Moving Ω to an Object-Oriented Platform

John Plaice, Yannis Haralambous, Paul Swoboda, Gabor Bella

Moving Ω to an Object-Oriented Platform

John Plaice[1], Yannis Haralambous[2], Paul Swoboda[1], and Gábor Bella[2]

[1] School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
{plaice,pswoboda}@cse.unsw.edu.au
[2] Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
{yannis.haralambous,gabor.bella}@enst-bretagne.fr

Abstract. The code for the Ω Typesetting System has been substantially reorganised. All fixed-size arrays implemented in Pascal Web have been replaced with interfaces to extensible C++ classes. The code for interaction with fonts and Ω Translation Processes (ΩTP's) has been completely rewritten and placed in C++ libraries, whose methods are called by the (now) context-dependent typesetting engine. The Pascal Web part of Ω no longer uses change files. The overall Ω architecture is now much cleaner than that of previous versions.
Using C++ has allowed the development of object-oriented interfaces without sacrificing efficiency. By subclassing or wrapping existing stream classes, character set conversion and ΩTP filter application have been simultaneously generalised and simplified. Subclassing techniques are currently being used for handling fonts encoded in different formats, with a specific focus on OpenType.

1 Introduction

In this article, we present the interim solution for the stabilisation of the existing Ω code base, with a view towards preparing for the design and implementation of a new system. We focus on the overall structure of the code as well as on specific issues pertaining to characters, fonts, ΩTP's and hyphenation.

Since the first paper on Ω was presented at the 1993 Aston TUG Conference, numerous experiments with Ω have been undertaken in the realm of multilingual typesetting and document processing. This overall work has given important insights into what a future document processing system, including high quality typesetting, should look like. We refer the reader to the 2003 TUG presentation [7], as well as to the position papers presented to the Kyoto Glyph and Typesetting Workshop [3, 6, 8]. Clearly, building an extensive new system will require substantial effort and time, both at the design and the implementation levels, and so it is a worthwhile task to build a production version of Ω that will be used while further research is undertaken.

The standard `web2c` infrastructure, which assumes that a binary is created from a single Pascal Web file and a single Pascal Web change file, is simply not well suited for the development of large scale software, of any genre. For this reason, we have eliminated the change files, and broken up the Pascal Web file into chapter-sized files. All fixed-size arrays have been reimplemented in C++ using the Standard Template Library. Characters are now 32 bits, using the `wchar_t` data type, and character set conversion is done automatically using the routines available in the `iconv` library. The entire Pascal Web code for fonts and ΩTP's, including that of Donald Knuth, has been completely rewritten in C++ and placed in libraries. Clean interfaces have been devised for the use of this code from the remaining Pascal code.

2 Problems with Pascal Web

When we examine the difficulties in creating Ω as a derivation of `tex.web`, we should understand that there is no single source for these difficulties.

Pascal was designed so that a single-pass compiler could transform a monolithic program into a running executable. Therefore, all data types must be declared before global variables; in turn, all variables must be declared before subroutines, and the main body of code must follow all declarations. This choice sacrificed ease of programming for ease of compiler development; the resulting constraints can be felt by anyone who has tried to maintain the TEX engine.

Pascal Web attempts to alleviate this draconian language vision by allowing the arbitrary use within code blocks – called *modules* – of pointers to other modules, with a call-by-name semantics. The result is a programming environment in which the arbitrary use of GOTOs throughout the code is encouraged, more than ten years after Dijkstra's famous paper. Knuth had responded correctly to Dijkstra's paper, stating that the reasonable use of GOTOs simplifies code. However, the arbitrary use of GOTOs across a program, implicit in the Pascal Web methodology, restricts code scalability. Knuth himself once stated that one of the reasons for stopping work on TEX was his fear of breaking it.

For a skilled, attentive programmer such as Knuth, developing a piece of code that is not going to evolve, it is possible to write working code in Pascal Web, *up to a certain level of complexity*. However, for a program that is to evolve significantly, this approach is simply not tenable, because the monolithic Pascal vision is inherited in Pascal Web's change file mechanism. Modifications to TEX are supposed to be undertaken solely using change files; the problem with this approach is that the vision of the code maintainer is that they are modifying functions, procedures, and so on. However, the *real* structure of a Pascal Web program is the interaction between the Pascal Web *modules*, not the functions and procedures that they define. Hence maintaining a Pascal Web program is a very slow process. Back in 1993, when the first Ω work was being undertaken, "slow" did not just mean slow in design and programming, but also in compilation: the slightest modification required a 48-minute recompilation.

The size limitations created by `tex.web`'s compile-time fixed-size arrays are obvious and well known. This issue was addressed publicly by Ken Thompson in the early 1980s, and both the existing Ω and the `web2c` distribution have substantially increased the sizes. However, these arrays raise other problems. The `eqtb`, `str_pool`, `font_info` and `mem` arrays all have documented programming interfaces. However, whenever these interfaces are insufficient, the TEX code simply makes direct accesses into the arrays. Hence any attempt to significantly modify these basic data structures requires the modification of the *entire* TEX engine, and not simply the implementations of the structural interfaces.

In addition, the single input `buffer` for all active files of `tex.web` turns out to be truly problematic for implementing ΩTP's. Since an ΩTP can read in an arbitrary amount of text before processing it, a new input buffer had to be introduced to do this collection. The resulting code is anything but elegant, and could certainly be made more efficient.

Finally, problems arise from the `web2c` implementation of Pascal Web. Many of the routines written in C to support the `web2c` infrastructure make the implicit assumption that all characters are 8 bits, making it difficult to generalise to Unicode (currently 21 bits), even though C itself has a datatype called `wchar_t`.

3 Suitability of C++

The advantages of the use of C++ as an implementation language for stream-oriented typesetting, over the Pascal Web architecture, are manifold. The chief reason for this is that the rich set of tools and methodologies that have evolved in the twenty-five years since the introduction of TEX includes developments not only in programming languages and environments, but in operating systems, file structure, multiprocessing, and in the introduction of whole new paradigms, including object-oriented software and generic programming.

C++ is the *de facto* standard for object-oriented systems development, with its capability to provide low-level C-style access to data structures and system resources (and, in the case of Unix-like systems, direct access to the kernel system call API), for the sake of efficiency.

In addition, the C++ Standard Template Library (STL) offers built-in support for arbitrary generic data structures and algorithms, including extensible, random-access arrays. It would be foolish to ignore such power when it is so readily available.

Since C++ is fully compatible with C, one can still take advantage of many existing libraries associated with TEX, such as Karl Berry's `kpathsea` file searching library, and the `iconv` library character-set conversion between Unicode and any other imaginably-used character set.

The abilities to use well-known design patterns for generic algorithm support (plug-in paragraphers, generic stream manipulation), as well as generic representation of typesetting data itself, add a wealth of possibilities to future, open typesetting implementations.

4 Organisation of the Ω Code Base

Obviously, we are moving on. Our objective is to include the existing Ω functionality, to stretch it where appropriate, leaving clean interfaces so that, if others wish to modify the code base, they can do so. Our current objective is not to rewrite TEX, but its underlying infrastructure.

4.1 Reorganising the Pascal Web Code

The `tex.web` file has been split into 55 files called `01.web` to `55.web`. The `tex.ch` file has been converted into 55 files, `01.ch` to `55.ch`. Data structure by data structure – specifically the large fixed-size arrays – we have combed the code, throwing out the definitions of the data structures and replacing their uses with Pascal procedure calls which, once passed through the `web2c` processor, become C++ method calls. In the process, most of the code in the change files ends up either being unnecessary, or directly integrated in the corresponding `.web` files.

4.2 The External Interface with Ω

We envisage that Ω will be used in a number of different situations, and not simply as a batch standalone program. To facilitate this migration, we have encapsulated the interface to the external world into a single class. This interface handles the interpretation of the command line, as well as the setup for the file searching routines, such as are available in the `kpathsea` library. Changing this class will allow the development of an Ω typesetting server, which could be used by many different desktop applications.

4.3 Characters, Strings and Files

The other interface to the outside world is through the data passed to Ω itself. This data is in the form of text files, whose characters are encoded in a multitude of different character encodings.

For characters, TEX has two types, `ASCII_code` and `text_char`, the respective internal and external representations of 8-bit characters. The new Ω uses the standard C/C++ data type, `wchar_t`. On most implementations, including GNU C++, `wchar_t` is a 32-bit signed integer, where the values `0x0` to `0x7fffffff` are used to encode characters, and the value `0xffffffff` (-1) is used to encode `EOF`. Pascal Web strings are converted by the `tangle` program into `str_number`, where values `0` to `255` are reserved for the 256 8-bit characters. We have modified `tangle` so that the strings are numbered `-256` downwards, rather than `256` upwards. Hence, `str_number` and `wchar_t` are both 32-bit signed integers.

When dealing with files, there are two separate issues, the file names, and the file content. Internally, all characters are 4-byte integers, but on most systems, file names are stored using 8-bit encodings, specified according to the user's locale. Hence, character-set conversion is now built into the file-opening mechanisms, be they for reading or writing.

The actual content of the files may come from anywhere in the world and a single file system may include files encoded with many different encoding schemes. We provide the means for opening a file with a specified encoding, as well as opening a file with automatic character encoding detection, using a one-line header at the beginning of the file. The actual character set conversion is done using the `iconv` library. As a result of these choices, the vast majority of the Ω code can simply assume that characters are 4-byte Unicode characters.

In addition to the data files, the following information must be passed through a character encoding converter: command line input, file names, terminal input, terminal output, log file output, generated intermediate files, and `\special` output to the `.dvi` file.

4.4 The Fixed-Size Arrays

The core of the the new Ω implementation is the replacement of the large fixed-size arrays, which are quickly summarized in the table below:

`str_pool`	string pool
`buffer`	input buffer
`eqtb`, etc.	table of equivalents
`font_info`, etc.	font tables
`mem`	dynamically allocated nodes
`trie`, etc.	hyphenation tables

For the cumulative data arrays, such as the string pool, we have created a new class, `Collection`, subclass of `vector`, that can be `dump`'ed to and `undump`'ed from the format file.

Currently no work has been done with the dynamically allocated nodes and the hyphenation tables. Replacing the `mem` array with any significantly different structure for the nodes would effectively mean rewriting all of TEX, which is not our current goal.

4.5 The String Pool

The TEX implementation used two arrays: `str_pool` contained all of the strings, concatenated, while `str_start` held indices into `str_pool` indicating the beginning of each string. This has all been replaced with a `Collection<wstring*>`, where `wstring` is the STL string for 4-byte characters. As a result, we can directly take advantage of the hashing facilities provided in the STL. Note that the `omega.pool` file generated by `tangle` has been transformed into a C++ file.

4.6 The Input Buffer

The TEX implementation used a single array `buffer`, holding all the active lines, concatenated. This has now been broken up into a `Collection` of string streams. This setup simplifies the programming of ΩTPs, which must add to the input buffer while a line is being read.

4.7 The Table of Equivalents

The table of equivalents holds the values for the registers, the definitions for macros, and the values for other forms of globally accessible data. The TEX implementation used three arrays: `eqtb` held all of the potential equivalent entries, `hash` mapped string numbers to equivalent entries, and `hash_used` was an auxiliary Boolean table supporting the hashing.

The table has now been broken into several tables `map<unsigned,Entry*>` (for characters or register numbers) or `map<wstring,Entry*>` (for macro definitions), where `Entry` is some kind of value. Support is provided for characters up to `0x7fffffff`, and the STL hashing capabilities are used. This infrastructure has been built using the `intense` library [9], thereby allowing each `Entry` to be *versioned*. allowing different definitions of a macro for different contexts.

4.8 Fonts and ΩTPs

In terms of numbers of lines written, most of the new code in Ω is for handling fonts and ΩTPs. However, because we are using standard OO technology, it is also the most straightforward.

The original TEX and Ω code for fonts was concerned mostly with bit packing of fields in the `.tfm` and `.ofm` files, and unpacking this information inside the typesetting engine whenever necessary. This approach was appropriate when space was at a premium, but it created very convoluted code. By completely separating the font representations in memory and on disk, we have been able to provide a very simple OO interface in the character-level typesetter of the Ω engine, greatly simplifying the code for ligatures and kerning inside the typesetter, as well as for the font conversion utilities.

Similarly, for the ΩTPs, filters can be implemented as function objects over streams using iterators, tremendously simplifying the code base.

5 Supporting OpenType

Since we are using a programming language supporting type hierarchies, it is possible to support many different kinds of font formats. In this section, we consider different options for supporting OpenType, the current de facto standard.

The OpenType font format has been officially available since 1997. Unlike its predecessors, TrueType and PostScript Type 1 and 2, it facilitates handling of LGC (Latin-Greek-Cyrillic) scripts and also provides essential features for proper typesetting of non-LGC ones. Competing formats with similar capabilities (Apple GX/AAT and Graphite) do exist, but the marketing forces are not as strong.

At the EuroTEX conference in the summer of 2003, we presented our first steps towards an OpenType-enabled Ω system. At the time, OpenType and Ω were just flirting, but since last year their relationship has become more and more serious. In other words, what began simply as the adaptation of Ω to OpenType fonts has now become a larger-scale project: the authors are planning

to restructure Ω's font system and make OpenType a base font format. As it will be shown, full OpenType compatibility requires serious changes inside both Ω and `odvips`. The other goal of the project is to simplify the whole font interface, eliminating the need for separate metric files, virtual fonts and the like (while the old system will of course continue to be supported).

Such a project, however, will certainly need some time to finish. Fortunately, the work done until now already provides users with the possibility to typeset using OpenType fonts, even if only a limited number of features are supported. It will be shown below that further development is not possible without major restructuring of the Ω system. Nevertheless, the present intermediate solution is in fact one of the three that we will retain.

Before getting to the discussion of possible solutions, let us briefly present the most important aspects of OpenType and their implications for Ω development.

5.1 OpenType vs. Omega

The key features of the OpenType format are summarised in the list below. As each one of these features raises a particular compatibility issue with Ω, they will all be elaborated below.

1. Font and glyph metric information;
2. Type 2 or TrueType glyph outlines (and hints or instructions);
3. Advanced typographic features (mainly GSUB and GPOS);
4. Clear distinction between character and glyph encodings;
5. Pre-typesetting requirements;
6. Extensible tabular file format.

Font and Glyph Metrics. OpenType provides extensive metric information dispersed among various tables (`post`, `kern`, `hmtx`, `hdmx`, `OS/2`, `VORG`, etc.), both for horizontal and vertical typesetting. Although in most cases Ω's and OpenType's metrics are interconvertible a few but important exceptions do exist (e.g., height/depth) where conversion is not straightforward. See [1, 4].

Glyph Outlines, Hints and Instructions. Since the OpenType format itself is generally not understood by PostScript printers, a conversion to more common formats like Type 1 or Type 42 is necessary. As explained in [1], to speed up this conversion process, we create Type 1 charstring collections using our own PFC tables which are used by `odvips` to create small, subsetted Type 1 fonts (a.k.a. *minifonts*) on the fly. This solution, on the other hand, does not preserve hints nor instructions, at least not in the present implementation. We are therefore planning to also provide Type 42 support for TrueType-flavoured OpenType. This solution would allow us to preserve instructions, at the expense of subsetting and compatibility.

Advanced Typographic Features. These are perhaps the most important aspect of OpenType. Its GSUB (glyph substitution) and GPOS (glyph positioning) tables are essential for typesetting lots of non-LGC scripts. In Ω, the equivalent of GSUB features are the ΩTP's: they can do everything GSUB features can, including contextual operations. Glyph positioning is a different issue: since the ΩTPs are designed for text rearrangement (substitutions, reordering etc.), they are not suitable for doing glyph placement as easily. Context-dependent typesetting *microengines* for character-level typesetting have been proposed for Ω to provide modular, script- and language-specific positioning methods, along the lines of ΩTP files; however, they have yet to be implemented. The positioning features in OpenType GPOS tables are in fact the specifications for microengines.

Character and Glyph Encodings. The above discussion of advanced typographic features brings us to a related issue: the fundamental difference between Ω's and OpenType's way of describing them. Although both Ω and OpenType are fully Unicode compatible, OpenType's GSUB and GPOS features are based on strings made of glyph ID's and not of Unicode characters. As for Ω and some of its ΩTP's, tasks such as contextual analysis or hyphenation are performed on character sequences and the passage from characters to "real" glyph ID's happens only when `odvips` replaces virtual fonts by real ones. To convert a glyph-based OpenType feature into a character-based ΩTP would require Ω to offer means of specifying new "characters" (the glyph ID's) that do not correspond to any Unicode position. The conversion itself would not be difficult since Ω's possible character space is much larger than Unicode's. This, however, would lead us to glyph ID-based, hence font-specific, ΩTP's and hyphenation, which is not a lovely prospect, to say the least. To solve this problem, it will certainly be necessary to keep both character and glyph information of the input text in parallel during the whole typesetting and layout process. This dual representation of text is also crucial for the searchability and modifiability of the output (PDF, PS, SVG or any other) document.

Pre-typesetting Requirements. OpenType relies on input text reordering methods for its contextual lookups to work correctly. If Ω is to use the same lookups, these reordering methods must also be implemented, either by ΩTP's or by an external library.

Extensibility. Finally, the OpenType format has the important feature of being extensible: due to its tabular structure, new tables can be added into the font file, containing, for example, data needed by Ω with no OpenType-equivalents (like metrics or PFC charstrings, see below). However, it is necessary that the given font's license allow additions.

5.2 Solutions

From the above discussion it should now be clear that complete and robust OpenType support is not a simple patch to Ω and `odvips`. Three solutions are proposed below, in order of increasing difficulty and of our working plan.

1. Convert OpenType fonts into existing Ω font metrics and ΩTP's;
2. Provide built-in support within Ω for a fixed but extensive set of OpenType features and read data directly from the OpenType font file;
3. Finally, provide extensible means for using the full power of OpenType fonts.

The Current Solution. This, described in detail in the EuroTEX article [1], corresponds to the first solution. Here we give a short summary.

The initial solution was based on the approach that OpenType fonts should be converted to Ω's own formats, i.e., `.ofm` (metrics), `.ovf` (virtual fonts) and ΩTP. Anish Mehta wrote several Python scripts to generate these files, of which the most interesting is perhaps the one that converts the whole OpenType GSUB table into ΩTP's. Type 2 and TrueType outlines themselves are converted into the Type 1-based PFC format and are subsetted on the fly by a modified `odvips`.

In summary, the present solution is a *working one*. Admittedly far from being complete (GPOS support is missing, among others), it is intended to provide Ω users with the possibility to typeset using OpenType fonts, including even some of its advanced features, while further development is being done.

Future Solutions. The second and third solutions mentioned above require that the Ω engine be capable of directly reading OpenType fonts, which can be done using a public library such as `freetype` or Kenichi Handa's `libotf`. This would also eliminate the need to create `.ofm` and `.ovf` files.

Providing built-in support for a fixed set of features corresponds to the aforementioned microtypesetting engines. For a given set of features, a new engine can be written. This approach can be taken using standard OO techniques.

A more general approach requires the ability to reach into an OpenType font, reading tables that were not known when the Ω engine was written. For this to work requires some kind of programming language to be able to manipulate these new tables. A simple such language is Handa's Font Layout Tables [2].

It should be clear that these solutions are not mutually exclusive and that backwards compatibility with the classic font system will be maintained.

6 Conclusions

At the time we are writing, this work is not completely finished. Nevertheless, it is well advanced: the infrastructure is substantially cleaned up, and is extensible, with clear API's. Detailed documentation will be forthcoming on the Ω website.

If we view things in the longer term, we are clearly moving forward with two related goals, the stabilisation of existing Ω infrastructure, and abandonment of the TEX infrastructure for the design and implementation of a next-generation open typesetting suite.

Such a suite should be a generic framework with an efficient C++ core, that is universally extensible through a number of well-known scripting interfaces, for example, Perl, Python, and Guile. Implementation of libraries similar to the popular LATEX suite could then be done directly in C++, on top of the core API, or as a linked-in C++ stream filter.

References

1. Gábor Bella and Anish Mehta. Adapting Ω to OpenType Fonts. *TUGboat*, 2004. In press.
2. Kenichi Handa, Mikiko Nishikimi, Naoto Takahashi and Satoru Tomura. FLT: Font Layout Table. Kyōto University 21st Century COE Program, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/052-handa.pdf`
3. Tereza Haralambous and Yannis Haralambous. Characters, Glyphs and Beyond. Kyōto University 21st Century COE Program, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/017-tereza.pdf`
4. Yannis Haralambous and John Plaice. Omega and OpenType Fonts. Kyōto University 21st Century COE Program, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/067-yannis.pdf`
5. The OpenType Specification v1.4. `http://www.microsoft.com/typography/otspec/default.htm`
6. John Plaice and Chris Rowley. Characters are not simply names, nor documents trees. Kyōto University 21st Century COE Program, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/009-plaice.pdf`
7. John Plaice, Paul Swoboda, Yannis Haralambous and Chris Rowley. A multidimensional approach to typesetting. *TUGboat*, 2003. In press.
8. Chris Rowley and John Plaice. New directions in document formatting: What is text? Kyōto University 21st Century COE Program, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/001-rowley.pdf`
9. Paul Swoboda and John Plaice. A new approach to distributed context-aware computing. In A. Ferscha, H. Hoertner and G. Kotsis, eds., *Advances in Pervasive Computing*. Austrian Computer Society, 2004. ISBN 3-85403-176-9.