docx2tex: Word 2007 to TeX

Krisztián Pócza
Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary
kpocza (at) kpocza dot net
http://kpocza.net/

Mihály Biczó
Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary
mihaly.biczo (at) t-online dot hu
http://avalon.inf.elte.hu/personal/hdbiczo/

Zoltán Porkoláb
Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary
gsd (at) elte dot hu
http://gsd.web.elte.hu/

Abstract

Docx2tex is a small command line tool to support users of Word 2007 to publish documents when typography is important or only papers produced by TeX are accepted. Behind the scenes, docx2tex uses common technologies to interpret Word 2007 OOXML format without utilizing the API of Word 2007. Docx2tex is published as a free and open source utility that is accessible and extensible by everyone. The source code and the binary executable of the application can be downloaded from http://codeplex.com/docx2tex/. This paper was originally written in Word 2007 and later converted to TeX using docx2tex.

1 Introduction

There are two general methods to produce human readable and printable digital documents:

1. Using a WYSIWYG word processor
2. Using a typesetting system

Each of them has its own advantages and disadvantages; therefore each of them has many use cases where one is better than the other.

WYSIWYG [1] is an acronym for the term *What You See Is What You Get* that originates from the late '70s. WYSIWYG editors are usually favored by everyday computer users whose aim is to produce good-looking documents in a fast and straightforward way exploiting the rich formatting capabilities of such systems. WYSIWYG editors and word processors ensure that the printed version of the document will be the same as the document that is visible on the screen during editing. The first WYSIWYG word processor called Bravo was created at Xerox by Charles Simonyi, who is the inventor of intentional programming. In 1981 Simonyi left Xerox and joined Microsoft where he created Microsoft Word [2, 3], the first and still the most popular word processor. Word is capable of producing simple and also complex documents, including those with many mathematical symbols. Another important feature of Word is *Track Changes* that supports team work. Using Track Changes any of the team members can modify the document while these modifications are tracked and can be accepted or refused by the team leader.

Typesetting is the process of putting characters of different types in their correct place on the paper or screen. Before electronic typesetting systems became widely used, printed materials had been produced by compositors who worked by hand or using special machines. The aim of typesetting systems is to create high-quality output of materials that may contain complex mathematical formulae and complex figures. Similarly, electronic typesetting systems follow this goal and produce high quality, device independent output. The most popular

typesetting system is TEX [4] created by Donald E. Knuth. TEX is mainly used by researchers and individuals whose aim is to achieve the best quality output without sacrificing platform or device independence. The users of TEX use a special and extensible DSL (Domain Specific Language) that was designed to solve complex typesetting problems, produce books containing hundreds of pages, and more.

There is a big gap between these systems because each tries to satisfy different demands, namely: produce common documents quickly even in a group setting vs. achieve the best quality and typographically correct printout. To bridge this difference there are both commercial and non-commercial tools that support conversion from Word or other WYSIWYG formats to TEX (and back). The first direction, converting from WYSIWYG (Word) formats to TEX, has perhaps more frequent usage because many users edit the original text in Word for the sake of simplicity and efficiency, and later convert it to TEX by hand in order to ensure quality.

The problems with present conversion applications include the following:

1. many of them are available only as proprietary tools;

2. thus they have limitations (running times or page limit) when not purchased;

3. they support only the old, binary Word or Rich Text document format (.doc, .rtf); and/or

4. they use the Word's COM API to process documents, which makes them complex.

In this paper we present an open source and free solution that is capable of handling the new and open Word 2007 .docx format natively by using standard technologies without using the COM API of Word and without even installing Word. In this article the current features are presented along with further development directions.

2 Existing solutions

It has always been challenging to convert proprietary, binary or any other document formats to TEX. Because Word is the most common editor, many tools try to convert from Word documents. One of these tools is the proprietary *Word2TeX* [5], which makes Word capable of saving documents in TEX format. This tool is embedded into Word, has an evaluation period and can be purchased in different license packages. A similarly featured tool named *Word-to-Latex* [6] does not provide sources, though it is available at no cost.

Another possibility is to use OpenOffice.org [7], which is capable of reading Word documents and also saving them in TEX format. It is a free and open source application; however, it interprets the binary data of Word documents.

Rtf2latex2e [8] is the most recent solution that translates .rtf files to TEX. It is a free and open source application.

3 Technology

In this section we will enumerate and then briefly review the technologies that are used in docx2tex and show how they cooperate.

The technologies used are the following:

1. Office Open XML (ECMA 376 Standard [9], recently approved as an ISO Standard), the default format of Word 2007. In brief: OOXML.

2. Microsoft .NET 3.0 (CLI is ECMA 335 Standard [10] and ISO/IEC 23271:2006 Standard [11]).

3. ImageMagick [12] to convert images.

OOXML files are simply XML and media files compressed using ZIP. Docx2tex uses Microsoft .NET 3.0 to open and unzip OOXML Word 2007 (.docx extension) documents. Microsoft .NET 3.0 has some special classes in the *System.IO.Packaging* namespace that facilitate opening and unzipping OOXML files and also provide an abstraction to represent the included XML and media files as packages. The operations performed by this component are described by the object line called *OOXML depackaging* in Figure 1.

The most important component of docx2tex is the *Core XML Engine* that implements the basic conversion from XML files to TEX. The Core Engine is responsible for reading and processing the XML data of the OOXML documents that is served by the OOXML depackaging component. The Core Engine identifies parts of the OOXML document and processes the contents of these parts (paragraphs, runs, tables, image references, numberings, . . .). It is not responsible for processing parts of the OOXML document that are available through a relation. For those, docx2tex has a set of internal *helper functions* that are responsible for driving the processing of related entities such as image conversion, special styling and resolving the properties of numbered lists.

When an image reference is found in the XML, ImageMagick is called to produce EPS files from the original image files. The resulting EPS files can be embedded easily into TEX documents.

We support the exact output produced by Word 2007; other output variations saved by third party applications that may differ from the ECMA 376 Standard are not supported.
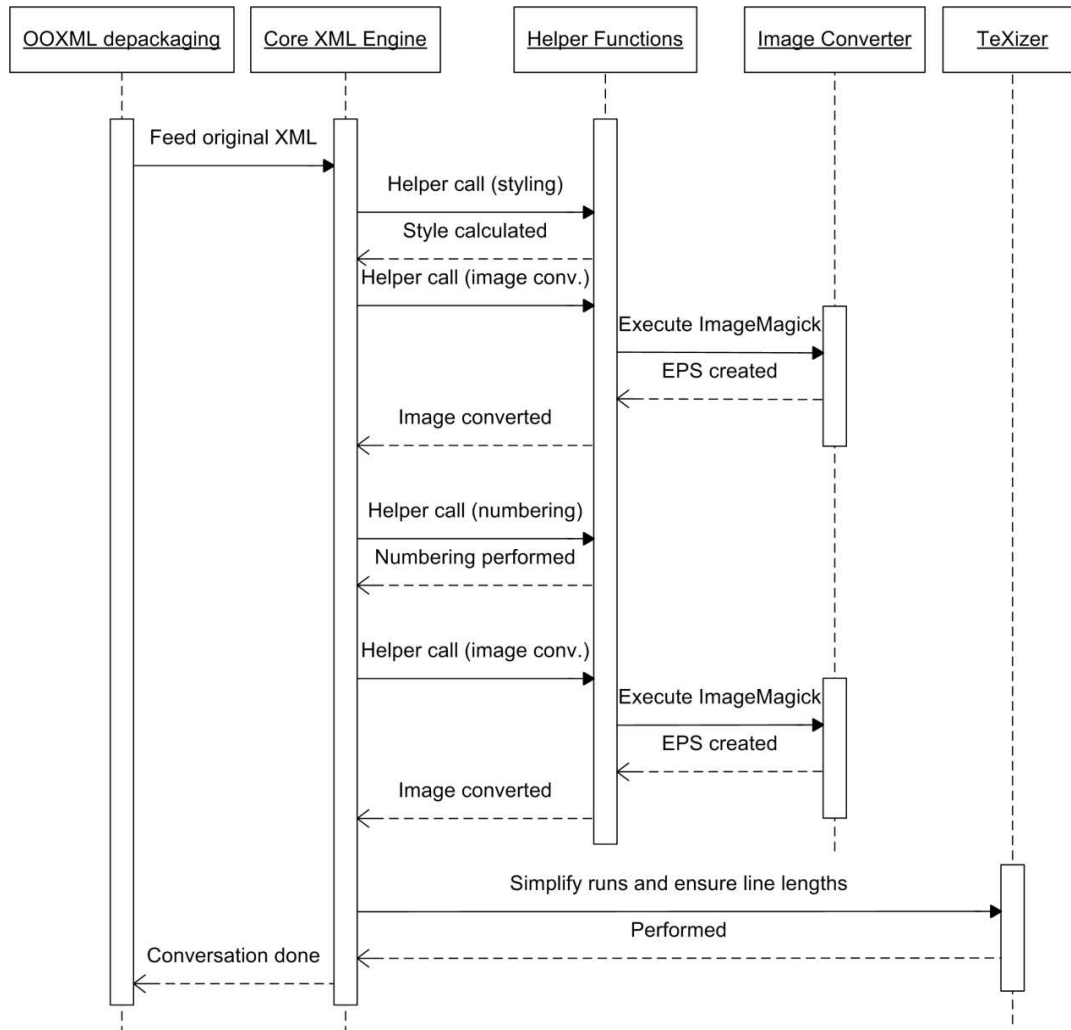
Krisztián Pócza, Mihály Biczó and Zoltán Porkoláb

Figure 1: UML sequence diagram

In the next sections the structure of OOXML will be briefly discussed; first, let us review the idea of *runs*. A *run* is a piece of text which also has some style specification. Runs are placed and removed dynamically while the Word document is edited. A sentence or even a word can be divided into more than one run with the same style. The component called *TeXizer* is responsible to join runs having the same style to a simple run in the outgoing TEX code and break the source line length at some predefined value (default is 72).

The previous description is illustrated by the UML sequence diagram in Figure 1.

4 Features of docx2tex

In this section we list the supported and the unsupported features of docx2tex.

4.1 Supported features

Docx2tex supports the following features of Word 2007 and TEX:

1. Normal text
2. Italic, bold, underlined, stroked, small capitals, . . .
3. Left, right, center aligned text
4. Headings and sections, three levels
5. Verbatim text
6. Style mapping
7. Simple tables
8. Line and page breaks
9. Numbered and bulleted lists
10. Multilevel lists and continuous numbered lists
11. Figure, table and listing captions
12. Cross references to captions and headings

13. Image conversion from various formats (including .png, .jpeg, .emf, etc.) to .eps
14. Substitution of special characters (e.g. \, #, {, }, [,], %, &, ˜, . . .)
15. Text boxes
16. Basic math formulae, Word Equations support

docx2tex supports normal and special text styles and also text alignments. We support heading styles *Heading1*, *Heading2*, and *Heading3* that convert to \section, \subsection, and \subsubsection respectively. Word does not support verbatim text while TEX does. To work around this deficiency, text marked with *Verbatim* style is converted to verbatim text surrounded by \begin{verbatim} and \end{verbatim}. There are many cases where we are required to use different styles for headings or even verbatim.

Only simple, left-aligned tables are supported. Both numbered and bulleted lists are supported, including mixing and nesting. Continuous lists are also supported using the \setcounter, \enumi, and \theenumi commands. Figure, table and listings captions are recognized and we support referencing them; likewise with heading references. Image references are resolved and the images (mainly .png and .jpeg) embedded in the OOXML documents are converted to .eps. The width and height properties are queried and the same properties are used in the resulting TEX documents. Some special TEX characters are also resolved and escaped in the resulting TEX document. Text found in *Text Boxes* of Word documents are also processed and inserted in place of the resulting TEX document.

4.2 Unsupported features

Docx2tex has only basic Word Equations (mathematical formulae) support at the time of publishing this paper. We plan to add more support for Word 2007 Equations and Drawings that can be converted to TEX mathematical formulas and xfigs respectively. Both of them are described in XML format therefore our standard solution can be extended without introducing other technologies.

5 A complex example

In this section we will show a complex example broken into significant parts that introduces the most important features of docx2tex.

5.1 The structure of the OOXML ZIP package

To inspect the content of an OOXML ZIP package we first unzip the contents of our Word 2007 document to a directory and get a recursive directory listing:

```
PS C:\Phd\conf\2008_4_tex\example.docx>
    ls -Recu |% {$_.FullName.SubString(30)}
example.docx\customXml
example.docx\docProps
example.docx\word
example.docx\_rels
example.docx\[Content_Types].xml
example.docx\customXml\_rels
example.docx\customXml\item1.xml
example.docx\customXml\itemProps1.xml
example.docx\docProps\app.xml
example.docx\docProps\core.xml
example.docx\word\media
example.docx\word\theme
example.docx\word\_rels
example.docx\word\document.xml
example.docx\word\fontTable.xml
example.docx\word\numbering.xml
example.docx\word\settings.xml
example.docx\word\styles.xml
example.docx\word\webSettings.xml
example.docx\word\media\image1.jpeg
example.docx\word\theme\theme1.xml
example.docx\word\_rels\document.xml.rels
example.docx\_rels\.rels
```

The most important part is the document.xml file that contains the document itself and references to external items. The numbering.xml file specifies the style of the numbered or bulleted lists contained in document.xml. The styles.xml file specifies information about the styles used in the document. Under the media subdirectory the embedded images can be found (image1.jpeg in our example).

5.1.1 Structure of the document

The text in the document.xml file is grouped into *paragraphs*. Every segment of the document is a paragraph (normal text, heading texts, images, etc.) except for some special elements like tables. Paragraphs are further divided into *runs*. A *run* is a piece of text that also has some style specification.

5.2 Text conversion

The most fundamental feature of tools like docx2tex is the ability to interpret text runs with many basic styling properties and convert them to TEX format. Consider the following example sentence: This is a ^{sentence} ***that*** contains text ***with*** ~~different~~ $formatting$.

This sentence looks like the following in OOXML:

```
<w:p w:rsidR="004F5706" w:rsidRDefault=
        "004F5706" w:rsidP="004F5706">
  <w:r w:rsidRPr="0030655B">
    <w:t xml:space="preserve">
```

```
                    This is a </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:i/>
    <w:vertAlign w:val="superscript"/>
  </w:rPr>
  <w:t>sentence</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:b/>
    <w:i/>
  </w:rPr>
  <w:t xml:space="preserve"> that</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>contains</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> text </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:b/>
    <w:i/>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>with</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:strike/>
  </w:rPr>
  <w:t>different</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:vertAlign w:val="subscript"/>
  </w:rPr>
  <w:t>formatting</w:t>
</w:r>
</w:r>
```

```
  <w:r w:rsidRPr="0030655B">
    <w:t>.</w:t>
  </w:r>
</w:p>
```

The XML node `<w:p>` and `</w:p>` encloses a paragraph while `<w:r>` and `</w:r>` encloses a run. A run contains a range of text (between `<w:t>` and `</w:t>`) and may contain some formatting between `<w:rPr>` and `</w:rPr>` (e.g. `<w:b/>` means bold, while `<w:i/>` means italic font style).

The TeX output generated by docx2tex of the previous sentence looks like the following:

```
This is a \textit{$^{sentence}$}\textbf{%
\textit{that}} \underline{contains} text
\textbf{\textit{\underline{with}}} \sout{%
different} $_{formatting}$.
```

5.3 Headings and verbatim

Headings and verbatim are handled the same way because they can be identified in the source document by examining paragraph level styles.

Consider the following OOXML fragment that describes a first level heading:

```
<w:p w:rsidR="004F5706" w:rsidRPr=
  "0030655B" w:rsidRDefault="004F5706"
                w:rsidP="000136DF">
  <w:pPr>
    <w:pStyle w:val="Heading1"/>
  </w:pPr>
  <w:bookmarkStart w:id="0" w:name=
                "_Ref186547407"/>
  <w:r w:rsidRPr="0030655B">
    <w:t>Heading text</w:t>
  </w:r>
  <w:bookmarkEnd w:id="0"/>
</w:p>
```

The `<w:pStyle w:val="Heading1"/>` node specifies that a first level heading begins, while the contained node `<w:bookmarkStart w:id="0" w:name="_Ref186547407"/>` identifies a unique internal reference (bookmark) to the heading that can be cross-referenced from any part of the document. For each referenceable item Word generates an ugly unique number prefixed with `_Ref` as an identifier (in our example, `_Ref186547407`).

The generated TeX output is the following (line break is editorial):

```
\section{Heading text}\label{section:
_Ref186547407}
```

It is possible to map custom styles to certain TeX elements. The special mappings are loaded from a file with the same name having the extension `.paraStyleName` (that is, a file `example.docx` has

the mapping file `example.paraStyleName`). The Word 2007 styles appearing on the right side of these equations have to be the `w:styleId` attribute of one of the styles found in the `styles.xml` file (names are case sensitive).

Here is a listing to help understand the format of the `.paraStyleName` files:

```
section=Myheading1
subsection=Myheading2
subsubsection=Myheading3
verbatim=Myverbatim
```

5.4   Images and cross references

In OOXML, images are described in a very complex and loose way; there is no space here to show the original XML fragment. Instead we show only the generated TEX code:

```
\begin{figure}[h]
\centering
\includegraphics[width=10.52cm,height=
8.41cm]{media/image1.eps}
\caption{\label{figure:_Ref186544261}:
Figure caption}
\end{figure}
```

The image is centered and the width and the height of the image are preserved. `image1.jpeg` was converted to `image1.eps` and the file was saved in the `media` subdirectory. When the image has a caption then it is also added to the output so that it can be referenced.

Reference to the previous figure is described in OOXML in the following form:

```
<w:p w:rsidR="004F5706" w:rsidRPr=
          "0030655B" w:rsidRDefault=
          "004F5706" w:rsidP="004F5706">
  <w:pPr>
    <w:keepNext/>
  </w:pPr>
  <w:r w:rsidRPr="0030655B">
    <w:t xml:space="preserve">Reference to
                      the figure: </w:t>
  </w:r>
  <w:r w:rsidR="007A289D">
    <w:fldChar w:fldCharType="begin"/>
  </w:r>
  <w:r w:rsidR="006B4DA8">
    <w:instrText xml:space="preserve">
      REF _Ref186544261 \h </w:instrText>
  </w:r>
  <w:r w:rsidR="007A289D">
    <w:fldChar w:fldCharType="separate"/>
  </w:r>
  <w:r w:rsidR="006B4DA8">
```

```
    <w:t xml:space="preserve">Figure</w:t>
  </w:r>
  <w:r w:rsidR="006B4DA8">
    <w:rPr>
      <w:noProof/>
    </w:rPr>
    <w:t>1</w:t>
  </w:r>
  <w:r w:rsidR="007A289D">
    <w:fldChar w:fldCharType="end"/>
  </w:r>
</w:p>
```

The generated TEX code is simple (it can be seen that the *Figure 1* text has been omitted from the output because it is internal to Word):

```
Reference to the figure: \ref{figure:
_Ref186544261}.
```

Referencing tables is the same as for figures and sections.

5.5   Lists and tables

There are two main categories of lists supported by OOXML and Word 2007: numbered and bulleted. Both types of lists are allowed to have multiple levels, and numbered lists can be continuous, meaning that the list can be interrupted by some other content and then continued at the same number.

The first item of a numbered list looks like the following:

```
<w:p w:rsidR="004F5706" w:rsidRPr=
    "0030655B" w:rsidRDefault="004F5706"
                  w:rsidP="004F5706">
  <w:pPr>
    <w:pStyle w:val="ListParagraph"/>
    <w:keepNext/>
    <w:numPr>
      <w:ilvl w:val="0"/>
      <w:numId w:val="1"/>
    </w:numPr>
  </w:pPr>
  <w:r w:rsidRPr="0030655B">
    <w:t>First</w:t>
  </w:r>
</w:p>
```

The nodes enclosed in `<w:numPr>` and `</w:numPr>` specify that we have a list. The `w:val` attributes of `w:numId` and `w:ilvl` specify numbering identifier and level parameters (style 1 at level 0 in our example). It may seem strange that the *w:numPr* nodes describe both numbered and bulleted lists. It is the numbering identifier and the level parameter that distinguishes between the two categories of lists. These parameters are defined in the `numbering.xml`

file that is processed by docx2tex. The above element is part of a complex multilevel numbered list:

```
\newcounter{numberedCntA}
\begin{enumerate}
\item First
\item Second
\item Third
\begin{enumerate}
\item First
\item Second
\item Third
\end{enumerate}
\item Fourth
\setcounter{numberedCntA}{\theenumi}
\end{enumerate}
```

When a previous list is continued, `\setcounter{enumi}{\thenumberedCntA}` is inserted after the `\begin{enumerate}` by docx2tex.

We omit showing a bulleted example since it differs only in the TEX output: the *itemize* keyword is used instead of *enumerate*, and we do not have to maintain counters for continuous lists.

Docx2tex supports only simple tables, so no merged, divided or differently aligned table cells are possible, but the current features still allow docx2tex to be able to convert most tables.

Again there is not enough space here to show the OOXML version of a simple table. The generated TEX output of a table with four cells is the following:

```
\begin{tabular}{|l|l|}
\hline
1 & 2 \\
\hline
3 & 4 \\
\hline
\end{tabular}
\caption{\label{table:_Ref186545972}:
caption}
\end{table}
```

5.6 Special characters

TEX uses some special characters to place formatting commands to structure or change the appearance of text. When we want to place these special characters in running text they have to be described in a special way.

Consider the following set of special characters:

```
< > as'q ... # \ { } % ~ _ ^ & $ ""
```

These are described in OOXML in the following form:

```
<w:p w:rsidR="00AB630B" w:rsidRDefault=
        "00AB630B" w:rsidP="00AB630B">
  <w:r>
    <w:t xml:space="preserve">&lt;&gt;
```

```
                                    </w:t>
  </w:r>
  <w:proofErr w:type="spellStart"/>
  <w:r>
    <w:t>as'q</w:t>
  </w:r>
  <w:proofErr w:type="spellEnd"/>
  <w:r>
    <w:t xml:space="preserve"> ...# \ { }
                                </w:t>
  </w:r>
  <w:proofErr w:type="gramStart"/>
  <w:r>
    <w:t>% ~</w:t>
  </w:r>
  <w:proofErr w:type="gramEnd"/>
  <w:r>
    <w:t xml:space="preserve"> _ ^ &amp;
                         $ ""</w:t>
  </w:r>
</w:p>
```

The resulting TEX code is:

```
$<$$>$ as'q ...\# $\backslash$ \{ \} \% \~
\_ \^ \& \$ "\,"
```

5.7 Math formulae

Math formulae (Word Equations) are described in a hierarchical XML structure inside the OOXML document. It can be easily walked by a recursive algorithm to create TEX output. The name of the XML node that hosts mathematical formulae is `m:oMath`. While standard parts of the document are described by nodes that have the `w` XML namespace, Word Equations have the `m` namespace.

There is no space to show an OOXML fragment that holds a quite complex formula, but here is the formula and the resulting TEX code that was generated by docx2tex:

$$B(v) \cong \frac{1}{\sqrt{2\pi}} \int_v^\infty e^{-\frac{t^2}{2}} dt$$

```
$B(v)\cong \frac{1}{\sqrt{2\pi }}\int_{v}^
{\infty }{e^{-\frac{t^{2}}{2}}dt}$
```

6 A use case

Let us suppose the following scenario: Two authors decide to write a scientific article about their research topic and submit it to a conference or journal. First they split the proposed article into sections and assign each section to one of the authors. They start to work independently using Word 2007. After both of the authors finish they merge the resulting text into a single document. After that step the first author reads the whole document and makes changes using the Track Changes function of Word.
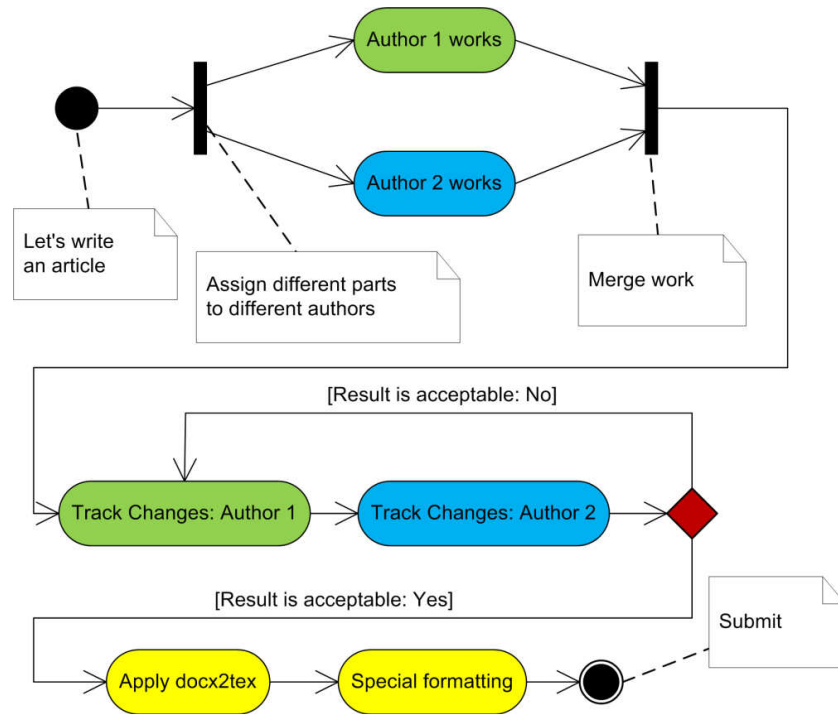
Figure 2: Workflow

The second author accepts or rejects the changes of the first author and also makes his own changes using the Track Changes function of Word. When the authors agree that the quality of the article is acceptable, they convert it to TeX using docx2tex and apply special formatting required for the particular conference or journal. Now the article can be submitted.

This workflow is illustrated in Figure 2.

As can be seen we exploited the strengths of both the WYSIWYG Word 2007 system to support effective team work and the typesetting TeX system to produce the best quality printout. The conversion between the file formats they use was performed using docx2tex. Note that currently docx2tex is able to do rough conversion and cannot apply special commands and styles.

Readers not familiar with the Track Changes function should consider Figure 3.

7 Conclusion and further work

In this article we introduced a tool called docx2tex that is dedicated to producing TeX documents from Word 2007 OOXML documents. The main advantage of this solution over classical methods is that we process the bare XML content of OOXML packages instead of processing binary files or exploiting the capabilities of the COM API of Word, thus mak-

ing our solution more robust and usable.

We presented the main features of docx2tex, related primarily to text processing and formatting, structuring the document, handling images, tables and references. There are some important features that we consider worth implementing in the future:

1. More Equations support
2. Embedded vector graphical drawings
3. Configuration settings
4. Optional font and coloring support
5. Documentation
6. Automated installer

Multicolumn document handling and templating may also worth considering.

We published the application as free and open source software under the terms of the BSD [13] license, so anybody can use it royalty free and can add new features to the current feature set.

The source code and the binary of the application can be downloaded from `http://codeplex.com/docx2tex/`. If the reader would like to participate in the development of docx2tex, please contact the authors.

References

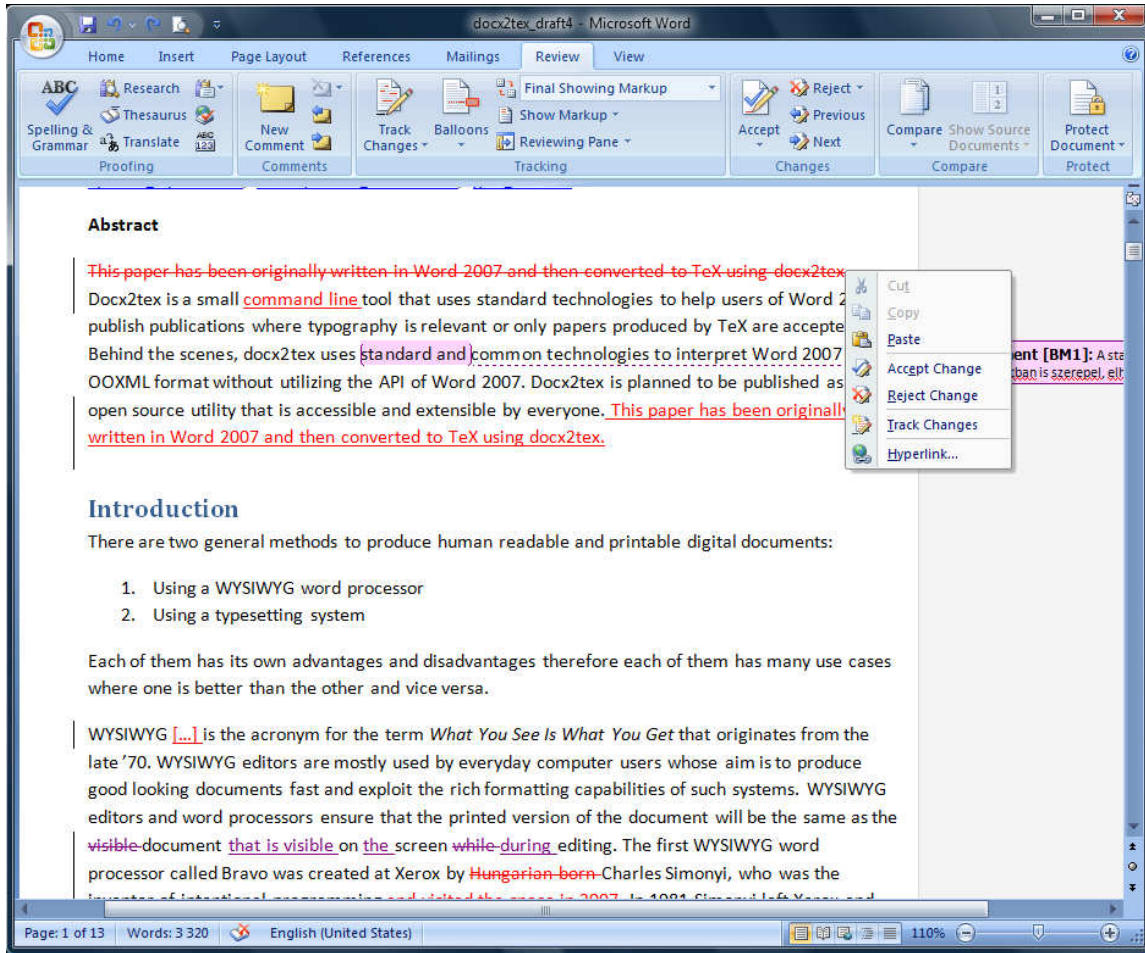[1] Wikipedia article on What You See Is What You Get. `http://en.wikipedia.org/wiki/WYSIWYG`

Figure 3: Track Changes function of Word

[2] Wikipedia article on Microsoft Word. `http://en.wikipedia.org/wiki/Microsoft_Office_Word`

[3] Product page of Microsoft Word. `http://office.microsoft.com/en-us/word/FX100487981033.aspx`

[4] Wikipedia article on TeX. `http://en.wikipedia.org/wiki/TeX`

[5] Product page of *Word2Tex*. `http://www.chikrii.com/`

[6] Product page of *Word-to-LaTeX*. `http://kebrt.webz.cz/programs/word-to-latex/`

[7] Product page of OpenOffice.org. `http://www.openoffice.org/`

[8] Project page of *rtf2latex2e*. `http://sourceforge.net/projects/rtf2latex2e/`

[9] ECMA 376 OOXML Standard. `http://www.ecma-international.org/publications/standards/Ecma-376.htm`

[10] ECMA 335 .NET CLI Standard. `http://www.ecma-international.org/publications/standards/Ecma-335.htm`

[11] Link to ISO/IEC 23271:2006 Standard. `http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html`

[12] Project page of ImageMagick. `http://www.imagemagick.org/`

[13] Article on BSD license. `http://en.wikipedia.org/wiki/BSD_license`