Improving Version-Aware Word Documents

by

Alexandre Azevedo Filho

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin–Milwaukee

December 2017

Abstract

Improving Version-Aware Word Documents

by

Alexandre Azevedo Filho

The University of Wisconsin–Milwaukee, 2017
Under the Supervision of Professor Ethan V. Munson

Coakley *et al.* described how they developed Version Aware Word Documents, which is an enhanced document representation that includes a detailed version history that is self-contained and portable. However, they were not able to adopt the unique-ID-based techniques that have been shown to support efficient merging and differencing algorithms. This thesis describes how it is possible to adapt existing features of MS Word's OOXML representation to provide a system of unique element IDs suitable for those algorithms. This requires taking over Word's Revision Save ID (RSID) system and also defining procedures for specifying ID values for elements that do not support the RSID mechanism. In addition, we provide an updated version of the MS plug-in developed by Coakley *et al.*. Important limitations remain but appear surmountable.

Table of Contents

List of Figures

Acknowledgements

I would like to do a special acknowledgment to my advisor, Dr. Ethan V Munson. He is the main reason for everything I have accomplished. Giving me all the necessary support, not only as an advisor, but as a friend. Thank you for all the availability, assisting, advising and correcting.

Thank to all the other professors who contributed to my development as a student, as a professional and as a person. Especially, to my committe members John Boyland and Ichiro Suzuki.

Thanks to Cheng Thao for all the support and previous work shared.

Thanks to my family for giving all necessary support: my parents, Alexandre Azevedo and Ana Paula Azevedo, my brothers Henrique Azevedo, Arthur Azevedo and my sister Luana Azevedo.

I would like to thank my fiance and future wife Leticia for giving me extra support and encouragement.

I would also like to thank my friend and coworker Ahmed Shatnawi who was always there to support, discuss and collaborate.

Chapter 1

Introduction

In collaborative writing, multiple people work together on the same document. It is a common approach in many different situations. For example, when students are required to work in group or when coworkers needed to work in the same document simultaneously. It is not difficult for each person in a group to get the original document and do their part of the work in the document. Most of the challenges in collaborative writing comes when the group needs to put together their individual work to make a final result. Especially when dealing with large and complex documents, manual merging can be a very difficult task. One solution for those challenges is the use of a version control system, which provides a set of methods that allow groups to achieve an organized and productive collaboration. Two of its main functions are to allow users to maintain a history of changes to the document and to merge changes made in parallel by different authors.

Sophisticated Version Control Systems(VCSs) like Git [2], TFS [3] and Subversion [4] are popular among technical users like software developers for keeping track of their software development process. However, they are often ignored by non-technical users who find them difficult to work with. A common option adopted by non-technical users is to use online software like Google Docs [5], Microsoft Office [6] and Dropbox [7]. Those collaborative applications allow users to share documents, simultaneously edit and keep a version history of changes. For those applications, the user needs to be connected and use the online text editor in order to make changes to the document. Thus, the problem with both VCS and online collaborative writing application is that they usually require access to a central repository to store and manipulate version data.

Without the use of VCS and/or access to the online collaborative applications, a common approach among users is to track different revisions of a document by saving them with different names that indicate the document evolution. This approach is not

only common because of the lack of knowledge of the VCS and tools mentioned above. In most of the cases, it is chosen because it is easy and fast. However, in a collaborative situation, collaboration can be archive only by manual branching and merging or by using a merging tool that would help in the process of reintegrating the different document into a unique document. Both tasks can be confusing and time consuming. Thus, developing a sophisticated offline VCS for documents could be a useful service.

Microsoft Word(MS Word) is one of the most widely used word processors. For this reason, we have decided to target it. Being able to have a offline VCS for Microsoft Word that has an easy interface and powerful functionality would help users to easily keep track of their changes and also merge documents within the Microsoft Word software. Microsoft does provide a number of services related to versioning, but their capability is limited to the form of current version/past-version or comparison of two documents:

- The "Track Changes" feature tracks changes made on top of working document. This functionality can be used throughout the development of the whole document and it will look like having the history of the changes with the author inside the document. However, once a change in the track system is accepted or rejected, there is no way to know the changes that were there before. So, the user has lost the "history of changes".

- Compare and combine features are also present in MS Word, but it only works with a maximum of two documents at time (Original and Revised). The features assumed that anything in the original document that is not also in the revised file would be marked as added. Anything from the original document that is not in the revised document is marked as deleted. They are an useful tool for lightly formatted documents. However, when comparing or combining heavily formatted documents with multiple columns, MS Word would sometimes have trouble matching the text, marking large blocks of unchanged text as added or deleted. Furthermore, if the documents the user is trying to compare or combine are using "Track Change", MS Word will assume that the changes in the track system is already accepted.

This thesis is a continuation of a series of projects investigating improved version control for offline documents, such as these produced by Coakley et al [8] who implemented a plug-in to support the version-aware approach proposed by Thao and Munson [9] into MS Word. Previous research by Thao and Munson showed that high quality merging is possible when unique element IDs are stored inside XML documents [1]. Since Microsoft Word last format (.docx) is just a ZIP file of XML documents and in order to support their approach, we are introducing a novel way to preserve IDs in Word documents. This show allow future developers to provide a sophisticated versioning solution for MS Word documents that relies only on standard Office Open XML(OOXML) attributes and elements. OOXML is a XML-based format for office documents developed by Microsoft. It includes word processing documents, spreadsheets, presentations, as well as charts, diagrams, shapes, and other graphical material [10]

The remainder of this paper is organized as follows. In Chapter 2, we describe the background work done prior to this work. Chapter 3 describes the Word format, how it started and its structure. Chapter 4 presents our solution, including other approaches we tried prior to adopting the proposed solution. The final chapter describes our conclusion and future work.

Chapter 2

Background

This thesis is a continuation of a line of research on versioning of office documents. Thao and Munson [1] initially presented an efficient three-way merging algorithm for XML documents that relies on the document elements having unique IDs. A three-way merge algorithm is performed when two versions of a document *version 1*(V1) and *version 2*(V2) that were derived from the same origin, or parent, *version 0*(V0) are incorporate into a new common *version 3*(V3) as shown on Figure 2.1. Their approach assigns unique identifiers (UIDs) to all new elements in the XML document. By using node identity to keep track of what parts of the XML document have changed, they were able to present a faster algorithm that uses less memory and is more precise than other algorithms.

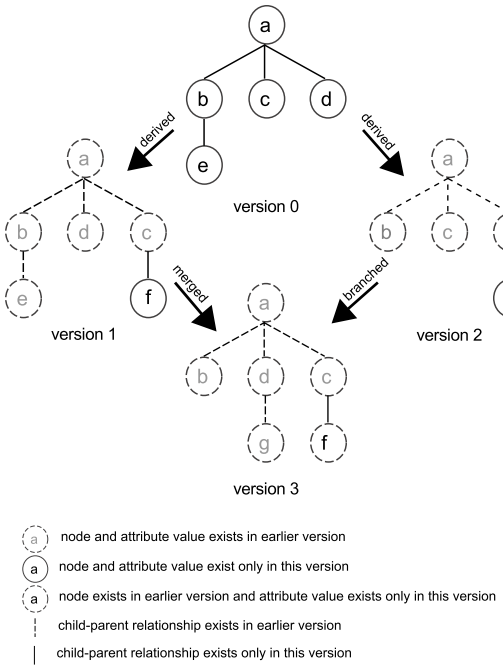

Figure 2.1: Three-way merged of versioned tree. Figure from Thao and Munson [1]

Earlier, Lindholm developed a three-way algorithm for XML documents called *3dm* based on the matching technique [11]. The technique relies on the fact that a matching between nodes in original version V0 and modified version V1 can indicate what type

of edit operations had been performed. For example, if a node is in V0 with no match in V1, it is because the n was deleted. Similarly, if a node is in V1 with no match in V0, it is because the node was added. In Lindholm's approach, each XML document is encoded as a set of content and parent-child-successor(PCS) relations. Then, one-to-one node matching is performed between V1 and V0, and also between V2 and V0 to create a set of changes where merge rules will be applied to generate the incorporated document.

The *molhado* XML merge algorithm developed by Thao and Munson was compared with two three-way merge algorithms: *3dm* and *deltaxml*. The *deltaxml* is a commercial XML merge tool. In addition, they also compared it to *xcc*, which is a two-way patching algorithm. Among other test scenarios, the experiments evaluated the total execution time as changes and elements increase and also memory usage as changes and elements increase.

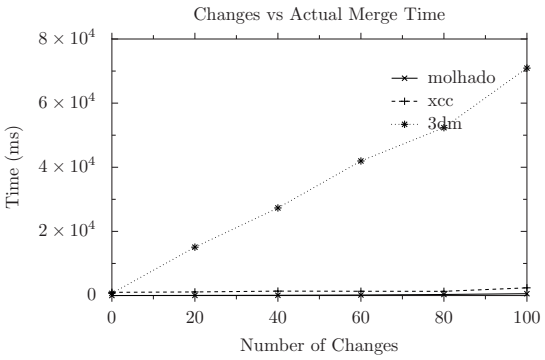

Figure 2.2: Merge time as changes increase.

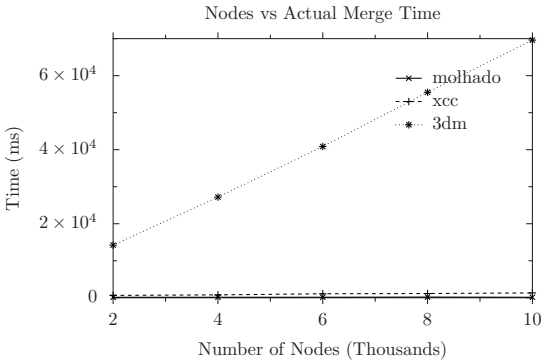

Figure 2.3: Merge time as elements increase.

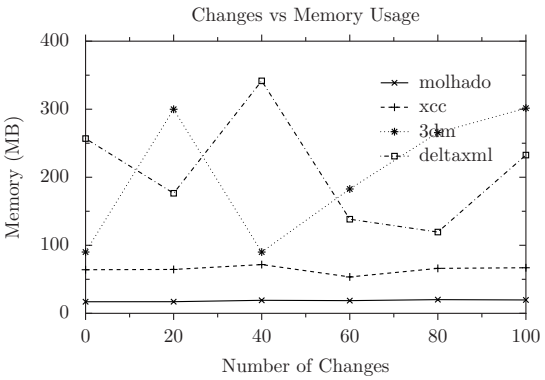

Figure 2.4: Memory usage as changes increase.

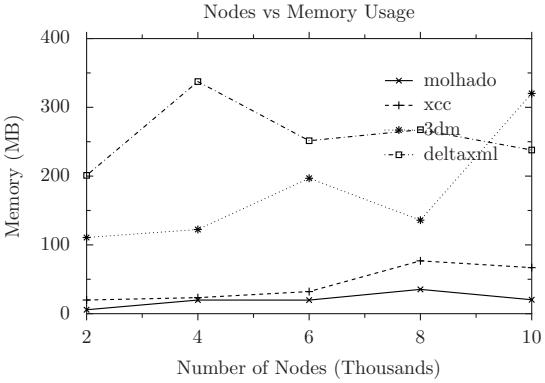

Figure 2.5: Memory usage as elements increase.

The main results from, the Thao and Munson paper are shown in Figures 2.2, 2.3, 2.4 and 2.5 (all taken from the original paper) and described below:

- In all three systems, execution time was linear in relation to changes and number of document nodes.
- The *3dm* execution time increases linearly and fast with an increase in changes and also when increasing elements while *molhado* and *xcc* have the lowest total execution time.
- When comparing memory usage, *3dm* and *deltaxml* presented high variability as the number of changes or elements increase.
- *molhado* is the tool that uses the least amount of memory among all the tools compared.

The whole experiment was able to show that *3dm's* performance is drastically affected with permutations of attributes and also the change list created by the matching technique will be inconsistent when there are multiple moves within the same parent. The reason behind the efficiency showed in the experiments is because the algorithm uses a versioned data structured tree data where only one full document tree needs to be stored in memory, along with tree deltas to represent any changes. When a merge needs to be made, only the nodes that have changed will be manipulated and all the others will be ignored.

Later, Thao and Munson described the Version-Aware XML Document format [9], which stores the full version history of the document in a preamble of the document file and also relies on the unique ID (UID) approach for version differencing and merging. The original XML document must be labelled with the UIDs prior to the distribution of the document. Thus, the molhado XML algorithm would be able to perform differencing and merging of the nodes in the document. In the proposed approach, they used reverse deltas in order to encode the version history of the document. Reverse delta is a technique where only the latest version is stored in full form, while all other versions are stored as deltas that contains the changes between each version of the file. Version Aware documents can

be stored offline, outside a repository, while maintaining the kind of versioning history and supporting the operations that are typical of VCSs.

It is reasonable to ask whether one can expect XML-based applications to maintain unique IDs on their document elements. Pandey and Munson [12] addressed this question by showing that the LibreOffice Writer application could be modified to support unique IDs with only small changes to the application source code. Coakley *et al.* [8] tried to bring the version-aware approach to MS Word documents by implementing a version-aware plug-in. However, they were unable find a way to use the unique ID approach and instead adopted matching techniques similar to those of Lindholm's *3dm* system. Furthermore, they had to store their versioning information in separate XML files within the larger zip file used by MS Word. The problem with matching techniques is that they are slower and less correct than the unique ID approach.

MS Word itself supports version control through two main features. "Track Changes" keeps a detailed history of changes made to the document, even if made by multiple users. A user trying to create a final version of the document can choose to accept or reject each of the changes. The "Compare" feature is used when authors have been making changes in parallel to the same document. It allows the differences in the two parallel versions to be compared and also supports merging of the two versions. A key limitation of MS Word is that it does not maintain a full version history graph. Instead, it only has the notion of an original accepted version and of a current version produced by the users' changes.

In this research, we extend the work of Coakley *et al.* to provide better Version Aware Word Documents. We propose a structure where we add unique IDs to Microsoft Word XML documents relying only on XML elements and attributes provided by the Microsoft Word architecture and its Office Open XML file format (OOXML) [13]. This allows our system to use the XML differencing and merging algorithms of Thao and Munson [1], which have better performance than Coakley's matching techniques. Also, our version history information can be embedded directly in the document content file (`document.xml`), which has the potential to simplify document security measures.

Chapter 3

Microsoft Word File Structure

Microsoft Office(MS) 2007 introduced new file formats based on the Office Open XML (OOXML) [10] for all of its authoring applications: Word, Excel and PowerPoint. All new OOXML-based formats became the default when saving new documents, leaving as a second option the binary file format used in Microsoft Office 97 through Microsoft Office 2003.

The new OOXML-based formats brought many advantages to MS Office. The new file format is much smaller than the earlier format, because it takes advantage of ZIP technology. The ZIP file contains a collection of files and folders that is described below. ZIP compression reduces overall file size. Plus, there is a improvement in recovering damaged files, since different data components are separated in different components inside the ZIP. Thus, it is possible to open files even if a component (table or image) is corrupted. In addition, the ability to work with a document file as XML allows custom APIs to interact with a MS Word document, for example, without the need to open the MS Office Word editor. "Custom XML" is supported and it means that information can be created/edited in XML, and then added to the ZIP file structure [14].

Although MS Office 2007 was the first version to introduce new standard formats based on OOXML, the use of XML was introduced much earlier in older versions of MS Office. Microsoft started using XML in Microsoft Office XP in 1999 with SpreadsheetML, which was also present in Office Excel 2002. SpreadsheetML is a XML format for storing Excel spreadsheets. However, it cannot describe all the parts of the spreadsheet [14].

Microsoft then introduced the WordprocessingML for the Word 2003. WordprocessingML was a significant improvement because it could describe every part of the document. Nevertheless, it still had some limitations. One of the problems was that binary data (such as images) would have to be encoded as text within the XML file itself, which

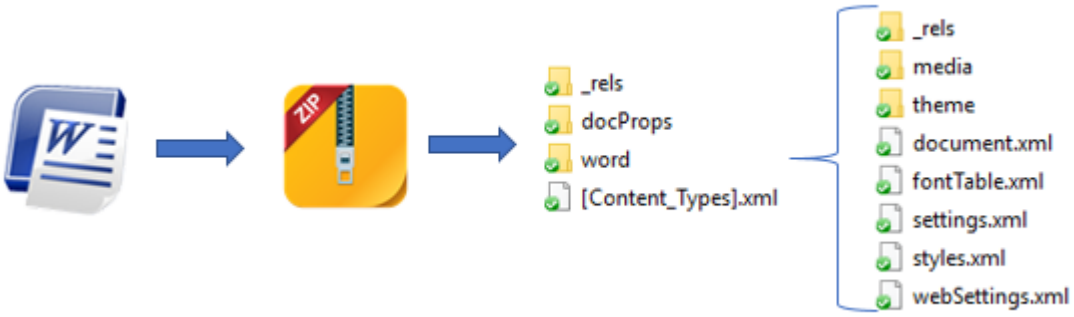

Figure 3.1: Microsoft Word document ZIP structure

increases file size when working with a document containing many images. Furthermore, Word 2003 embeds all custom XML data directly into the WordprocessingML that describes the document. This can make the custom XML difficult to access and manipulate by external processes. The problems in WordprocessingML were solved with the release of the new file format in Word 2007 by dividing the file into document parts with each piece representing a part of the content of the file [14].

Figure 3.1 shows the overall structure of an MS Word file's ZIP structure. The most important file inside the compressed ZIP seen in Figure 3.1, is *word/document.xml* which specifies the document's textual content. Non-text content is generally stored in other files or subfolders and linked to *document.xml* by an indirect reference scheme. The files inside both *_rels* folders, for example, store relationship items that specify how collection of document parts come together to form the document. Below we have a list of all the default files and their functions in the document:

- The *_rels* folder contains a *.rels* XML file. This file contains references that indicate MS Word where to find document contents. Examples of files referenced are *word/document.xml* and *docProps* folder files.

- The *docProps* folder contains two XML files: *core.xml* and *app.xml*. The *core.xml* file has information related to document revisions like who created, who last modified and number of revisions. It is important to mentioned that it is not a complete version history track. It only contains creator and last editor. The *app.xml* has information belonging to the document like number of words, pages, characters etc.

- The *Content_ Types* folder, as the name suggests, contains information about the types of media inside the document.
- The *word/_ rels* folder contains a XML file, named *document.xml.rels*. It is a relationship file and creates links between references in the *word/document.xml* and resources that are not stored inside *word/document.xml*, such as images.
- *word/media* stores all the media files that are used in the document. The media files are not stored with the same file format they had when they were added to the document. Images, for example, are stored as .emf, an file format used when printing by the Windows operating system.
- *word/theme* contains information about the document's theme. For instance, color scheme, font, and format schemes.
- *word/document.xml* is the main XML with the document's textual content. It contains many elements. Paragraghs, Tables and Sections are the main block-elements inside the *document.xml*.
- *word/fontTable.xml* has information about fonts used in the document.
- *word/settings.xml* specifies the settings for the document, including whether or not to hide spelling and grammatical errors, track revisions and write protection.
- *word/styles.xml* contains the definitions for the set of styles used by the document.
- *word/webSettings.xml* contains information for web-specific settings used by the document. For example, it has settings that affects how the document is handled when saved as HTML.

Chapter 4

Our Solution

The main goal of this research was to find a solution to the problem faced by Coakley *et al.*.They were not able to store IDs inside the MS Word text content (document.xml). Therefore, they had to use an merge technique based on matching instead of the merge algorithm developed by Thao and Munson that would provide a much faster and reliable approach for the merge in the plug-in developed.

The fact that Microsoft Office is not open source software presented some obstacles. Due to the lack of detailed documentation about the software, a good part of the research was done using unofficial documentation. Furthermore, we had to use primarily a trial-and-error approach in order to test different possible solution scenarios. In some cases, changing a single element, attribute or property would make MS Word report the file as corrupted, or simply ignore our additions and/or changes and delete them without a trace.

Our first attempt was to use the *ignorable namespaces*. We observe that in the OOXML document, namespaces can be declared as ignorable, allowing identification of some markup as not essential to the document content. Thus, those namespaces would be inside the XML document but ignored by MS Word editor. After testing, we were able to add *ignorable namespaces* to the document without causing file corruption as shown in Figure 4.1. However, after editing the document and saving it using the MS Word editor, all the *ignorable namespaces* were silently removed. Later, we found that custom XML markup using *ignorable namespaces* used to be a feature of OOXML, and while the markup is still part of the standard, custom XML markup in the document part is no longer supported.

The second approach we tried was the use of invisible bookmark elements. A bookmark is a region of text that has a unique name and id as attributes and acts as a target for

```
<w:document
    ...
    xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
    mc:Ignorable="w14 w15 wp14">
    <w:body>
        <w14:vids w14:id="14242V1F"></w14:vids>
        <w:p w:rsidR="00CD7EC6">
            <w:r>
                <w:t>Text</w:t>
            </w:r>
        </w:p>
        <w:p w:rsidR="009418CB" />
        +<w:sectPr w:rsidR="009418CB">
    </w:body>
</w:document>
```

Figure 4.1: Example of addition of ignorable namespaces

a link. Bookmarks in OOXML have a starting element *w:bookmarkStart* and the ending is defined by an empty element *bookmarkEnd*. In order to define an invisible bookmark, the name attribute needs to start with an underscore, for instance, <w:bookmarkStart w:id="0" w:name="_Name"/>. The bookmarks worked well in the initial tests we made. We were able to wrap other elements with invisible bookmarks and we used the name attribute to store the id and identify the wrapped element as shown in Figure 4.2. Unfortunately, the bookmarks are not totally invisible for the user. The user would be able to change or delete a defined bookmark, even if by mistake. Another problem we would possibly have had is that we would need to wrap every element tag. For large documents, that would increase the document size substantially. While we were testing the bookmarks, we discovered the RSID attributes and decided that RSID would be a better approach.

RSID (Revision Save ID) attributes are important for our application. RSIDs attributes were introduced in 2003 to allow applications to merge two versions of the same document that have been edited in parallel [15]. Each time a document is opened, edited, and saved, a random RSID value is generated for the editing session and certain changed elements (mostly the block elements, such as paragraphs) are marked with this session RSID. The ID values are 32-bit unsigned hexadecimal integers. Thus, RSIDs allow you to see what was done in a single session and most documents have multiple elements with the same RSID value. There are multiple RSID attributes and, as far as we can tell, their differences are not well-documented. Figure 4.3's paragraph element shows three

```
<w:document
    ...
    xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
    mc:Ignorable="w14 w15 wp14">
    <w:body>
        <w:bookmarkStart w:id="0" w:name="_FF4124FF"/>
            <w:p w:rsidR="00CD7EC6">
                <w:r>
                    <w:t>Text</w:t>
                </w:r>
            </w:p>
        <w:bookmarkEnd w:id="0"/>
        <w:bookmarkStart w:id="1" w:name="_0041223F"/>
            <w:p w:rsidR="009418CB"></w:p>
        <w:bookmarkEnd w:id="1"/>
        +<w:sectPr w:rsidR="009418CB">
    </w:body>
</w:document>
```

Figure 4.2: Example of the use of invisible bookmarks. The bookmarks are wrapping only paragraph elements.

such attributes: w:rsidR, w:rsidRDefault, and w:rsidP. We use the w:rsidR attribute, because our study of MS Word's behavior leads us to believe that this is the primary RSID attribute.

In the following subsections, we will describe how we use the RSIDs in our solution for each main block-element present in the MS Word XML structure.

4.1 Paragraphs

The paragraph is the main block-level content element in an OOXML word processing document. In OOXML terms, *block-level* means a unit of content that starts on a new line. The paragraph element is used to represent many document elements that other document formats (e.g. HTML) represent with distinct elements, such as lists, list items, quotations, etc. Each paragraph element <w:p> can contain a single *paragraph properties* element <w:pPr> and one or more *run* <w:r> elements. A *run* element generally defines a non-block area of text that shares a common set of formatting properties, though it can also contain non-text elements like images, drawings and special characters. MS Word uses *run* elements to represent many types of inline formatting features for which other formats have distinct elements. Each *run* element contains a single *run property* element <w:rPr> and may contain a single content element, which is most commonly

text <w:t> [10]. The OOXML code for a simple paragraph is shown in Figure 4.3.

```
<w:p w:rsidR="0011569B" w:rsidRDefault="0011569B" w:rsidP="00E200CD">
    <w:pPr>
        <w:pStyle w:val="TableContents"/>
        <w:snapToGrid w:val="0"/>
        +<w:rPr>
    </w:pPr>
    <w:r>
        +<w:rPr>
        <w:t>Brazil</w:t>
    </w:r>
</w:p>
```

Figure 4.3: XML listing for a paragraph structure

The Version Aware Document approach works best if each versioned element has a unique ID within the document. In fact, the differencing and merging algorithms do not pay attention to element type and use only the IDs. So, we take two steps to support those algorithms. First, we "hijack" the RSID system in order to record partial UIDs on certain elements. Second, we develop a process for defining *versioning IDs* (VIDs) for all elements, based on the explicitly recorded RSIDs. It is the VIDs that are used in the merging algorithm, while only the RSIDs appear in the document.xml file.

We modify the document.xml file by giving a unique rsidR value to each paragraph <w:p> element. So, even if a set of edits were made in the same session, each edited paragraph will have a distinct RSID. (Obviously, this will defeat the MS Word "combine documents" feature.) Furthermore, we use another RSID attribute, "w:rsidRPr" to give each *run* <w:r> element a unique RSID value. This allows us to distinguish the multiple runs in a paragraph from each other. Figure 4.4 shows the XML for the content shown in Figure 4.3 after we modify it so that the paragraphs have distinct w:rsidR values and the *run* has gained a w:rsidRPr value.

A VID is a unique ID for all elements within a paragraph, including elements like *paragraph properties* <w:pPr> and text <w:t>. The value of a VID is a 64-bit unsigned hexadecimal integer. For elements that have an explicit RSID attribute value (either w:p or w:r), the VID is the RSID value concatenated with #00000000. All other elements in a paragraph will be children of an element with an RSID value. The VID for these elements will be the RSID value of the parent concatenated with the cryptographic hash

```
<w:p w:rsidR="7E80B124" w:rsidP="00E200CD" w:rsidRDefault="0011569B">
    <w:pPr>
      <w:pStyle w:val="TableContents"/>
      <w:snapToGrid w:val="0"/>
       +<w:rPr>
      </w:pPr>
      <w:r w:rsidRPr="6CA63AA1">
      +<w:rPr>
          <w:t>Brazil</w:t>
      </w:r>
</w:p>
```

Figure 4.4: XML listing for a paragraph in our application

of the element's name (as a 32-bit unsigned hexadecimal integer). This rather odd tactic works because all elements that appear in the paragraph structure, other than the run <w:r> element, can only appear a single time within their parents.

When running our application on the XML document from Figure 4.3, the only changes in the XML document would be the value changed in the w:rsidR element in the paragraph to make it unique from the others and the addition of a w:rsidRPr element in the *run* element as can be seen in Figure 4.4.

4.2 Tables

Tables are also a widely used feature of word processing documents and they presented challenges for defining IDs that required us to be creative. The main components of a Table, shown in Figure 4.5, are the elements: table <w:tbl>, table row <w:tr>, table cell <w:tc> and their respective properties plus the table grid <w:tblGrid>. In general, a table contains one or more rows, each of which contains one or more cells. In general, cells contain only a series of (possibly empty) paragraphs, but while it is rare, the table structure can be recursive with a cell element holding another table in addition to one or more paragraphs.

The way we assign IDs in the table XML structure is the following:

- To identify the Table element <w:tbl>, we place an ID in the value attribute of the table caption element <w:tblCaption>, which is one of the table's property elements. This is necessary because is not possible to add an RSID attribute to

```
<w:tbl>
    <w:tblPr>
        <w:tblW w:w="0" w:type="auto"/>
        <w:tblInd w:w="-8" w:type="dxa"/>
        <w:tblLayout w:type="fixed"/>
       +<w:tblCellMar>
        <w:tblLook w:noVBand="0" w:noHBand="0" w:lastColumn="0" w:firstColumn="0" w:lastRow="0" w:firstRow
            ="0" w:val="0000"/>
    </w:tblPr>
    <w:tblGrid>
        <w:gridCol w:w="3600"/>
        <w:gridCol w:w="3634"/>
    </w:tblGrid>
    <w:tr w:rsidTr="00E200CD" w:rsidR="0011569B">
        <w:tc>
            <w:tcPr>
                <w:tcW w:w="3600" w:type="dxa"/>
               +<w:tcBorders>
                <w:shd w:val="clear" w:color="auto" w:fill="auto"/>
            </w:tcPr>
           +<w:p w:rsidR="0011569B" w:rsidP="00E200CD" w:rsidRDefault="0011569B">
        </w:tc>
        <w:tc>
            +<w:tcPr>
            +<w:p w:rsidR="0011569B" w:rsidP="00E200CD" w:rsidRDefault="0011569B">
        </w:tc>
    </w:tr>
    +<w:tr w:rsidTr="00E200CD" w:rsidR="0011569B">
    +<w:tr w:rsidTr="00E200CD" w:rsidR="0011569B">
</w:tbl>
```

Figure 4.5: XML Listing for table. This table has 3 rows, but the listing has only expanded the first

the table tag. Adding a table caption element does not cause any visible changes because the caption is only accessible via a table properties dialogue. (Of course, a user could alter the ID we create through the table properties dialog and undermine our technique.) The ID in the table caption is expanded into a VID by concatenating it with zero (#00000000).

- All the remaining table property tags appear only once within the table. So, as we did for the paragraph properties elements, each one receives a VID by concatentation of the ID in the table caption and the hash of the property's element name.

- Table grid <w:tblGrid> is one of the few places that an individual tag, for example <w:gridCol>, can appear more than one time. To solve this issue we keep a list of the children elements of the table grid.

- Table rows do support the w:rsidR attribute, so we use it as we did for runs within

paragraphs. The VID for the row is the RSID value concatenated with zero.

- Unfortunately, table cells also do not support the RSID attribute, but we need to compute a unique VID for each cell. However, we noted that every table cell contains a paragraph, even though the paragraph might be empty. So, we create a VID for each cell by computing the hash of w:tc element name and concatenating it with the RSID of the cell's first paragraph.

- For the table column property elements, the VID is the concatenation of the RSID from the cell's first paragraph with the hash of the property's element name.

```
<w:tbl>
    <w:tblPr>
        <w:tblCaption w:val="ID:F6AB2FBF"/>
        <w:tblW w:w="0" w:type="auto"/>
        <w:tblInd w:w="-8" w:type="dxa"/>
        <w:tblLayout w:type="fixed"/>
       +<w:tblCellMar>
        <w:tblLook w:noVBand="0" w:noHBand="0" w:lastColumn="0" w:firstColumn="0" w:lastRow="0" w:firstRow
            ="0" w:val="0000"/>
    </w:tblPr>
    <w:tblGrid>
        <w:gridCol w:w="3600"/>
        <w:gridCol w:w="3634"/>
    </w:tblGrid>
    <w:tr w:rsidR="00E200CD" w:rsidTr="00E200CD">
        <w:tc>
            <w:tcPr>
                <w:tcW w:w="3600" w:type="dxa"/>
               +<w:tcBorders>
                <w:shd w:val="clear" w:color="auto" w:fill="auto"/>
            </w:tcPr>
           +<w:p w:rsidR="7E80B124" w:rsidP="00E200CD" w:rsidRDefault="0011569B">
        </w:tc>
        <w:tc>
           +<w:tcPr>
           +<w:p w:rsidR="41DF3A23" w:rsidP="00E200CD" w:rsidRDefault="0011569B">
        </w:tc>
    </w:tr>
    +<w:tr w:rsidR="85AA33DD" w:rsidTr="00E200CD">
    +<w:tr w:rsidR="C2409107" w:rsidTr="00E200CD">
</w:tbl>
```

Figure 4.6: XML Listing for table in our application. This table has 3 rows, but the listing has only expanded the first

4.3 Sections

Sections are another important block-level element in an OOXML word processing document. They are responsible for grouping a number of paragraphs that have a particular

set of properties, defining the pages on which the text will appear. Section elements <w:sect> are important for storing information related to page composition, such as footer and header references, borders, margins, page orientation and page size.

```
<w:p w:rsidR="004B172F" w:rsidRDefault="004B172F" w:rsidP="004B172F">
    <w:pPr>
        <w:pageBreakBefore/>
        <w:spacing w:line="480" w:lineRule="auto"/>
        <w:jc w:val="center"/>
        <w:sectPr w:rsidR="004B172F">
            <w:headerReference w:type="default" r:id="rId5"/>
            <w:footerReference w:type="even" r:id="rId6"/>
            <w:footerReference w:type="default" r:id="rId7"/>
            <w:headerReference w:type="first" r:id="rId8"/>
            <w:footerReference w:type="first" r:id="rId9"/>
            <w:pgSz w:w="12240" w:h="15840"/>
            <w:pgMar w:top="1732" w:right="1440" w:bottom="1732" w:left="2160" w:header="1440" w:footer="
                1440" w:gutter="0"/>
            <w:cols w:space="720"/>
            <w:docGrid w:linePitch="360"/>
        </w:sectPr>
    </w:pPr>
    +<w:r>
</w:p>
```

Figure 4.7: XML for Section properties

The way we create the VIDs for the Section paragraph structure is the following:

- Section properties elements <w:sectPr> do support the RSID attribute. So, similar to what we did for paragraphs, the VID is formed by concatenating the RSID of the element with zero (#00000000).

- Similar to table grid, here we have two individual elements <w:headerReference> and <w:footerReference> that can appear more than one time within the section property <w:sectPr>. The solution, however, is simpler than the table grid element. Both elements here contain a r:id attribute and we made use of those attributes to uniquely identify the elements. Thus, for <w:headerReference> and <w:footerReference> elements the VIDs is formed by the concatenation of RSID from parent tag <w:sectPr> with the hash value of r:id attribute.

- All the remaining section properties elements appear only once within the section. So, as we did for the paragraph and table properties elements, each one receives a VID by concatenation of the ID in the section properties element and the hash of the property's element name.

4.4 External Objects

One challenge that we still face involves external objects that are stored outside the main XML document file. External objects are non-textual media (images, video, animation) that Word includes as external references. As was mentioned in the Microsoft Word Structure chapter, MS Word docx files are actually zipped folders containing files and further subfolders. Figure 3.1 illustrated the structure of a docx file. For the current discussion, it is important to reinforce that there are three important elements of the zipped structure involved with the external objects, all of which are in the "word" folder:

- The "document.xml" file is the main XML content file and essentially stores the structure and textual content of the document. External media elements, such as images, are specified in document.xml using abstract reference values, like "rId3" and "rId7".

- The "media" subfolder holds media elements like images that are included by indirect reference in the document.xml file.

- The `_rels/document.xml.rels` file encode the relationships between the abstract reference values and the names of files in the media folder.

When an image is linked to in a Word document, the XML code will contain the properties of the image and an abstract reference as described above. The challenge arises because of the naming conventions for abstract references and for media files. Suppose an initial "doc0.docx" is created without any images. Then, two new documents are derived from it, "doc1.docx" and "doc2.docx", and each one adds an image. The images in these two different files are likely to have abstract references of "rId1" and to be stored in the media folder with the name "image1.emf".

Our current implementation focuses entirely on the document.xml file and ignores the media folder. Some of our future work will be to extend our processing to include these media elements, as well as the styles.xml file, which encodes named styles and is also outside our application's current scope.

4.5 Plug-in

The plugin developed by Coakley *et al.* was designed to provide a user interface that is simple and easy to understand [8]. It allows users to interact with the version-system within the Microsoft Word interface. In order to prove that our solution works, we update the plug-in developed by Coakley *et al.* to use Molhado merge algorithm with our implementation.

Figure 4.8: Microsoft Word plugin

The interface is similar to the original. It has three components: a ribbon tab, a side panel, and a history graph. The ribbon tabs contains the following buttons:

- "Commit" will allow the user to enter some comments and commit the document as a new revision.
- "Show Revisions" will display in the side panel a list of all the revisions.
- "History Graph" show the history of revisions in a directed graph in a new window. This functionality is current not working due to outdated libraries.
- "Merge" will call the Molhado merge with our set of rules in order to do the merge.
- "Generate VIDs" It will generate the VIDs our solution needs in other to merge the documents.
- "Settings" It allow you some basic configuration as enable the commit changes when closing document and selecting the path of the merge tool.

We have tested our new version of the plug-in with real examples. Below we will describe a simple example, but that involves all the main elements of a MS Word document as paragraphs, tables, properties and styles.

- We start with the *version 0*(V0) containing a single paragraph and a simple table with two columns as shown in Figure 4.9. Note that this version is already stamped with the VIDs, which can be seen in Appendix A.

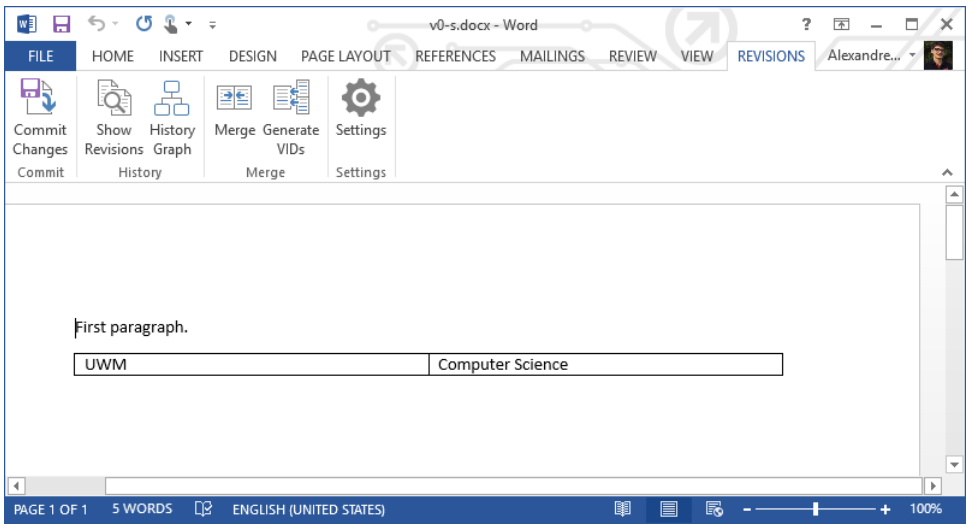

Figure 4.9: V0 MS word document. One paragraph and table with one row and two columns

- In *version 1*(V1), shown in Figure 4.10, derived document from V0, a second paragraph is added. The new paragraph is marked as bold and with underline text to add properties to the paragraph. The V1 XML document can be seen in Appendix B.

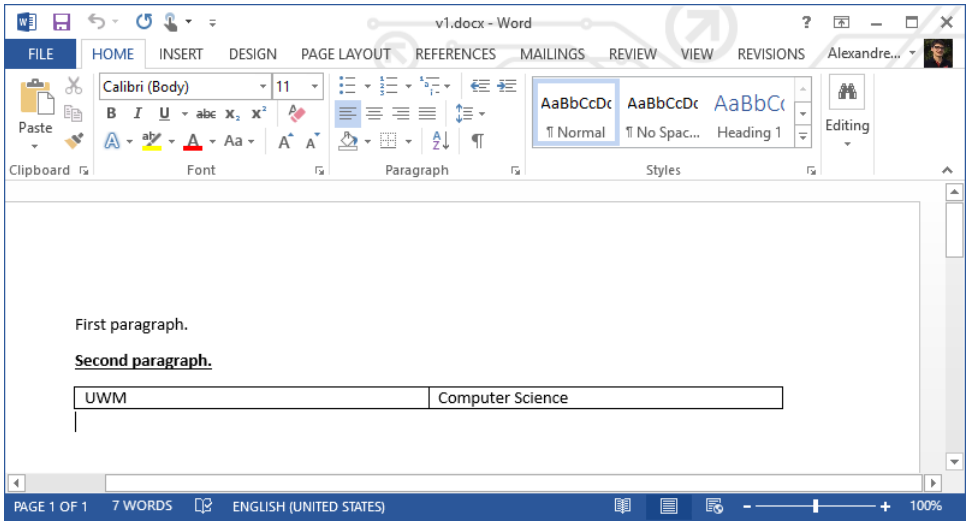

Figure 4.10: V1 MS word document. A new paragraph is added in relation to V0 on Figure 4.9.

- *Version 2*, Figure 4.11, also derived from V0, receives a new row for the existing

table. The text in the new row is marked to be centered. This is also adding more properties elements to the regular XML table text. The XML document of V2 can be seen in Appendix C.

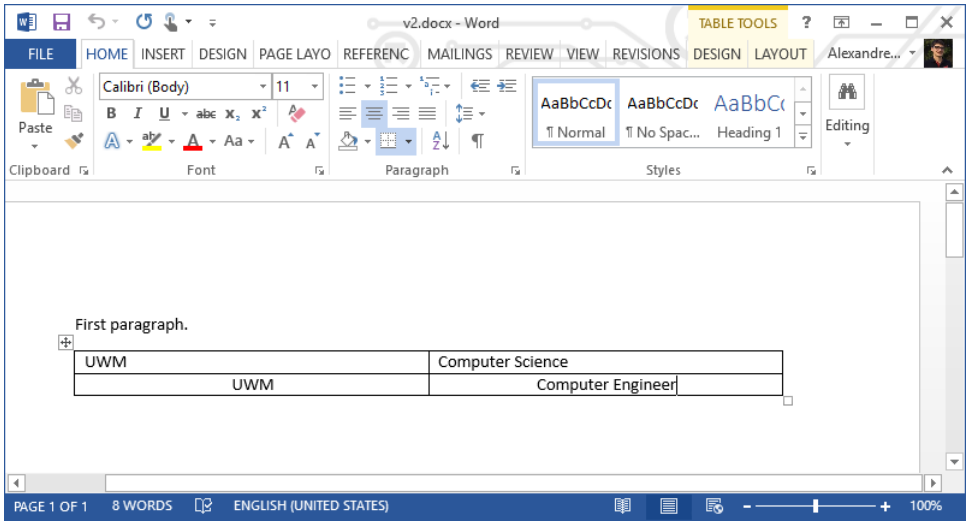

Figure 4.11: V2 MS word document. A new row on the table is added in relation to V0 on Figure 4.9.

- After merging V1 and V2, the merged document V3, Figure 4.12, contains all the changes made in V1 and V2. Including all the style properties we have added. We can see in Appendix D that all the elements from V1 and V2 are merged correctly to form V3 XML document. Showing how the our solution is being applied well within the plug-in.

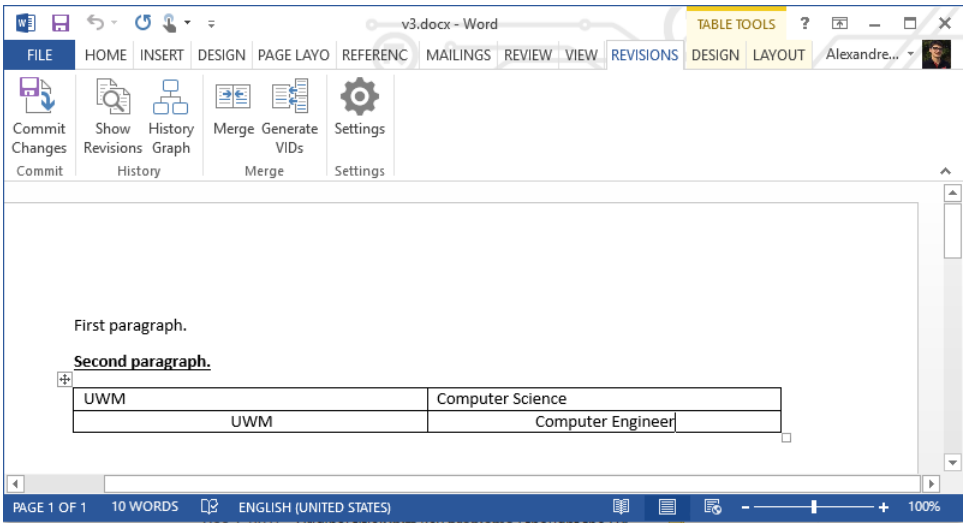

Figure 4.12: V3 MS word document. The merged result from derived documents V1(Figure 4.10) and V2(Figure 4.11).

Chapter 5

Conclusion

We have been able to develop elements of an improved Version Aware Word Document system by extending the work of Coakley *et al.* [8]. By developing strategies to define Versioning IDs for all document elements, we have made it possible to use the *molhado* XML differencing and merging algorithms developed [1]. This required identifying OOXML structures, the RSID attributes, that we could take over in order to record partial IDs on important elements. Then, to create fully usable VIDs, we defined rules by which they could be reliably created, based on the RSID values recorded in the XML file. We can be confident that these changes improve on the original work because the ID-based three-way merging algorithms are faster and more accurate than the matching-based approach used by Coakley *et al.*.

We showed that it is possible to create a structure where we can rely only on the XML elements in the Microsoft Word XML architecture. Thus, other solutions can use the same idea to rely only on the Word XML document instead of trying to embed extra XML files in the docx structure. We also note that the entire effort would be simplified if Word (and other similar applications) would provide an attribute on all elements that could be used as a unique ID.

The main limitation our solution has is the reference to external objects, such as images. Based on the fact that our implementation focuses entirely on the main textual file, the *document.xml*, we do not have a solution for keeping track when the external objects and their references change. In addition, some particular test scenarios would required the style XML file (*styles.xml*) to be merged. For example, if document V1 is edited with a specific style format that is not present in V0 style format file. Our solution does not cover those situations.

In the future, the implementation to support the external objects would be important

in order to have a system that has full support to all MS Word elements. As well as the merge of style format files. In addition, the plug-in can also have features expanded. For example, it may be possible to use the interface of "Track Changes" or "Compare Combine" to show the changes to a document.

Bibliography

[1] Cheng Thao and Ethan V Munson. Using versioned tree data structure, change detection and node identity for three-way xml merging. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 77–86. ACM, 2010.

[2] Git. `https://git-scm.com/`. Accessed: 2017-11-11.

[3] Team foundation server. `https://www.visualstudio.com/tfs/`. Accessed: 2017-11-11.

[4] Subversion. `https://subversion.apache.org/`. Accessed: 2017-11-11.

[5] Google docs. `https://docs.google.com/`. Accessed: 2017-11-11.

[6] Microsoft Office online. `https://www.office.com/`. Accessed: 2017-11-11.

[7] Dropbox. `https://www.dropbox.com/`. Accessed: 2017-11-11.

[8] Stephen M Coakley, Jacob Mischka, and Cheng Thao. Version-Aware Word Documents. In *Proceedings of the 2nd International Workshop on (Document) Changes: modeling, detection, storage and visualization*, page 2. ACM, 2014.

[9] Cheng Thao and Ethan V Munson. Version-Aware XML documents. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 97–100. ACM, 2011.

[10] Office Open XML. `http://officeopenxml.com/`. Accessed: 2017-11-03.

[11] Tancred Lindholm. A three-way merge for xml documents. In *Proceedings of the 2004 ACM symposium on Document Engineering*, pages 1–10. ACM, 2004.

[12] Meenu Pandey and Ethan V Munson. Version-Aware LibreOffice documents. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 57–60. ACM, 2013.

[13] Iso. Information technology - Document description and processing languages - Office Open XML File Formats ISO/IEC 29500-1:2016, 2016.

[14] Walkthrough: Word 2007 XML Format. https://msdn.microsoft.com/en-us/library/bb266220(v=office.12).aspx. Accessed: 2017-11-01.

[15] Microsoft Office Word 2003 documentation. https://msdn.microsoft.com/en-us/library/ee364478(v=office.11).aspx. Accessed: 2017-05-27.

Appendices

Appendix A: V0 XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document mc:Ignorable="w14 w15 wp14"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
    xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
    xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape" >
    <w:body>
        <w:p w:rsidR="00963C28" w:rsidRDefault="00345B34" >
            <w:r w:rsidRPr="33ED3E7E">
                <w:t>First paragraph.</w:t>
            </w:r>
        </w:p>
        <w:tbl>
            <w:tblPr>
                <w:tblCaption w:val="ID:D4A9356E"/>
                <w:tblStyle w:val="TableGrid" />
                <w:tblW w:type="auto" w:w="0" />
                <w:tblLook w:firstColumn="1" w:firstRow="1" w:lastColumn="0" w:lastRow="0" w:noHBand="0" w:noVBand="1" w:
                    val="04A0" />
            </w:tblPr>
            <w:tblGrid>
                <w:gridCol w:w="4414" />
                <w:gridCol w:w="4414" />
            </w:tblGrid>
            <w:tr w:rsidR="009858B0" w:rsidTr="009858B0" >
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:type="dxa" w:w="4414" />
                    </w:tcPr>
                    <w:p w:rsidR="9718FD83" w:rsidRDefault="009858B0" >
                        <w:r w:rsidRPr="1517EBEC">
                            <w:t>UWM</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:type="dxa" w:w="4414" />
                    </w:tcPr>
                    <w:p w:rsidR="D8B0A603" w:rsidRDefault="009858B0" >
                        <w:r w:rsidRPr="E0CB0F5D">
                            <w:t>Computer Science</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
        </w:tbl>
        <w:p w:rsidR="8D1F5435" w:rsidRDefault="009858B0" />
        <w:sectPr w:rsidR="8D1F5435">
            <w:pgSz w:h="15840" w:w="12240" />
            <w:pgMar w:bottom="1417" w:footer="720" w:gutter="0" w:header="720" w:left="1701" w:right="1701" w:top="1417"
                />
            <w:cols w:space="720" />
            <w:docGrid w:linePitch="360" />
        </w:sectPr>
    </w:body>
</w:document>
```

Appendix B: V1 XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document
    xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
    xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
    xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape" mc:Ignorable="w14 w15 wp14">
    <w:body>
        <w:p w:rsidR="00963C28" w:rsidRDefault="00345B34">
            <w:r w:rsidRPr="33ED3E7E">
                <w:t>First paragraph.</w:t>
            </w:r>
        </w:p>
        <w:p w:rsidR="003538C1" w:rsidRPr="003538C1" w:rsidRDefault="003538C1">
            <w:pPr>
                <w:rPr>
                    <w:b/>
                    <w:u w:val="single"/>
                </w:rPr>
            </w:pPr>
            <w:r w:rsidRPr="003538C1">
                <w:rPr>
                    <w:b/>
                    <w:u w:val="single"/>
                </w:rPr>
                <w:t>Second paragraph.</w:t>
            </w:r>
            <w:bookmarkStart w:id="0" w:name="_GoBack"/>
            <w:bookmarkEnd w:id="0"/>
        </w:p>
        <w:tbl>
            <w:tblPr>
                <w:tblStyle w:val="TableGrid"/>
                <w:tblW w:w="0" w:type="auto"/>
                <w:tblLook w:val="04A0" w:firstRow="1" w:lastRow="0" w:firstColumn="1" w:lastColumn="0" w:noHBand="0" w:
                    noVBand="1"/>
                <w:tblCaption w:val="ID:D4A9356E"/>
            </w:tblPr>
            <w:tblGrid>
                <w:gridCol w:w="4414"/>
                <w:gridCol w:w="4414"/>
            </w:tblGrid>
            <w:tr w:rsidR="009858B0" w:rsidTr="009858B0">
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="9718FD83" w:rsidRDefault="9718FD83" w:rsidP="009858B0">
                        <w:r w:rsidR="1517EBEC">
                            <w:t>UWM</</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="D8B0A603" w:rsidRDefault="D8B0A603" w:rsidP="009858B0">
                        <w:r w:rsidR="E0CB0F5D">
                            <w:t>Computer Science</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
        </w:tbl>
        <w:p w:rsidR="8D1F5435" w:rsidRDefault="8D1F5435" w:rsidP="003538C1"/>
        <w:sectPr w:rsidR="8D1F5435" w:rsidSect="003538C1">
            <w:pgSz w:w="12240" w:h="15840"/>
            <w:pgMar w:top="1417" w:right="1701" w:bottom="1417" w:left="1701" w:header="720" w:footer="720" w:gutter="0"/>
            <w:cols w:space="720"/>
            <w:docGrid w:linePitch="360"/>
        </w:sectPr>
    </w:body>
</w:document>
```

Appendix C: V2 XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document
    xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    ...
    mc:Ignorable="w14_w15_wp14">
    <w:body>
        <w:p w:rsidR="00963C28" w:rsidRDefault="00345B34">
            <w:r w:rsidRPr="33ED3E7E">
                <w:t>First paragraph.</w:t>
            </w:r>
        </w:p>
        <w:tbl>
            <w:tblPr>
                <w:tblStyle w:val="TableGrid"/>
                <w:tblW w:w="0" w:type="auto"/>
                <w:tblLook w:val="04A0" w:firstRow="1" w:lastRow="0" w:firstColumn="1" w:lastColumn="0" w:noHBand="0" w:
                    noVBand="1"/>
                <w:tblCaption w:val="ID:D4A9356E"/>
            </w:tblPr>
            <w:tblGrid>
                <w:gridCol w:w="4414"/>
                <w:gridCol w:w="4414"/>
            </w:tblGrid>
            <w:tr w:rsidR="009858B0" w:rsidTr="009858B0">
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="9718FD83" w:rsidRDefault="009858B0">
                        <w:r w:rsidRPr="1517EBEC">
                            <w:t>UWM</</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="D8B0A603" w:rsidRDefault="009858B0">
                        <w:r w:rsidRPr="E0CB0F5D">
                            <w:t>Computer Science</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
            <w:tr w:rsidR="00BC50B6" w:rsidTr="009858B0">
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="00BC50B6" w:rsidRPr="1517EBEC" w:rsidRDefault="00BC50B6" w:rsidP="00BC50B6">
                        <w:pPr>
                            <w:jc w:val="center"/>
                        </w:pPr>
                        <w:bookmarkStart w:id="0" w:name="_GoBack"/>
                        <w:bookmarkEnd w:id="0"/>
                        <w:r>
                            <w:t>UWM</</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    <w:tcPr>
                        <w:tcW w:w="4414" w:type="dxa"/>
                    </w:tcPr>
                    <w:p w:rsidR="00BC50B6" w:rsidRPr="E0CB0F5D" w:rsidRDefault="00BC50B6" w:rsidP="00BC50B6">
                        <w:pPr>
                            <w:jc w:val="center"/>
                        </w:pPr>
                        <w:r>
                            <w:t>Computer Engineer</</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
        </w:tbl>
        <w:p w:rsidR="8D1F5435" w:rsidRDefault="00BC50B6"/>
        <w:sectPr w:rsidR="8D1F5435">
            <w:pgSz w:w="12240" w:h="15840"/>
            <w:pgMar w:top="1417" w:right="1701" w:bottom="1417" w:left="1701" w:header="720" w:footer="720" w:gutter="0"/>
            <w:cols w:space="720"/>
            <w:docGrid w:linePitch="360"/>
        </w:sectPr>
    </w:body>
</w:document>
```

Appendix D: V3 XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document mc:Ignorable="w14 w15 wp14"
    ...
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape" >
    <w:body>
        <w:p w:rsidR="00963C28" w:rsidRDefault="00345B34" >
            <w:r w:rsidRPr="33ED3E7E" >
                <w:t>First paragraph.</w:t>
            </w:r>
        </w:p>
        <w:p w:rsidR="003538C1" w:rsidRDefault="003538C1" >
            <w:pPr>
                <w:rPr>
                    <w:b />
                    <w:u w:val="single" />
                </w:rPr>
            </w:pPr>
            <w:r w:rsidRPr="003538C1">
                <w:rPr>
                    <w:b />
                    <w:u w:val="single" />
                </w:rPr>
                <w:t>Second paragraph.</w:t>
            </w:r>
        </w:p>
        <w:tbl>
            <w:tblPr>
                <w:tblCaption w:val="ID:D4A9356E"/>
                <w:tblStyle w:val="TableGrid" />
                <w:tblW w:type="auto" w:w="0" />
                <w:tblLook w:firstColumn="1" w:firstRow="1" w:lastColumn="0" w:lastRow="0" w:noHBand="0" w:noVBand="1" w:
                    val="04A0" />
            </w:tblPr>
            +<w:tblGrid>
            <w:tr w:rsidR="009858B0" w:rsidTr="009858B0" >
                <w:tc>
                    +<w:tcPr>
                    <w:p w:rsidR="9718FD83" w:rsidRDefault="009858B0" >
                        <w:r w:rsidRPr="1517EBEC" >
                            <w:t>UWM</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    +<w:tcPr>
                    <w:p w:rsidR="D8B0A603" w:rsidRDefault="009858B0" >
                        <w:r w:rsidRPr="E0CB0F5D" >
                            <w:t>Computer Science</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
            <w:tr w:rsidR="00BC50B6" w:rsidTr="009858B0" >
                <w:tc>
                    +<w:tcPr>
                    <w:p w:rsidR="EA25A9E6" w:rsidRP="00BC50B6" w:rsidRDefault="00BC50B6" >
                        <w:pPr>
                            <w:jc w:val="center" />
                        </w:pPr>
                        <w:r w:rsidRPr="52EC6D5B">
                            <w:t>UWM</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
                <w:tc>
                    +<w:tcPr>
                    <w:p w:rsidR="FE5BA188" w:rsidRP="00BC50B6" w:rsidRDefault="00BC50B6" >
                        <w:pPr>
                            <w:jc w:val="center" />
                        </w:pPr>
                        <w:r w:rsidRPr="452A67C9">
                            <w:t>Computer Engineer</w:t>
                        </w:r>
                    </w:p>
                </w:tc>
            </w:tr>
        </w:tbl>
        <w:p w:rsidR="8D1F5435" w:rsidRDefault="00BC50B6" />
        <w:sectPr w:rsidR="8D1F5435">
            <w:pgSz w:h="15840" w:w="12240" />
            <w:pgMar w:bottom="1417" w:footer="720" w:gutter="0" w:header="720" w:left="1701" w:right="1701" w:top="1417"
                />
            <w:cols w:space="720" />
            <w:docGrid w:linePitch="360" />
        </w:sectPr>
    </w:body>
</w:document>
```