Application-independent representation of text for document processing— —will Unicode suffice?

Frank Mittelbach and Chris Rowley

September 15, 1999

1 Abstract

The large number, and the importance, of applications that support multi-lingual document processing have shown that the current methods and concepts for representing textual data in document processing systems are deficient in various respects.

One major advance in this area is the development of the Unicode standard ([UC95]) for text character encoding. While this, or an similar updated standard, will play a prominent role in overcoming various problems in this area, contrary to common belief it is by no means all that is needed.

This paper describes the requirements of an application-independent representation of textual data that provides a well-defined interface for applications and processes that act on such data. It shows what additions are needed to the Unicode standard for text character encoding [UC95] in order to provide a concrete syntax for such a representation.

2 Introduction

Experience with the production and maintenance of document processing systems that support multi-lingual texts has led to an analysis of the internal representation of the textual content of the document. Although this started from the requirement of supporting high quality typesetting, the result of this study was the ability to specify a representation of textual data with wider uses: this is the 'transient representation' introduced in Section 3. This representation must support all the transformations and applications that act on the logical text in a document processing system, in the widest sense. These aspects of document processing are described in Section 3 which leads, in Section 4, to an outline of the textual data that the 'transient representation' must encode.

The paper concludes, in Section 5, with a description of how Unicode should be extended to fully support such an encoding of textual data in multi-lingual documents, at least for text using an alphabetic writing system. At the level of individual characters, Unicode itself resolves the ambiguities that arise from the use of many distinct 8-bit encodings.

However, similar problems arise on a slightly higher level due to other implicit assumptions; and the information required to resolve these is largely not encodable in Unicode. For example, Unicode does not encode the language used for a bit of text; this information is essential data for processes such as hyphenation or spell-checking in multi-lingual documents, thus Section 4.3 introduces the marking of all text by a 'language-label'.

The Unicode standard [UC95] covers a very wide range of writing systems; this paper relates directly to only some of these and, even in relation these, it is not a critique of Unicode as a standard for the encoding of logical characters. It is a study of what extra is needed to define an application-independent and formatting-independent (i.e. logical) representation of textual data.

Further information about Unicode and its relationship to current document exchange representations is given by André and Goosens [AG95].

3 Representations and transformations

The purpose of this section is to summarise the types of transformations that operate on the application-independent representation described in this paper and to highlight the relevant properties of some of them. This will lead to an analysis of the extra structures that need to be added to the text in order to support the data input and output of these transformations. These transformations can be classified as logical or visual transformations, or as mixtures which can be decomposed into logical and visual components.

However, it is first necessary to consider the larger context of the primary aims of a document processing system; this will lead to an outline of the need for a richer representation of the textual content.

3.1 Models of text processing

At a high level of abstraction the manipulation of textual data in a document processing system can be described as a series of transformations that alter the document representation. This usually involves enriching the document content by adding information; this extra information is derived from the current representation by the transforming process, using information external to the document.

At the start of the process the textual part of the document may exist in an 'exchange representation', e.g., in a form intended to be used as a transfer medium between different environments and applications. A simple[1] and common exchange representation would be straight ASCII text; in the future, Unicode may become the norm.

A possible 'final' form for the document is a 'visual representation', such as a page description language, not intended for further processing other than rendering on some output device.

Between these extreme points, i.e. inside all but the simplest document processing application, the text exists in a 'transient representation' on which various transformations operate.

[1] Note that 'simple' here refers only to the explicit representation of textual data; the complete document will typically also contain mark-up, e.g. in SGML or LaTeX.

These representations do not necessarily need to be distinct: in a simple model, as used by some types of document processing systems, the exchange representation of the text is acted on directly (in the abstract model) by various applications such as spell-checkers, and then finally used by the formatter to produce the formatted document.

In a more complex model the text is first transformed from the exchange representation into an internal, transient representation that is richer and so better supports a wide variety of applications including spell-checkers. This support includes providing the necessary inputs to these applications and being able to store the output from them, such as hyphenation points within words. The formatter then acts on this richer representation, using the extra information it contains, to produce the formatted document as in the simple model.

Section 5 describes the necessary features of this transient representation as application-independent extensions of the Unicode standard.

3.2 Logical transformations

Here is a list of the more well-known applications that act on the text in a document. They can all be reduced to transformations that must act on a logical representation of the text of the document rather than a visual one; in particular, they must act on a representation which is either completely independent of visual information, such as font usage and line- and page-breaks, or one in which such visual features can be ignored.

- Hyphenation.
- Spell-checking.
- Grammar/Style-checking (and context-dependent spell-checking).
- Resolution of cross-references.
- Lexical sorting (for indexes and bibliographies)
- Cross references to logical elements.
- Generation of structural summaries.

These transformations require no information that is not present in, or directly deducible from, the (complete) logical form of the document; they have no dependency on any properties of a formatting of the document.

A useful further categorisation of these transformations comes from considering their information requirements.

3.2.1 Local transformations

Examples of these include: finding sub-words; finding word-division points (hyphenation); spell-checking. They are characterised by the following properties.

- They do not need information from the formatted document.
- They do not need information from other parts of the document.

- They often depend on identifying words and sub-words within the text;
- They often depend on the language in which each word is written (note that this can vary, even within a paragraph).

Sub-word marking For many languages, called agglutinative—a well-known example being German, this is essential for efficient hyphenation and spell-checking. It includes both identifying the constituents of compound words and identifying prefixes and suffixes that should be treated similarly.

Hyphenation-point marking Finding allowed, and preferred, division points (and recording the alternate spellings required if a division is made) is a process similar to finding sub-words.

Spell-checking Spell-checking needs to be able to recognise words and also uses information about sub-words.

More sophisticated software such as grammar- or style-checkers need further information about the phrase and sentence structure of the text; such software could also be used to resolve cases where hyphenation points and sub-word marking are not defined unambiguously when considering a word alone, with no context.

3.2.2 Global transformations

Examples of these include: resolution of cross-references to elements of the logical structure; resolution of citations; production of structural summaries. They are characterised by the following properties.

- They do not need information from the formatted document.
- They do need information from other parts of the document (and external databases or other documents).
- They can depend on the language in use.

Cross references to logical objects These are references to things such as subsection numbers (or titles), figure numbers and equation numbers, but not to page numbers. This process can include providing the information needed to add hyper-text links to the formatted document.

Resolving logical citations This often involves referencing external documents; it is related to the production of document bibliographies from information in bibliographic databases.

Structural summaries These are the logical part of the tables of various types of contents but not the page numbers; for a typical hyper-text document this would be the complete table.

3.3 Formatting-related transformations

Some transformations, although they are applied to the logical form of the text, either require information from the formatted form of the document or are naturally part of the formatting of the document.

3.3.1 Local transformations

The major example of a local transformation that is closely related to the visual formatting of text is the process of forming ligatures and other alternate visual forms such as the normal and final forms of the Greek letter sigma.

Here the word 'ligature' is used for all the large range of cases in which the formatted form of a word is not in direct correspondence with the string of characters forming the logical word. In all such cases, at some stage in the processing of the document, the logical form of the word has to be transformed. Following Haralambous ([Har95]), ligatures are here classified as (in this generalised sense) as either linguistic, contextual or aesthetic. All these types are dependent on the language in use, but in somewhat different ways.

The 'linguistic' ligatures are so much part of the logical language that, for example, they have a distinct place in the language's logical alphabet. There use is not context-dependent, indeed there is no discretion as to whether one is used; thus they are not our concern here.[2]

By 'contextual' ligature, Haralambous refers only to the logical context, e.g. the many 'obligatory ligatures' of Arabic typesetting. Since these do not depend on the font used or on other visual input, it is possible to introduce these by transforming the internal representation. However, if (as seems likely) the alternate forms cannot be used by the text applications such as spell-checkers, then their introduction should be delayed until the last stage of logical processing; thus they will effectively become part of the process of transformation to the visual representation. Haralambous' work [Har95, HP94] contains further discussion of the use of such transformations for a variety of languages and writing systems.

The 'aesthetic' ligatures usually depend on the font being used and may also depend on other aspects of the visual representation of the document. Their use therefore must be finally decided as part of the formatting of the document. However, the formatter will need to be passed information that determines whether certain such ligatures should be used; for example, typographic conventions for Turkish and Portuguese preclude the use of certain ligatures commonly present in Latin fonts.

A powerful example of the use of such aesthetic ligatures is provided by calligraphic fonts, e.g. Champion [Bol95] and Poetica [HP94]. Such fonts show clearly that it is not feasible for applications like spell-checking to act on the formatted text of a document.

The papers of Haralambous [Har95, HP94] and André [And95] contain further pertinent discussion of many types of alternate forms and their use in the typographic traditions of various languages.

[2] If one of these fails to appear in the input text then it should be produced as part of the translation from the exchange representation of the document so that the ligature is used in the input to hyphenation or spell-checking applications.

3.3.2 Global transformations

Examples of these include: resolution of cross-references to visual elements, e.g. page numbers, in structural summaries, cross-references and indexes. Typically, this type of transformation has two parts: a purely logical part and a formatting dependent part; a well-known example is a table of contents that in a traditionally printed document has the following parts.

- Formatting-independent structure, i.e. the logical hierarchy of sections, subsections, appendixes etc;
- Cross references to page numbers: the actual number that should be inserted depends on the formatting of the document.

4 Textual data

This section considers in more detail what exactly is being transformed by the text applications and processes described in Section 3. Thus it describes, working bottom-up, the form of the textual data that is required as input and output by such processes.

4.1 Characters

Characters are the atoms of any logical representation of the text itself. To support processing by text applications they need to be divided into categories, called 'types' in the Unicode documentation, such as letters, punctuation, digits, symbols. These categories can be used to help in distinguishing linguistic elements such as words, sentences, clauses and phrases when these are not explicitly marked-up. The details of what types are needed depends on the requirements of different languages and writing systems.

4.1.1 Relations between characters

There are various relationships, often described as 'tables', used by applications; these also often depend on the language in use.

Upper/Lower case This relationship is used specifically in the Roman, Cyrillic and Greek alphabets (and their derivatives), but similar considerations may apply to text using other scripts.

If the only need for automatically changing case were to satisfy the design and editorial specifications imposed on the formatting of the document then this could be implemented as a font axis, e.g., typesetting a running heading in uppercase could be implemented by selecting a font that contains the corresponding uppercase glyph in each of the lowercase slots.

However, transformations such as adding hyphenation points, or spell-checking, can reduce their search space drastically by first lower-casing all characters; thus the upper/lower

case correspondences need to be accessible by text applications. Further practical complications related to upper/lower-casing and the needs of various languages have been described by Braams, Haralambous and Plaice [HPB95].

Open/close pairs For some applications it is useful to classify certain punctuation characters as paired parenthesis-markers, e.g. brackets, quotation marks; an obvious example is a sub-system which checks that such pairs are properly balanced. These pairings, particularly for quotation marks, are also language-dependent.

Kerning tables Although it is often closely linked to ligatures, the kerning of glyphs is, by contrast, one process that does not depend in any sense on the language in use. It depends on the details of the glyphs used in the visual representation and not on the presence of logical structures such as hyphenation points or sub-word divisions, i.e. such structures should be transparent to the kerning process. Kerning tables must therefore be attached to particular fonts.

4.2 Logical structure of text

For reasons of encoding efficiency, textual data is normally only sparsely marked up, if at all. The lowest level of structure that is explicitly encoded in normal text is often the paragraph but it is clear that text has a lot of structure below this level. Here is a brief description of some of this structure that needs to be handled by common text applications.

4.2.1 Sub-words

As was shown in Section 3, this is an important part of the structure of text and it is used by many applications (indeed, it has far wider use than the more commonly provided hyphenation points). Like hyphenation points, it is not normally marked-up explicitly in the text; nor can it be deduced directly, even given a classification of characters into types (see above). Therefore an application similar to a hyphenation application needs to be used to find the sub-word division of each word. Sub-word information can be useful for full-text searches and it is essential for correct formatting, e.g. to suppress ligatures such as 'ff' being used inappropriately when the two 'f's are in distinct sub-words.

4.2.2 Words

There is a clear need to identify strings of characters that form 'words'. A good functional definition of a word in this context is as 'a character string that is the input for a hyphenation algorithm'; most spell-checkers will recognise the same collection of 'words' as their input.

For many texts the 'words' can be found automatically, given sufficient information about 'types' of characters. However, in some cases it may be necessary to explicitly delimit a word.

4.2.3 Sentences, Clauses & Phrases

These intermediate-level text structures need to be identified in order for use by applications such as grammar-checkers, context-dependent spell-checkers and sophisticated full-text searchers. Although typical logically marked-up documents explicitly identify some of these structures (e.g. quotes), most are only identified implicitly, and hence by chance rather than design.

4.3 Language

Most of the transformations, relationships and structures discussed in this paper would be informally described as being dependent on the language in which the text is written.

Although this paper will not attempt to give a formal definition of the concept 'language', it is nevertheless a concept central to almost all the text applications considered above. Fortunately, for our purposes it is simply a label; its importance is such that each bit of text must have a unique 'language-label' attached to it. This 'language-label' can be (and usually is) used by each text application in order to define or indicate what transformations are appropriate for that part of the text. It is also used by the formatter since it influences many aspects of typesetting.

The following algorithms and applications, amongst others, need to use information obtained by first interrogating this 'language-label':

- finding sub-words;
- the hyphenation rules;
- the dictionaries and heuristics used by the spell-checker;
- grammar- and style-checking;
- the generated text used in various places: for example, the word 'Chapitre' in the title **Chapitre 10 Quelle langue?**;
- which aesthetic ligatures should be used when the necessary glyphs are available in the font being used;
- the use of alternate forms (ligatures) that are dependent on the logical context (but independent of the font);
- the collating sequence used and other aspects of sorting for ordered lists such as indexes and bibliographies.

Some applications also use the language-label to specify exactly what character-codes to expect within text with that language-label; for example, the system may wish to issue an error-message or a warning when an unexpected character appears.

As many of the transformations above suggest, this concept of a single non-structured language-label is not completely congruent with more general uses of the word 'language'. For example, to encode the use of different hyphenation rules for the same written language would require different language-labels; more obviously, if the same spoken language can

be written with distinct scripts, each of these will require a separate language-label. It is, however, at the correct level of abstraction for an application-independent representation and it provides an efficient encoding of information that is required by a large range of text applications.

5 The representation

Much of the required representation can be provided by use of Unicode. Those features that are not directly supported by Unicode, according to the published standard [UC95], are listed here with some comments.

First are those that need only specify that an existing or a new character should be used for a particular purpose. Since the extra characters are very similar in purpose to the many non-printing characters in the current standard, they introduce no new principles and their treatment by conforming applications raises no new issues.

- A character to indicate a point of division of a word into sub-words (including prefixes and suffixes). It has been suggested ([HP95]) that code point 200B(ZERO WIDTH SPACE) should be used for this purpose but this conflicts with the Unicode documentation which states that this character indicates a space which is a possible break point and which can be used explicitly to denote an inter-word break. A better choice is 200C(ZERO WIDTH NON-JOINER).
- Explicit 'begin-word' and 'end-word' characters for use when automatic detection is not possible.
- A method of indicating a preference between different word-division (hyphenation) points within a word. The standard already contains a character (00AD) called SOFT HYPHEN.
- A method of indicating alternate forms where the choice depends on the visual context; in particular, spelling changes necessitated by line-breaks after a hyphen.

Note: these last two are categorised as simply needing new characters despite the fact that the Unicode documentation explicitly excludes line-breaking and hyphenation; this is because the standard nevertheless does include some closely related characters.

There is also a need for extensions to the part of the standard that classifies characters into types.

It is also necessary to be able to indicate correspondences between subsets of the characters, such as the upper/lower case correspondence and the open/close pairs for parenthesis.

Finally, as explained in Section 4.3, there is the one essential addition to the data structure that is essential to any but the most primitive uses of text: the need to specify a 'language-label'. This will necessarily be an extension to the Unicode standard if not a higher-level protocol; it will add to the rich variety of characters afforded by Unicode the extra information that, in multi-lingual documents, is necessary and sufficient for a large range of essential transformations of text.

Although the Unicode standard does not directly support anything like our concept of 'language-label', it does provide very sophisticated support for one very important, but limited, aspect of multi-lingual document processing: bidirectionality (sic). Thus the extension to other equally important language-related requirements of document processing would seem to be both natural and within the spirit of this standard.

To summarise, these enhancements and additions to the Unicode standard will provide a suitably rich and consistent interface to many applications that act at the the logical level on textual data.

References

[And95] Jacques André. Ligatures et informatique. *Cahiers GUTenberg*, 22:61–86, 1995.

[AG95] Jacques André and Michel Goosens. Codage des caractères et multi-linguisme: de l'Ascii à Unicode et ISO/IEC-10646. *Cahiers GUTenberg*, 20:1–53, 1995.

[Bol95] François Boltana. Ligatures & calligraphie assistée par ordinateur. *Cahiers GUTenberg*, 22:107–124, 1995.

[Har95] Yannis Haralambous. Tour du monde des ligatures. *Cahiers GUTenberg*, 22:87–100, 1995.

[HP94] Yannis Haralambous and John Plaice. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15(3):344–352, 1994.

[HP95] Yannis Haralambous and John Plaice. Omega, une extension de TEX incluant Unicode et des filtres de type Lex. *Cahiers GUTenberg*, 20:55–79, 1995.

[HPB95] Yannis Haralambous and John Plaice and Johannes Braams. Never again active characters! Ω-Babel. *TUGboat*, 16(4):418–427, 1995.

[UC95] The Unicode Consortium. *The Unicode Standard*. Addison-Wesley, 1995.