondences have been adapted, and orthographic rules have been specified mainly on the transliteration from Arabic script into Tifinaghe one (for more details, the reader can consult (Ataa Allah and Boulaknadel, 2012)).

Fig. 11: Transliterator interface

Furthermore, the tool enables the user to set his/her proper correspondence mapping table that could be saved for another reuse (cf. Fig. 12). Once the new mapping table is saved, it will appear in the dropdown list.

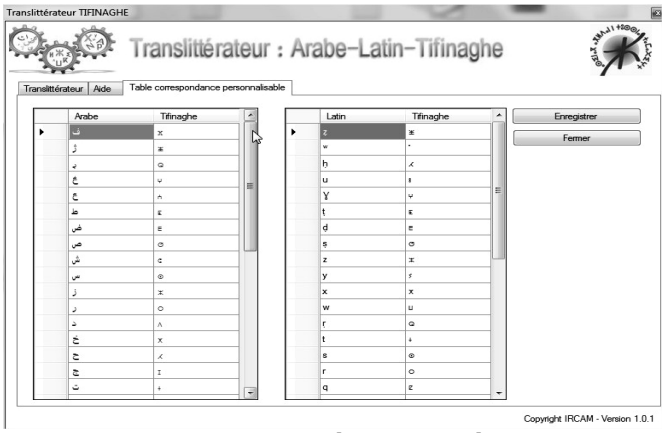

Fig. 12: Correspondence setting layout

5. Conclusion

In the aim to promote the Amazigh language and to preserve its literary heritage, this paper has presented a desktop converter based on the new WordprocessingML technology. This tool enables the Tifinaghe ANSI-Unicode transcoding and the Arabic-Latin-Tifinaghe transliteration even in real time; and retain the file layout and the processing of all documents' parts; particularly headers, footers, and footnotes.

References

Ameur, M., Bouhjar, A., Boukhris, F., Boukouss, A., Boumalk, A., Elmedlaoui, M., Iazzi, E. M. and Souifi, H. Initiation à la langue amazighe, IRCAM, 2004.

Ataa Allah, F. and Boulaknadel, S. "Convertisseur pour la langue amazighe : script arabe - latin – tifinaghe", The 2nd Symposium International sur le Traitement Automatique de la Culture Amazighe, SITACAM 2011, 6-7 May 2011, Agadir, Morocco, 2011.

Ataa Allah, F. and Boulaknadel, S "Toward computational processing of less resourced languages: Primarily experiments for Moroccan Amazigh language", in Text Mining. Rijeka: InTech, pp. 197-218, november 2012.

Ataa Allah, F., Frain, J. and Ait Ouguengay, Y « Amazigh Language Desktop Converter », Actes du 3ème Symposium International sur le Traitement Automatique de la Culture Amazighe, qui a eu lieu à Beni Mellal le 2-4 May 2013.

Hachid, M. "Les premiers berbères", Entre Méditérranée, Tassili et Nili. Aix-en-Provence-Alger : Edisud-Ina-Yas, 2000.

Moukhlis, M. and Ouniam, L. Bulletind'information de l'Institut Royal de la Culture Amazighe, n° 5&6, June 2006.

Skounti, A., Lemjidi, A. and Nami, E. M. Tirra aux origines de l'écriture au Maroc, IRCAM, 2003.

Stroomer, H. Textes berbères des Guedmioua et Goundafa (Haut Atlas, Maroc), basés sur les documents de F. Corjon, J.-M. Franchi et J. Eugène, Edisud, 2001.