An extensible approach to high-quality multilingual typesetting

John Plaice[1], Yannis Haralambous[2], and Chris Rowley[3]

[1] School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
plaice@cse.unsw.edu.au
[2] Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
BP832, F-29285 Brest Cedex, France
Yannis.Haralambous@enst-bretagne.fr
[3] Faculty of Mathematics and Computing
The Open University, UK
Milton Keynes MK7 6AA, United Kingdom
C.A.Rowley@open.ac.uk

Abstract. We propose to create a new model for multilingual computerized typesetting, in which each of language, script, font and character is treated as a multidimensional entity, and all combine to form a multidimensional context. Typesetting then becomes a four-stage process of preparing the input stream for typesetting, segmenting the stream into clusters or words, typesetting these clusters, and then recombining them. Each of the stages, including their respective algorithms, is dependent on the multidimensional context. This approach will support quality typesetting for a number of modern and ancient scripts.

1 Introduction

We present in this paper a new approach to computer typesetting, "The production of printed matter by computer, usually by producing a master copy for offset reproduction" [2]. The key innovation is to consider each of language, script, font, and character as multidimensional entities, as opposed to the current view, reiterated at length in the Unicode standard [12], that they are discrete and unchanging. As a result, typesetting will be undertaken in a *multidimensional context* — formally a point in a multidimensional space — that summarizes the current linguistic and cultural space. This point of view, consistent with the intensional programming approach, explained below, will allow for much greater variation in the behavior of the typesetting engine. In fact, this approach will allow the typesetter to be integrated with more general text processing tools, such as spell-checkers, style-checkers, content-checkers, transliterators, or translators.

2 Background

2.1 Computer Typesetting

The first steps in computer typesetting took place in the 1950s, but it was not until 1982, when Donald Knuth introduced TEX [4], that it became possible to use computer software for high-quality typesetting of English and mathematics, as in *The Art of Computer Programming* series.

TEX's low-level typesetter maps *characters* in the input file into *glyphs* in the current font, and places the glyphs side-by-side, on the *baseline*; a font-specific finite-state automaton inserts *ligatures* and *kerning*; a font-specific parameter defines the stretchable interword space. Each glyph has width, height, and depth, and these are used for higher-level layout.

The Ω system [7], developed by the first two authors, is an extension of the TEX model supporting multilingual typesetting. It has been used for typesetting languages in the following scripts: Latin (including Gothic and Gaelic), Greek, Cyrillic, Armenian, Georgian, Arabic, Hebrew, Syriac, Tifinagh, Japanese, Thai, Khmer, Devanagari (for Hindi, Sanskrit), Malayalam and Tamil.

In Ω, the input character stream is processed by a series of filters, each reading from standard input and writing to standard output. Once all of the filters are applied, the stream is passed to the TEX low-level typesetter. Filters have been written for character set conversion, transliteration, morphological analysis, spell-checking, and contextual analysis, and two-dimensional layout.

There are two current limitations to the use of Ω. First, because Ω is so versatile, it is difficult to define a higher-level interface, that can be used by a novice. Second, the output from applying several filters is often human-unreadable; when this output is placed in a PDF document, the visual layout is correct but no searching is possible, despite that fact that the PDF standard was designed precisely for this dual-purpose.

2.2 Intensional Programming

Intensional programming [9] is an approach to computing that supposes that there is a multidimensional context, and that all programs are capable of adapting themselves to this context. The context can be tested or manipulated, dimension by dimension. The context is pervasive, and can simultaneously affect the behavior of a program at the lowest, highest and middle layers.

Intensional versioning [11] is the dual of intensional programming, where the definitions themselves vary with the context. Any entity can be defined in multiple versions, and when that entity is needed, the most relevant version, with respect to the current context, is chosen. This is called the *variant substructure principle*.

The ISE programming language [10] combines both intensional programming and versioning with the features of the procedural scripting language Perl, and it has greatly facilitated the creation of multidimensional Web pages. Similar experimental work has been undertaken with C, C++, Java, and Eiffel. And,

when combined with a context server, it becomes possible for several documents or programs to be immersed in the same context.

3 Significance

The significance of high-quality computerized multilingual typesetting cannot be overestimated. We know from Marshall McLuhan's work [5] just how important was the introduction of metal type to European society. Typesetting was, in some sense, the first industrial process, upon which all others were based. It was also the process that enabled the others, since it allowed knowledge to spread rapidly across Europe. It also facilitated the rise of national vernaculars and the subsequent creation of nation-states.

Today, with the development of the Internet and even more so the Web, something different is occurring. We now have *access* to online documents in hundreds of languages, using a multitude of scripts. At the same time, grandiose endeavors such as the Million Book Project [6] (scanning of about 4% of the books ever written) are being undertaken. Bit by bit, the world's collected writings are being made available, to everyone. And, with miniaturization of storage, these writings will be available not just online, but on our portable devices.

However, making these works available is not sufficient. They still need to be printed, whether it be on a screen, in a bound paper volume, or on some future substrate. But we are not yet at a point where we can automatically reproduce the quality of books typeset in the nineteenth century, particularly for the non-Latin scripts. In fact, the problem is harder, because we now need real-time printing of documents from the Web.

In India, this problem is of utmost importance. India has 2 national languages (Hindi and English), 1 recognized mother language (Sanskrit), and 14 official languages, each with its own script. In addition, there are approximately 200 minority languages. Clearly, a general approach to multilingual typesetting is necessary, that promises ease of use with high-quality.

4 Approach

In this section, we define what a context is, then show how it is to be used for computerized typesetting. We then explain how these ideas will be validated with real multilingual texts.

Contexts are simply dictionaries (pairs of attribute-values), where some of the values are themselves dictionaries. For example, a context describing Australian English could be `script:Latin+lang:(English+dialect:Australian)`. We refer to `script` and `lang` as *dimensions*, and to `lang:dialect` as a *nested dimension*. There is a refinement relation $\sqsubseteq$ on the contexts that forms a partial order.

At all times that the system is running, there is a current context. This context can be modified in a *relative* manner, dimension by dimension, or in an *absolute* manner. If the context is as above, then `vmod[lang:dialect:Canadian]`

would change it to `script:Latin+lang:(English+dialect:Canadian)`. On the other hand, the expression `vset[script:Hanzi+lang:Chinese]` would completely replace the current context with a Chinese one.

The overall algorithm is as follows. The input character stream will pass through four separate *phases*: *preparation*, *segmentation*, *micro-typesetting* and *recombination*.

The preparation phase is similar to the current situation in the Ω system. At all times, there is an active Ω Translation Processing List (ΩTP-list), which consists of a sequence of individual Ω Translation Processes (ΩTP's), which are filters reading from standard input to standard output. What is new is that the whole process will become context-dependent. First, the most relevant ΩTP-list, with respect to the context and using the refinement relation over contexts, is the one that is active. Second, once chosen, it can test the current context, and adapt its behavior, by selectively turning on or off, or even replacing, individual ΩTP's.

The preparation phase should work mainly at the *character*, i.e. the *information exchange* level. It is designed so that additional markup may be inserted into the character stream, so that the following stages may have more detailed information, allowing for better typography.

The segmentation phase splits the stream of characters into clusters of characters; typically, segmentation is used for word detection. In English, this is a trivial problem, and segmentation just means recognizing the blank character, Unicode `U+020`. On the other hand, in Thai, where there is no word-delimiter in the character stream (blanks are traditionally only used as sentence-delimiters), it is impossible to do any form of automatic processing unless a sophisticated morphological analyzer is being used to calculate word and syllable boundaries. The choice of segmenter is once again context-dependent.

During the micro-typesetting phase, a *μ-engine* processes a cluster, taking into account the current context, including language and font information, and produces typeset output. If hyphenation or some other form of cluster-breaking is allowed for the current language-script combination, then there will be multiple possible typeset results, and all of these possibilities must be handled. When dealing with complex scripts or with fonts allowing great versatility (as with Adobe Type 3 fonts), numerous different *μ-engines* will need to be written, and they will be selected and their behavior will be finetuned according to the context.

The final phase, before calling higher-level processes such as a paragrapher, is the recombination phase. Here, the typeset clusters are placed next to each other. For simple text, such as the English in this proposal, this simply means placing a fixed stretchable space between typeset words. In situations such as Thai and some styles of Arabic typesetting, kerning would take place between words. Once again, the recombiner's behavior is context-dependent.

Given that the choice of segmenter, the μ-engine and recombiner are all context-dependent, and that the actions of each of these, once they are chosen, also depends on the context, this new model of typesetting engine is *much* more

powerful than anything previously proposed or implemented. We intend to test it and to validate on the following scripts:

- *Latin, Greek and Cyrillic, IPA*: left-to-right, discrete glyphs, numerous diacritics, stacked vertically, above or below the base letters, widespread hyphenation;
- *Hebrew*: right-to-left, discrete glyphs, optional use of diacritics (vowels and breathing marks), which are stacked horizontally below base letter;
- *Arabic, Naskh style*: right-to-left, contiguous glyphs, contextually shaped, numerous ligatures, optional use of diacritics (vowels and breathing marks), placed in two-dimensionals, above and below;
- *Indic scripts*: left-to-right, two-dimensional layout of clusters, numerous ligatures, applied selectively according to linguistic and stylistic criteria;
- *Chinese, Japanese*: vertical or left-to-right, often on fixed grid, with annotations to the right or above the main sequence of text, automatic word recognition needed for any form of analysis;
- *Egyptian hieroglyphics*: mixed left-to-right and right-to-left, two-dimensional layout.

Once the basic typesetting is validated, then further experiments, viewing language as a multidimensional entity, will be undertaken. Already with Omega, we have typeset Spanish with both the Hebrew and Latin scripts; Berber with the Tifinagh, Arabic and Latin scripts; Arabic with Arabic, Hebrew, Syriac, Latin and even *Arabized Latin* (Latin script with a few additional glyphs reminiscent of the Arabic script). The Arabic script can be rendered in Naskh or Nastaliq or many other styles. Japanese can be typeset with or without *furigana*, little annotations above the *kanji* (the Chinese characters) to facilitate pronunciation. The objective is to incorporate all of these problems, currently solved in an *ad hoc* manner, into the proposed framework; each time, the key is to correctly summarize the context.

5 Conclusions

If the model that we propose to develop is successful, then we will be able to produce, with relative ease, high-quality documents in many different languages and scripts. In particular, the third author, as one of the leaders of the LaTeX3 project, will develop a LaTeX interface to the new functionality.

Furthermore, this new approach will provide a simple high-level interface allowing the user to take advantage of new developments in font technologies. In particular, Adobe Type 3 fonts are designed so that glyphs can be generated differently upon each rendering (see [1] for a discussion of a number of effects). On another level, the OpenType standard [8], jointly developed by Adobe and Microsoft, allows for many different kinds of parameters — well beyond the basic three of width, height, and depth —, multiple baselines, and a much richer notion of ligature. The new typesetting engine will provide new capabilities, adaptable to new kinds of parameters, and increased control.

At another level, the existing Ω system has already influenced the specifications of XML [13] (how to deal with multiple character sets) and XSL [14] (the model for printing in multiple-directions). If the proposed research in typesetting is successful, additional contributions to XSL may take place, by providing natural specifications for low-level XSL formatting objects, currently missing from the standard.

Finally, this proposed model should be understood as the preparation for a much more ambitious project, that will deal not just with low-level typesetting but also general problems of document structuring and layout. Serious detailed discussion has already been initiated between the Ω and LaTeX3 projects.

References

1. Jacques André. *Création de fontes en typographie numérique* [Creating fonts for digital typography]. Documents d'habilitation, IRISA+IFSIC, Rennes, 1993.
2. Computer Typesetting. `http://www.xrefer.com/entry/441575`
3. Yannis Haralambous. Unicode et typographie : un amour impossible [Unicode and typography: an impossible couple]. *Documents numériques* 1:1, 2002.
4. Donald Knuth. *Computers and Typesetting.* 5 volumes, Addison-Wesley, 1986.
5. Marshall McLuhan. *The Gutenberg Galaxy: The Making of Typographic Man.* University of Toronto Press, 1962.
6. Million Book Project. `http://zeeb.library.cmu.edu/Libraries/LIT/Projects/1MBooks.html`
7. Omega Typesetting and Document Processing System. `http://omega.cse.unsw.edu.au`
8. OpenType. `http://www.opentype.org`
9. John Plaice and Joey Paquet. Introduction to intensional programming. In *Intensional Programming I*, World-Scientific, Singapore, 1996.
10. John Plaice, Paul Swoboda and Ammar Alammar. Building intensional communities using shared contexts. In *DCW 2000*, *LNCS* 1830:55–64, 2000.
11. John Plaice and William W. Wadge. A new approach to version control. *IEEE-TSE* 19(3):268–276, 1993.
12. Unicode Home Page. `http://www.unicode.org`
13. Extensible Markup Language (XML). `http://www.w3c.org/XML`
14. The Extensible Stylesheet Language (XSL). `http://www.w3c.org/Style/XSL`