Preface

During the ten or so years of my career prior to joining Netscape in 1998, I had the good fortune to work on a wide variety of projects, on an equally diverse set of platforms. I worked on an embedded kernel of my own design for a somewhat obscure CPU (the TMS34020). I obtained experience in Windows kernel development, writing file systems drivers for the Windows NT and Windows 98 kernels, to support the development of a Network File System (NFS) client. In user space, I mostly specialized in user interface development, initially developing Motif (Z-Mail) and OpenWindows applications for UNIX, eventually getting exposure to Win32 and the Microsoft Foundation Classes (MFC) toolkit on the Windows platform. I even had the chance to write code for the Classic Mac OS to support a project that would be shipped by Apple, using the Mac Toolbox application programming interface (API). All of this code was written in the C language, and all of it was highly nonportable, written only with a concern for the job, and platform, at hand.

But then I joined Netscape, as a UNIX expert of sorts. Initially, I was assigned the task of working on bugs in the 4.x Netscape browser, specifically handling defects filed against a port of the browser to IBM's Advanced Interactive eXecutive (AIX) platform. Netscape had a contract with IBM to ensure that bugs filed against the AIX version of the Netscape browser, or bugs that IBM considered important, were fixed in a timely fashion, and this contract funded my hiring. Similar deals were cut with SGI, Hewlett-Packard, and Sun, and perhaps others, and these deals funded additional Netscape staff. Two or three of us were assigned to deal with AIX, specifically.

During this time, portability had not yet been perfected at Netscape. Although much of the codebase of Netscape was portable, the project did not have a unified build system, and the user interface code was completely

platform specific. Many bugs had a decidedly platform-specific nature to them (hence the need to have separate teams to support individual platforms). Code that worked flawlessly on the Windows version of Netscape ran horribly on less well-supported platforms. Not all platforms had the same set of features, and features varied from platform to platform.

Within a year of joining Netscape and fixing AIX bugs, I somehow earned my way onto the Netscape Instant Messenger team, and worked on the new codebase based on the open source Mozilla platform. This team, which consisted of three engineers, was tasked with porting the AOL Instant Messenger client to the Netscape browser. The Netscape IM team was hastily formed right after AOL acquired Netscape, to try to bring AOL-based functionality into the application. (The other major AOL property integrated into Netscape was support for AOL Mail.)

The new Netscape client, in development at that time, was, as mentioned previously, based on the open source codebase named Mozilla. This codebase, at the time, was largely the product of Netscape engineers located in offices located in San Diego, and Mountain View, but contributions from the open source community were on the rise. (I refer to the project as Netscape/Mozilla in the remainder of this Preface.)

Netscape was in fierce competition with Microsoft for the browser market at this time, which meant the browser of course had to work well, and ship on time on the Windows platform. Netscape also generated significant advertising revenue through the Netscape portal, and clicks there were highest when a new version of the browser was released, and tens of millions of users visited the portal to download a fresh copy of Netscape. Supporting Netscape not only on Windows but also on Mac OS and Linux helped keep the number of visits high and generate revenue. So, Linux and Mac OS were treated as equals with Windows within the Netscape culture, not only because it was the morally right thing to do (as many of us believed), but also because every visit to the Netscape portal was important to the bottom line of the company.

Netscape/Mozilla was a complete departure from anything that I had ever seen or worked on before. First of all, this new version of Netscape was not just a browser, it was a platform, capable of hosting applications. (The primary examples shipped with Netscape were AIM, the Mail/News client, a WYSIWYG HTML editor named Composer, the Chatzilla IRC client, and the browser itself. Extensions, such as those available for Firefox today, are closely related.) Instead of building the graphical user interface (GUI) for these applications in C or C++, using APIs provided by a native platform

GUI toolkit such as MFC or Gtk+, Netscape/Mozilla-developed technologies were used instead. Static Extensible Markup Language (XML) files were used to describe the layout of the user interface, much like HTML is used to describe the layout of a Web page. The XML dialect developed for this purpose was called XML User Interface Language (XUL). JavaScript code, linked to the XML elements via attributes much like JavaScript is linked to HTML elements in a Web page, was used to respond to menu item selections and button clicks. To build an application for Netscape/Mozilla, all one needed was this combination of XML and JavaScript; and because both JavaScript and XML are intrinsically portable, so were the user interfaces that were designed using these technologies. When JavaScript wasn't enough to do the job (as was the case for any real applications, like those shipped with Netscape and Mozilla), JavaScript code could call C++ functions that provided the guts of the application, housed in shared libraries. These shared libraries, or components, were directly supported in the Netscape/Mozilla architecture via two technologies: XPConnect and XPCOM. These technologies allowed component developers to define platform-agnostic interfaces using an Interface Description Language (IDL). Using XPCOM and XPConnect, JavaScript code could query for the existence of a component, and from there, query for a specific interface. If all was good, the JavaScript code was handed an object that it could call just like any other object, except the object was written in C++, and was capable of doing things that JavaScript programmers could only dream of. The interfaces, by their nature, were highly platform agnostic.

The impact of the work done to support portability in the Netscape/Mozilla architecture was not, quite frankly, immediately apparent to me. But, over time, I came to appreciate the power of such an approach. The positive effects of the decisions of those who came up with the architecture are indisputable; during its heyday, Netscape was shipping tens of millions of browsers to users, not just for Windows, Mac, and Linux, but for SunOS, AIX, HP-UX, SGI Irix, and a host of other UNIX-based platforms. The "tier-1" platforms (Mac OS, Linux, and Windows) literally shipped at the same time. Each of these ports had, for the most part, the same feature set, and mostly shared the same set of bugs and quirks. To achieve portability at such a grand scale required a very special architecture, and it is one of the goals of this book to give you a good understanding (if not an appreciation) for how the Netscape/Mozilla architecture directly impacted the portability of the codebase.

However, it was not just the architecture of Netscape/Mozilla that made the browser and related applications (AIM, Mail, Composer) portable. To pull this sort of thing off, one needs not only a solid architecture, but also a culture of policies and procedures that put cross-platform development high on their lists of priorities—as well as large doses of discipline to ensure these best practices were followed. Tools, such as Tinderbox and Bugzilla, both of which are described in this book, were invested in heavily by Netscape and Mozilla, and the investment paid off in spades. Engineers were forced to consider other platforms, not just their own, and a regression found during daily testing on just one platform could halt development on all platforms, not just the one affected, because Netscape and Mozilla realized that the only true way to achieve portability was to deal with the issues in the here and now. A good chunk of this book steps away from the code, and describes these best practices, because no matter how good your architecture is in supporting cross-platform, you have to work all the platforms you plan to support with the level of care and devotion to detail if they are going to make it to the finish line with the same levels of quality.

Similar to the way that the programs we write are made up of data structures and algorithms, portability, in my opinion, consists largely of architecture and process, and this conviction is at the foundation of the book that you now hold in your hands.

How This Book Is Organized

This book is organized as a series of themed chapters. Most of these chapters consist of a set of items, with each item covering a specific topic supporting the chapter's overall theme. Early in the book, you will find sections that contain items presenting best practices that must be communicated to the entire development organization, including management, development, and testing. Later chapters cover software-engineering topics that management should be aware of, but these chapters have been written primarily for readers who will be implementing the code. In all, there are 23 items presented in these initial chapters.

The implementation of a user interface is a major concern in the development of cross-platform desktop applications. Item 23 serves to introduce the topic. The final two chapters of the book are therefore devoted to cross-platform GUI-related topics. The first of these chapters

provides a comprehensive introduction and tutorial to the wxWidgets cross-platform GUI toolkit. After reading my introduction to wxWidgets, you may want to check out Prentice Hall's detailed treatment on the subject, *Cross-Platform GUI Programming with wxWidgets,* by Julian Smart, et al. wxWidgets is not the only cross-platform GUI toolkit available for use in your projects. Another capable, and very popular cross-platform GUI toolkit, Qt, is not covered in this book. However, if you are interested in Qt, a few books are currently available that cover the subject in great detail, perhaps most notably *C++ GUI Programming with Qt 4,* by Jasmin Blanchette and Mark Summerfield, also published by Prentice Hall (see also their Qt 3-specific book).

The last chapter of this book, Chapter 9, "Developing a Cross-Platform GUI Toolkit in C++," starts with an introduction to the cross-platform GUI toolkit, XPToolkit, which is a major component of Netscape and Mozilla's cross-platform browser suite. It then goes on to detail the implementation of a toolkit I created especially for this book, Trixul. Trixul has many of the same attributes that made up the Netscape/Mozilla XPToolkit we used at Netscape to construct cross-platform GUIs. Both XPToolkit and Trixul, for example, allow you to describe the user interface of an application in XML and JavaScript, both support a component-based model that allows the user interface to call into shared libraries written in C or C++, and both are highly portable. However, there are two major differences between Trixul and the Mozilla offering. First, Trixul is a desktop GUI toolkit, whereas XPToolkit applications execute within the context of a Web browser only. Second, the overall design of Trixul is (I think) much simpler than XPToolkit, which (I am certain) allowed me to do a much better job of describing both the architecture of the toolkit, and the concepts behind its design, than I otherwise would have been able to do. Although I don't really expect that you will want to design a custom cross-platform GUI toolkit for your project, there is much to be learned from taking a look at how Trixul was designed and implemented.

The chapters, for the most part, have been written such that they can be read in any order. If you are in technical management, I recommend that you read the following chapters carefully:

- Chapter 1, "Policy and Management"
- Chapter 2, "Build System/Toolchain"
- Chapter 3, "Software Configuration Management"

Technical managers who are so inclined should consider at least scanning through the following sections:

- Chapter 4, "Installation and Deployment"
- Chapter 5, "Operating System Interfaces and Libraries"
- Chapter 6, "Miscellaneous Portability Topics"
- Chapter 7, "User Interface"

Developers should plan to read the entire book, although you might want to invert the recommendations made here for technical managers, and skim what they are supposed to read carefully, and read carefully what they are supposed to skim. If your focus is user interface development, I recommend reading Items 22 and 23, and Chapter 8, "wxWidgets." If you are interested in GUI toolkit internals, or plan to help out with the development of Trixul (described in the following section), you will definitely want to read Chapter 9, "Developing a Cross-Platform GUI Toolkit in C++."

A Word about Trixul

Trixul is an open source project that I put together specifically to aid me in the writing of this book. In part, I had the same intentions of the original authors of the Gtk+, to learn by doing. However, the more relevant goal behind Trixul was to develop a simple, cross-platform toolkit, the architecture and design of which could be easily described in fewer than 100 pages, and understood by mere mortals without the need to read huge globs of code. The design is heavily inspired by Netscape/Mozilla (the Document Object Model [DOM], the Gecko layout engine, XUL, XPConnect, and XPCOM are all Netscape/Mozilla technologies that have analogs in Trixul); and although the details differ, I am certain that much of what you learn about Trixul's architecture will help you to understand Netscape/Mozilla. Not everyone will want, or need, to write his own GUI toolkit, but Netscape did, and so did America Online (a not-so-portable effort named Boxely was developed in the years following Netscape's demise), and perhaps it makes sense for your company, too. The story of Mozilla/Netscape's portability is not at all complete without a discussion of the way in which the user interface problem was solved, and Trixul is, in my opinion, the best way to get the idea across in a reasonable number of pages.

However, Trixul isn't just for learning. It is my sincere hope that Trixul will take on a life of its own as a viable, next-generation desktop GUI toolkit. The project, at the time of writing this book, is in its infancy. If you

like what you read about Trixul and are interested in contributing to its development, or work on a port, I certainly want to hear from you. You can learn more by visiting www.trixul.com or the project page at http://sourceforge.net/projects/trixul.

References

The following is a short list of books that are either mentioned directly in the text, or have influenced in some way the content of this book:

Andrei Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied* (Reading, MA: Addison-Wesley, 2001).

Jasmine Blanchette and Mark Summerfield, *C++ GUI Programming with Qt 3* (Upper Saddle River, NJ: Prentice Hall, 2004).

Randal E. Bryant and David O'Hallaron, *Computer Systems A Programmer's Perspective* (Upper Saddle River, NJ: Prentice Hall, 2003).

David R. Butenhof, *Programming with POSIX Threads* (Upper Saddle River, NJ: Prentice Hall, 1997).

Paul Dubois, *Software Portability with imake* (Sebastopol, CA: O'Reilly Media, Inc., 1996).

Erich Gamma, et al., *Design Patterns* (Reading, MA: Addison-Wesley, 1995).

Simson Garfinkel and Michael K. Mahoney, *Building Cocoa Applications: A Step-by-Step Guide* (Sebastopol, CA: O'Reilly Media, Inc., 2002).

Ian Griffiths, et al., *.NET Windows Forms in a Nutshell* (Sebastopol, CA: O'Reilly Media, Inc., 2003).

Greg Lehey, *Porting UNIX Software* (Sebastopol, CA: O'Reilly Media, Inc., 1995).

Syd Logan, *Developing Gtk+ Applications in C* (Upper Saddle River, NJ: Prentice Hall, 2001).

Scott Meyers, *Effective C++* (Reading, MA: Addison-Wesley, 2005).

Andrew Oram and Steve Talbot, *Managing Projects with make* (Sebastopol, CA: O'Reilly Media, Inc., 1993).

Eric S. Raymond, *The Art of UNIX Programming* (Reading, MA: Addison-Wesley, 2003).

Julian Smart, et al., *Cross-Platform GUI Programming with wxWidgets* (Upper Saddle River, NJ: Prentice Hall, 2006).

Bjarne Stroustrup, *The C++ Programming Language* (Reading, MA: Addison-Wesley, 2000).