Automated Braille Production from Word-Processed Documents

Paul Blenkhorn and Gareth Evans

***Abstract*—This paper describes a novel method for automatically generating Braille documents from word-processed (Microsoft Word) documents. In particular it details how, by using the Word Object Model, the translation system can map the layout information (format) in the print document into an appropriate Braille equivalent.**

***Index Terms*—Blindness, Braille, document handling, visual impairment.**

I. Introduction

THE BRAILLE code is the main system for the majority of those blind people who read and write using tactile means. The characteristics of Braille have been described elsewhere [1], [2], and, in this paper, the authors assume that a reader has a reasonable knowledge of the basics of Braille representation and production.

There are a number of commercial software-based systems that translate from Braille into computer-readable text. An algorithm for carrying out this translation is described in [1]. There are also commercial systems that translate from computer-readable, text-based documents into Braille. Braille hardcopy may be produced, either by printing using a Braille printer, or by driving a Braille display. The algorithms for translating from text and both Grade 1 and Grade 2 (contracted) Braille are well understood and such an algorithm is given in [2].

This paper is concerned with the translation of text documents to Braille documents. Specifically, it considers how users of a word processor can produce printed Braille documents, when they have little or no knowledge of Braille. This situation is relatively common, for example a teacher producing material for a blind pupil in a mainstream school or an office worker copying a document to a blind colleague.

The work described in this paper is not, by any means, the first to address this problem and other approaches used by others are described in Section III. However, our approach is unique in the way that the translator is integrated with the word processor. The advantage of this approach is that it makes the process very easy for the user, who simply starts translation from a menu item in MSWord. As we will discuss later, it also makes the approach relatively "future proof." The downside of this approach is that the user is tied to one particular word processor. We attempt to address this problem by integrating the translation engine into Microsoft Word, which has extremely high market penetration and, consequently high availability, worldwide.

The work also has a degree of novelty in that it seeks to preserve the format of the original text document in the Braille copy. For example, the formats of tables are maintained in the Braille document. We discuss this issue in Section II.

II. The Format of Braille Documents

A typical Braille document will consist of an optional title page that has centered text followed by a series of pages, each of which follows a common format [3], [4]. All Braille characters have the same size and so on each page, the Braille characters are placed on equally spaced lines with no subscripts, superscripts or changes in font size. Apart from the title page, each of the pages will typically have the following.

1) A title line, which will normally include:
   - a left-hand column that holds the print page number that corresponds to the current Braille page. As written text is very much more compact than Braille, there will be, of course, a number of Braille pages that correspond to one text page;
   - a centered title that contains both the header and footer of the text document;
   - a right justified Braille number that is the number of the current Braille page.
2) A set of Braille paragraphs that have no blank lines between them. The first line of each paragraph generally starts with two spaces to indicate the start of a paragraph. Alternatively, some users prefer to have the start of a paragraph indicated by a blank line, the first line of Braille in the paragraph is not indented in this case. The latter convention does not strictly follow the Braille standard [3].
3) Italics, bold, underline and other font changes are indicated in Braille by using an italics sign (the Braille characters corresponding to "I"). Although marking in this way is currently under review, it is sufficient here to detail its past use. Any changes are relatively trivial to incorporate into the translation system. If a single word is to be highlighted, a single italics sign is placed before the word. If more than three consecutive words are to be highlighted, two italics signs are placed before the first word in the sequence and one italics sign is placed before the last word in the sequence.

Manuscript received April 17, 2000; revised June 26, 2000, September 12, 2000, and October 9, 2000.

The authors are with The Centre for Rehabilitation Engineering, Speech and Sensory Technology (CRESST), Department of Computation, UMIST University of Manchester Institute of Science and Technology, Manchester, M60 1QD U.K. (e-mail: p.blenkhorn@co.umist.ac.uk).

Publisher Item Identifier S 1534-4320(01)01773-9.

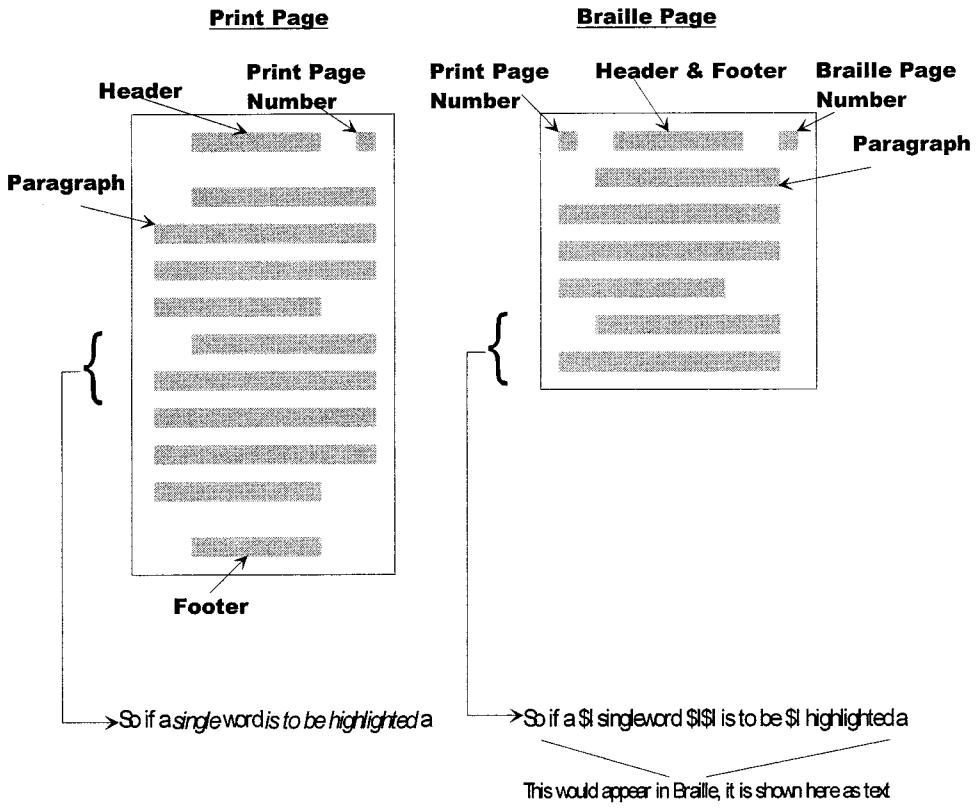

Fig. 1. The format of text and Braille Documents.

Fig. 1 illustrates the formatting of a standard text document and a Braille document. Braille is laid out in a relatively compact manner, with a minimum of blank lines and spaces. Even so, Braille documents are much bulkier than their text equivalents. One reason for this is that Braille paper is generally much thicker than the paper used for text documents. In addition, a Braille cell is much larger than the character font sizes used in text documents. A typical Braille page will have 1000 characters on the page (25 lines at 40 characters per page). An approach that attempts to almost halve the size of Braille documents is to print the document in *double-sided interpoint* format. Braille characters are printed on both sides of the page, with one side slightly offset with respect to the other, so that the Braille cells do not coincide. The reader is capable of reading the raised dots that form the text on the page that he/she is reading, whilst ignoring the indentations produced by the Braille on the other side of the page. Another approach to reduce the amount of space taken by the Braille, in some, but not all languages, is to apply a language-specific set of contextual rules that allow certain words, or parts of words, to be represented by less Braille cells than the direct text equivalent [1], [2]. This is generally referred to as contracted Braille (for English Braille this is also referred to as Grade II Braille). For example, in Grade II English Braille the sequence of Braille characters DCV represents the word "deceive."

Clearly, given the complexity and variety of documents produced by modern word processors, there are a number of difficulties and compromises in print to Braille format transformations.

III. Earlier Work on Converting the Format of Braille to Print

A. Introduction

To translate from text to Braille a system has to carry out two functions.

- Translate the text into a suitable set of characters that can be subsequently sent to the Braille printer. This process must, when desired, produce contracted Braille.
- Arrange to drive the printer so that the Braille document is handled correctly.

B. Early Work

The basic algorithms for computerized Braille translation have been around since the late 1960s [5]. These systems were designed specifically for producing Braille documents. First, the text was translated into Braille. Secondly, a Braille Formatting module converted the source text produced into a file suitable for driving a Braille printer. The text was formatted according to a set of Braille formatting codes that were embedded in the source text to indicate centered text, italics signs, etc. The codes had to be entered into the original text by the person who was producing the Braille document. This person, therefore, had to have a good understanding of Braille formatting.

C. Requirements for Braille Formatting

As computers became more widespread, and particularly with the advent of personal computers, word processing became more prevalent. This allowed blind computer users to produce

Braille versions of documents from computer-based sources and allowed nonspecialists to produce Braille documents; for example, an office worker producing a document for a blind colleague. Thus, the printing of Braille documents moved from a specialist document preparation to a standard word processing operation with the hard copy produced by a different type of printer.

In essence, there are three requirements to produce formatted Braille output from a word processor as follows.

1) The user of the word processor should not have to understand Braille formatting and he/she should not have to insert Braille.
2) The user of the word processor should not need to use significant amounts of additional software. Preferably he/she should be able to carry out the operation from within the word processor.
3) The approach should be resilient to changes in versions of the word processor. Over recent years, the word processors seem to have an operational life of around three to four years.

D. Approaches to Braille Formatting from Word Processors

Early Braille transcription packages made significant use of formatting codes that were inserted by the user into the source (text) document to control the formatting of the Braille document. Thus, the formatting information was explicitly given in the file typed by the user. The file produced by the word processor was, in effect, an ASCII file with certain character sequences used to indicate formatting. The Braille translation would typically take place in two passes. In the fist pass the text would be converted into Braille, with the formatting codes passing through unchanged. The second pass would then layout the Braille document in the manner indicated by the embedded formatting codes.

More recently, the requirement for the user to explicitly insert formatting codes has been removed, with the word processor providing a WYSIWYG[1] interface. In such systems, the formatting codes are hidden from the user and are implicitly inserted by the word processor in response to user options. This results in formating codes being inserted into the word processor's files, but this information is hidden from the user. The formatting information is used by the word processor to format the document in a suitable form for display on the screen or to drive a printer. One strategy to deal with such systems is for the Braille formatting program to read the formatting codes held in the word processor files[2] and to use this information to format the Braille for printing. The problem with this approach is that is tightly coupled to the word processor, because it uses the proprietary formatting codes. Over the years, different versions of ostensibly the same word processor can, and often have had, different file formats. Consequently, when a new word processor version is released, the Braille formatting program often needs to be updated. It also means that the formatting program is vendor-specific.

[1]What you see is what you get.

[2]In the early days, this was often WordStar. Later WordPerfect was used. These days a number of systems also support MS Word, for example, the Duxbury Braille Translator http://www.duxburysystems.com/.

To make the formatting program version and vendor independent, the formatting program can be made to operate on a standard, nonproprietary format. Most modern word processors can produce documents in rich text format (RTF). This is a vendor-independent format that contains formatting information. Therefore, a Braille formatting program can be written to process RTF files. This approach has the advantage of maintaining vendor and version independence. The only major difficulty that remains is writing a reliable formatter for RTF files. While this is a complex task, it is certainly tractable. The formatting software can be integrated into a word processor by writing a relatively simple macro.

A closely related approach is to translate the print file produced by the word processor, rather than the word processor's native files. This approach is also vendor and version independent. It also requires the development of a relatively complex formatting program [6]. This is, probably, a more complex task than writing the equivalent software for the files in the word processor format. One of the reasons for this is that the underlying structure of the document is lost. Therefore, it is can difficult to determine certain formatting features such as new paragraphs and tables.

IV. The Braille Out System

The approach adopted by the authors, which has resulted in a program called Braille Out, is quite radically different to those described above, it uses features that have become available in the past few years. This approach exploits facilities offered by the word processor itself via a defined programming interface, rather than operating on files produced by the word processor. In effect, the document appears to the formatting program as an object, that supplies a set of operations and attributes that can be used to enquire about the document's structure and to produce a new document in a form that can be sent to a Braille printer. The approach has the advantage of making the formatting program significantly simpler than those rely on processing the underlying files. This is because the object will explicitly provide information about a document's structure on demand. The drawback of the approach is that it is vendor specific, but probably not version specific because the word processor objects tend to offer backward compatibility between releases. Thus, the approach described is tied to one word processor, in this case Microsoft Word. However, as noted earlier, Word has very high market penetration and availability and, thus, the approach addresses the needs of a significant number of users.

A. Introduction to the Approach

Microsoft Word can be treated as a "word processing object" that allows programs to access methods and attributes. This can be achieved either by using OLE (object linking and embedding) to embed the MSWord object in a formatting program, or by writing the formatting program as a Visual Basic macro from within Word itself. In either case the formatting program has access to the documents and is capable of performing operations on them. The approach described uses Visual Basic macros because, as we discuss below, this is very much faster than using OLE. From the user's perspective he/she simply sees an additional menu item from the File menu.

B. The MS Word Object

The features of the Word Object model are given elsewhere [7]. However, in short Visual Basic supports a set of objects that correspond directly to elements in Word. There are objects that represent an open document, bookmarks, tables, paragraphs, etc. Indeed, "*Every type of element in Word—documents, tables, paragraphs, bookmarks, fields, and so on—can be represented by an object in Visual Basic*" [7]. These objects expose methods and properties that can be used to automate tasks in MS Word.

The objects of most relevance to this work are:

- The document object, which is used to open, close and indicate which open document is to be manipulated.
- The paragraph collection of objects, this contains all of the paragraphs in the document. Each paragraph includes information about headers, footers, lists, paragraph alignment (left, justified, centered, etc.) and, of course, the actual text and its font information.
- The tables collection, which contains all of the tables in the documents.
- The shapes collection, which contains information about the Word shapes, including those that contain text such as *AutoShapes*.
- The find object, which is used for find and replace operations.

C. The Braille-Out System

As noted in Section IV-A there are two approaches that can be used to access the Word Object Model, OLE and a Word macro. The authors have tried both approaches in writing Braille Out. Both use a fast dynamic link library (DLL), written in C, to translate the text into Braille. A Visual Basic (Version 6) program is used to interface between MS Word and the translation DLL.

The first approach to this problem was to use a stand-alone Visual Basic program that controlled Word by using OLE. This proved to be unacceptably slow—an early version of the algorithm running on a Pentium processor at 233 MHz took over a minute to translate and format a small document of some 203 words (in five paragraphs). By way of contrast the final version of Braille Out, which is written as a Word macro, takes just over a minute to translate and format the text in this paper (over 3500 words and 150 paragraphs) on the same specification of machine. It is believed that this is due to the Visual Basic program and the Word Object being in different contexts; Word is often quite slow when operations involve interprocess communications,[3] and Braille Out accesses the Word documents many times to analyze and translate different aspects of the print format. The problem is particularly acute because the formatting program needs to examine every paragraph a word at time to determine the changes in font style that need to be mapped into italics in Braille (see Section II). In an attempt to overcome the out-of-context overheads, the Visual Basic code was moved from the stand-alone program into a Word macro.

In the Braille-Out system the file to be translated is opened as one document. A second document, which is blank and which has a page and font size that matches the Braille document, is also opened. The formatted Braille text is inserted into this second document. Braille Out processes the first document in several passes converting both the text into Braille and the text formatting into a corresponding Braille format. Of particular interest is the capability of this approach to readily detect and appropriately convert document elements, such as the print page numbers, (floating) text boxes, tables and lists.

A high-level pseudocode description of the algorithm is given in the Appendix.

[3]A discussion on this issue is beyond the scope of this article. For further information on in-context and out-of-context operations can be obtained from the Microsoft Developers Library.

TABLE I
FORMATTING TIME FOR DOCUMENTS OF VARIOUS LENGTHS

Document Size	Time (Minutes)
525 lines,156 paragraphs, 3648 words, 19286 characters	1:03
1050 lines,312 paragraphs, 7286 words, 38572 characters	2:27
2100 lines,624 paragraphs, 14572 words, 77144 characters	6:33

V. RESULTS AND CONCLUDING REMARKS

The current version of Braille Out produces reliable contracted literary Braille from a wide variety of documents. The layout is deemed to be good and speed is acceptable. However there are a number of issues that need to be commented upon.

First, while the speed is generally acceptable, very large documents can cause the system to slow down unacceptably. The relationship between processing speed and document size is nonlinear. This can be demonstrated by the figures given in Table I. These were derived by noting the formatting time for three documents in the size ratio $1:2:4$. The document used was a draft version of this paper; copying and pasting the document at its end created longer versions. The machine used was a Toshiba Portégé subnotebook running at 233 MHz with 128 MB of RAM.

A very long document can take several hours. The reason for this is unclear at present. The only solution for Braille Out is to suggest an upper limit on the size of documents to users.

Word creates the Braille document before printing and inserts Braille page numbers in the page header by using the automatic page numbering facility. This causes a problem because Word produces a print number (e.g., 12) rather than the correct Braille number (which for 12 is #AB). This is, of course, incorrect. However, initial user feedback indicates that this is not too significant, as it is clear from the Braille what the actual page number should be. It is intended to correct this shortfall in a future release of the software.

A further minor problem concerns the use of embedded Word documents. Currently Braille Out does not support these. Finally, Braille Out does not currently support the capitals sign. This has been omitted because it is not commonly used in the U.K. Again, this is a shortfall, which will be corrected, in a future update.

Office 2000 offers support for multi-lingual Braille translation because it allows portions of a document to be marked as being of a particular language. Although our translation algorithm is multilingual [2], we do not currently change languages automatically.

APPENDIX

```
The Braille Conversion Algorithm
Open text document to translate
Create blank Braille document
Initialize Braille document header text
Initialize Braille document page numbers
For Each text box with text in document Do
  Convert text boxes into frames
End For
For Each table Do
  Convert each row into a set text fields
  separated by spaces
End For
For Each list (includes bullets and list
  numbers) Do
  Convert to text
End For
Using the Find object
  convert multiple spaces into single
  spaces
  convert tabs into three spaces
  remove multiple carriage returns
End Using
For Each paragraph In text document DO
  If hard page break in text Then
    put hard page break in Braille
  End If
End For
If print page number changed Then
  change print page number on Braille page
End If
If header changed Then
  change Braille page header
End If
Insert two spaces in Braille document
For Each word In paragraph Do
  If word is Bold or Italics or Underlined
  Then
    Insert italics information into text
    convert modified paragraph into
  Braille
    insert in Braille document
  End If
  If text paragraph alignment is full jus-
  tification Then ' full justification not
  valid in Braille
    Set Braille paragraph alignment to
    left justified
  Else
    Set Braille paragraph alignment to
    same as text
  End If
End For
Print Braille document To Braille printer
  ' emboss Braille
```

REFERENCES

[1] P. Blenkhorn, "A system for converting Braille into print," *IEEE Trans. Rehab. Eng.*, vol. 3, pp. 215–221, June 1995.

[2] ——, "A system for converting print into Braille," *IEEE Trans. Rehab. Eng.*, vol. 5, pp. 121–129, June 1997.

[3] British National Uniform Type Committee, *A Restatement of the Lay-Out, Definitions and Rules of the Standard English Braille System*. London: Royal National Institute for the Blind, 1955.

[4] American Association of Workers for the Blind and Association for the Education of the Visually Handicapped, *English Braille, American Edition*. Louisville, KY: American Printing House for the Blind, 1970.

[5] R. A. J. Gildea, G. Hubner, and H. Werner, Eds., "Computerised Braille production," in *Proceedings of the 1. International Workshop in Munster (Germany), March 1973*. Munster, Germany: Rechenzentrum der Universitat Munster, 1974.

[6] J. E. Sullivan, "Conversion of print format for Braille," in *Proc. 6th International Workshop on Computer Applications for the Visually Handicapped*, 1990, pp. 1–14.

[7] Microsoft Corporation, "Microsoft office 97/visual basic programmer's guide," Microsoft Corporation, 1996.

Paul Blenkhorn received the B.Sc.(Hons.) degree in mathematics from the University of Manchester (UMIST), U.K.

He has he has been an active developer of systems for people with disabilities for the past 18 years and worked at the Open University and at the Research Centre for the Visually Handicapped at the University of Birmingham, U.K. He was Co-Founder and Research Director of Dolphin Systems. He joined UMIST in 1991 and is currently a Senior Lecturer in the Department of Computation. Together with G. Evans, he is Co-Director of the Centre for Rehabilitation Engineering, Speech and Sensory Technology. He has broad research interests in the area of technology and people with disabilities.

Gareth Evans received the B.Sc. (Hons.) degree in electrical and electronic engineering and the Ph.D. degree in computation from the University of Manchester (UMIST), U.K.

He joined UMIST in 1987 and is currently a Senior Lecturer in the Department of Computation. Together with P.l Blenkhorn, he is a Co-Director of the Centre for Rehabilitation Engineering, Speech and Sensory Technology. His research interests include alternative interfaces to computers for people with disabilities, speech synthesis and training, and assistive devices for people with a range of disabilities.