Moving Omega to a C++-based Platform

John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
plaice@cse.unsw.edu.au

Paul Swoboda
pswoboda@cse.unsw.edu.au

Abstract

The code for the Omega 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 Omega Translation Processes (OTPs) has been completely rewritten and placed in C++ libraries, whose methods are called by the typesetting engine. The Pascal Web part of Omega no longer uses change files. The overall Omega architecture is now much cleaner than that of previous versions.

Introduction

Since the first paper on Omega was presented at the 1993 Aston TUG Conference, numerous experiments have been undertaken in the realm of multi-lingual 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. See, for example, the articles published in the 2003 EuroTEX and TUG conferences. Clearly, building an extensive new system will require substantial effort and time, both at the design and the implementation levels. In this article, we present the interim solution for the stabilisation of the existing Omega code base, with a view towards preparing for the design and implementation of a new system.

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 `stdc++` library. The entire Pascal Web code for fonts and OTPs, including that of Donald Knuth, has been completely rewritten in C++ and placed in libraries. Clean interfaces have been devised for the use of C++ code from the remaining Pascal code.

This exercise serves two purposes. The first is to stabilise and to simplify the existing Omega distribution. The second is to lay the groundwork for a forthcoming, complete reimplementation of open typesetting software.

Problems with Pascal Web

When we examine the difficulties in creating Omega 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 attempted to seriously modify 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 has stated that one of the reasons he stopped working on TeX was his fear that he might break it.

For a skilled, attentive programmer, developing a piece of code that is not going to evolve, it is possible to write working code, *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 the change file mechanism of Pascal Web. 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. As a result, maintaining a Pascal Web program is a very slow process. Back in 1993, when the first Omega 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 compile-time fixed-size arrays are obvious and well known. This was addressed publicly by Ken Thompson in the early 1980s and has been addressed in Omega as well as in the `web2c` framework, simply by changing these sizes. However, there are other problems in the way these arrays are used. The `eqtb`, `str_pool`, `font_info` and `mem` arrays all have programming interfaces using fixed-size arrays. Whenever these interfaces are insufficient, the TeX code simply makes direct accesses to these arrays. As a result, 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.

The single input buffer for all active files of `tex.web` turned out to be truly problematic for implementing Omega's OTPs. Since an OTP 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, even though C itself has a datatype called `wchar_t`.

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.

The 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.

The Standard C++ library also provides built-in wide character support, including, in the GNU `stdc++` implementation, the full `iconv` functionality for character-set conversion between Unicode and any other imaginably-used character set.

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.

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.

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.

Organisation of the Omega 2 Code Base

Obviously, we are moving on. Our objective is to include the existing Omega 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.

Pascal Web Components. 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, we have passed through 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 thrown out, or directly integrated in the corresponding `.web` files.

Characters, Strings and Files. For characters, TeX has two types, `ASCII_code` and `text_char`, the respective internal and external representations of 8-bit characters. The new Omega 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, as interpreted by the `tangle` program, are each assigned a `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` can be of the same data type.

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 `stdc++` local routines. As a result of these choices, the vast majority of the Omega code can simply assume that characters are 4-byte Unicode characters.

Fonts and OTPs. In terms of numbers of lines written, this is the most significant part of the new Omega; however, because we are using standard OO technology, it is also the most straightforward.

With respect to fonts, significant energy expended, both in the original code as well as in previous Omega implementations, for bit packing of fields in binary font formats, which are stored in memory as they are on disk. By providing a simple OO interface in the character-level typesetter of the Omega engine, we have been able to greatly simplify the code for both the typesetter, as well as the font utilities for conversion between formats.

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

Prospects

At the time we are writing, this work is not completely finished. Nevertheless, it is well advanced and detailed documentation will be forthcoming on the Omega website.

If we view things in the longer term, we are clearly moving forward with two related goals, the stabilisation of existing Omega 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.