УДК 004.75, 004.724.2

DESIGN CAUTIONS FOR THE MULTILINGUAL DATA PROCESSING IN P2P NETWORKS

Poryev G.V., National Technical University of Ukraine "Kyiv Polytechnical Institute", Kyiv, Ukraine

It is shown that the development of p2p software, especially server-side, should take into account the proper handling of various script systems. These include, in particular, ideographic systems, whose glyphs cannot be adequately mapped to an obsolete eight-bit character set encodings. It is therefore beneficial to use the most recent Unicode standard.

Introduction

Early electronic computing systems that featured textual input/output interfaces — teletypes, display terminals, etc. were, in most cases, conceived in the countries whose primary language utilized Latin alphabet. Because it consists of small number of fixed-shape symbols, it was relatively easy to implement those days. Due to binary nature of underlying storage mediums, character sets tended to be designed in amount of powers of two. For example, character set of 64 would fit for Latin minuscule and majuscule letters, Arabic digits, space bar, and end-of-file mark. Adding punctuation marks and special control characters (used to indicate how to process text in word processors) would require character set of 128, which was eventually devised in 1960s as ASCII-7 (American Standard Code for Information Interchange, 7 bit wide) standard [1]. Only 95 of 128 characters were printable, however.

The epoch of personal computers set a de-facto standard for characters to be represented by eight-bit bytes, sometimes called 'octets'. ASCII was subsequently extended to feature characters such as '½', '©', pseudo-graphic elements with single or double stroke lines, including combinations of both, number and radix signs, and various other symbols.

The problem encountered

It was at that time personal computers begin to spread across borders into the countries with non-Latin based languages. It had prompted the often-uncoordinated development of national character sets mostly in the manner of preserving the compatibility with ASCII-7 (as not to cause problems with already existing software) whereas adding national characters in the upper plane (with 7 bit on). One of the examples we believe is widely known to the readers would be so called 'codepage 866', loosely based on ГОСТ 19768-87 [2]. Unlike also Cyrillic 'codepage 1251', it preserves pseudo-graphics, and unlike 'KOI-8', it maintains alphabetical order of letters to simplify sorting operations. However, current versions of Windows operating systems employ both 866 and 1251 codepages for Cyrillic, for console and ANSI-mapped GUI, respectively, although it is customizable.

The advent of the Internet had only strengthened the need for some kind of character set unification, as the e-mail communications between countries and multilingual web resources begin their advance and, naturally, the need for wider range of symbols available to input and display, has grown substantially. It soon became ap-

parent that regular eight-bit character sets, initially introduced to feature national characters, brought more problems than they were resolving. The most significant of them was the need of conversion between the character sets for the same language, often without definitive information regarding what character set was used in the original text. For example, there were at least three aforementioned character sets invented for Cyrillic letters — codepages 866, 1251 and several versions of KOI-8. In addition, even for languages with Latin alphabet, various diacritic marks, often with syntactic meaning, do exist.

Internet became commercialized and widespread well before the concept of unification of character sets had grown to the apparent and obvious levels. In addition, marketing strategies and technological legacy of then-dominant OS vendors, such as Microsoft, IBM, and open-source community had not paid much attention to such unification for some significant time.

The solution to the problem

It was not until the mid-1990s, when the viable solution had actually entered the IT community. The most instrumental role in it was played by the Unicode [3, 4, 5].

Unicode is an industry standard allowing informational systems to adequately represent and manipulate text expressed in any of the writing systems, either existent or extinct, natural or artificial. Developed in tandem with the Universal Character Set standard and published in book form as The Unicode Standard, Unicode consists of a repertoire of about hundred thousands characters, a set of code charts for visual reference, an encoding methodology and set of standard character encodings, an enumeration of character properties such as majuscule and minuscule case, a set of reference data computer files, and a number of related items, such as character properties, rules for text normalization, decomposition, collation, rendering and bidirectional display order (for the correct display of text containing both right-to-left scripts, such as Arabic or Hebrew, and left-to-right scripts).

The Unicode Consortium, the non-profit organization that coordinates Unicode's development, has the ambitious goal of eventually replacing existing character encoding schemes with Unicode and its standard Unicode Transformation Format (UTF) schemes, as many of the existing schemes are limited in size and scope and are incompatible with multilingual environments.

Unicode's success at unifying character sets has led to its widespread and predominant use in the internationalization and localization of computer software. The standard has been implemented in many recent technologies, including XML, the Java programming language and modern operating systems.

Unicode has the explicit aim of transcending the limitations of traditional character encodings, such as those defined by the ISO 8859 standard which find wide usage in various countries of the world but remain largely incompatible with each other. Many traditional character encodings share a common problem in that they allow bilingual computer processing (usually using Roman characters and the local language) but not multilingual computer processing (computer processing of arbitrary languages mixed with each other).

Unicode, in intent, encodes the underlying characters — graphemes and graph eme-

like units — rather than the variant glyphs (renderings) for such characters. In the case of Chinese characters, this sometimes leads to controversies over distinguishing the underlying character from its variant glyphs (as in Han unification).

In text processing, Unicode takes the role of providing a unique code point — a number, not a glyph — for each character. In other words, Unicode represents a character in an abstract way and leaves the visual rendering (size, shape, font or style) to other software, such as a web browser or word processor. This simple aim becomes complicated, however, by concessions made by Unicode's designers in the hope of encouraging a more rapid adoption of Unicode [4].

The first 256 code points were made identical to the content of ISO 8859-1 so as to make it trivial to convert existing western text. A lot of essentially identical characters were encoded multiple times at different code points to preserve distinctions used by legacy encodings and therefore allow conversion from those encodings to Unicode (and back) without losing any information. For example, the "fullwidth forms" section of code points encompasses a full Latin alphabet that is separate from the main Latin alphabet section. In Chinese, Japanese and Korean (CJK) fonts, these characters are rendered at the same width as CJK ideographs rather than at half the width.

When writing about a Unicode character, it is normal to write "U+" followed by a hexadecimal number indicating the character's code point. For code points in the Basic Multilingual Plane (BMP), four digits are used; for code points outside the BMP, five or six digits are used, as required. Older versions of the standard used similar notations, but with slightly different rules. For example, Unicode 3.0 used "U-" followed by eight digits, and allowed "U+" to be used only with exactly four digits in order to indicate a code unit, not a code point [4, 5].

Unicode covers almost all scripts (writing systems) in current use today.

Although more than 30 writing systems (alphabets, syllabaries, and others) are included in Unicode, there remain many more still awaiting encoding. Further additions of characters to the already-encoded scripts, as well as symbols, in particular for mathematics and music (in the form of notes and rhythmic symbols), also occur. Michael Everson, Rick McGowan, and Ken Whistler maintain the list of such scripts and their tentative code block assignments on the Unicode Consortium Web site, at Unicode Roadmap. For some scripts on the Roadmap, encoding proposals have been made and are working their way through the approval process. For others, no proposal can be made until the scholarly communities involved can agree on the character repertoire and other details.

Among the scripts awaiting encoding are Egyptian Hieroglyphics, Babylonian and other cuneiforms, Phoenician, and Mayan, together with lesser-known scripts of Asia, Europe, Africa, and the Americas. Many of them are not understood, such as the Rongorongo of Easter Island, Linear A of Crete, and Meroitic of the Upper Nile.

Invented scripts, most of which do not qualify for inclusion in Unicode due to lack of real-world usage, are listed in the ConScript Unicode Registry, along with unofficial but widely-used Private Use Area code assignments. Similarly, many medieval letter variants and ligatures not in Unicode are encoded in the Medieval Unicode Font Initiative.

The readiness of environments

Although initial versions of market-ready Unicode standard were completed in 1991, it was not until 1993 when the first operating system using Unicode internally (and on all levels from kernel through user interface), Microsoft Windows NT 3.1 was released. However, marketing success of this release was limited, and Windows line of operating systems lacked Internet capabilities at that time, so it was no earlier than 1995 when Windows NT 3.51 for the first time included Internet server software as optional add-on pack, thereby marking the Unicode incursion into the worldwide web [6].

All subsequent versions of NT-based operating systems up to (and including) Windows Vista/2008 retained their native Unicode support in the form of UCS2 — fixed-width 16-bit character set incorporating Unicode BMP (see above). It is now obsolete and superseded by UTF-16, the difference being the ability to represent characters introduced after Unicode 2.0 with code points greater than 65535 [6, 7].

The Windows 9x line of operating systems (95, 98, ME) never had full Unicode support, as they inherited mostly 16-bit kernel. Although special option pack called 'Microsoft Layer for Unicode' was released to address this issue, it is no longer of significance as the 9x line is no longer developed.

The other widely popular family of UNIX-like operating systems, Linux, also did suffer from internationalization issues all throughout its history. Due to the distributed nature of its development and a significant expertise level required to set it up, the consequences was more severe than that in Windows. Until 2001 all major Linux kernels completely lacked Unicode support both internally and in user level so users were required to dig through numerous manuals, FAQs, and How-To's, often contradicting to each other, in order to set up proper language interfacing (if not English, of course) in their Linux distributions.

Following the unification trend presented by Windows NT family, 2001 year marked the first release of Linux kernel 2.4 which was redesigned as to process all underlying string data according to the UTF-8 specification. The UTF-8 was chosen in order to maintain binary and, to some extent, source compatibility with the software written for previous versions of Linux kernel, as string data in pure ASCII character set is binary identical to that in UTF-8.

The known implementations

Although the development environments for major operating systems were updated as to allow Unicode-aware software engineering, it is still relatively rare practice in the open-source and independent developer community to write software packages that does not suffer when confronted with the data in non-Latin character sets.

The client software for peer-to-peer (p2p) networks is not an exception. It seems that all of the modern popular p2p clients underwent some sort of Unicode awareness changes during their development lifetime. For example, the dominant eDonkey2000 network client called eMule was able to search content using Unicode since version 0.44b was released in 2004; as well as the Gnutella/Gnutella^2 network client Shareaza is Unicode-aware since version 2.1 also released in 2004 [8].

Most popular BitTorrent clients also seem to understand Unicode, either due to the environment they were designed in (for example, Azureus — Java BitTorrent cli-

ent supports Unicode because Java does so internally) or they were specifically made to, like μTorrent.

Summary

Taking into account the aforementioned considerations, we can make a set of recommendations to the software engineers intending to create new peer-to-peer software. These are the following:

1. It is beneficial to include multilingual support in the early stages of development. Engineering experience had shown that modifying core components of a large project as to accommodate 'larger-than-byte' characters (if they were not set this way previously) would require immense human and work resources, and potentially disrupt the compatibility with third-party software components not yet aware of such changes.
2. The Unicode Standard is widely accepted as best available solution regarding the multilingual data processing and is being implemented by large software corporations as well as independent developers.
3. It is important to keep to the latest version of officially published Unicode Standard as to minimize compatibility and code refactoring issues from the necessity to comply with its subsequent versions.

This research was conducted as a part of state-funded program under the government contract Ф25/624-2007.

References

1. Brandel, Mary. 1963: The Debut of ASCII: History of the origin of the ASCII standard.
2. ГОСТ 19768-87.
3. The Unicode Standard, Version 5.0, Fifth Edition, The Unicode Consortium, Addison-Wesley Professional, 27 October 2006. ISBN 0-321-48091-0.
4. Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard, Richard Gillam, Addison-Wesley Professional; 1st edition, 2002. ISBN 0-201-70052-2.
5. Unicode Explained, Jukka K. Korpela, O'Reilly; 1st edition, 2006. ISBN 0-596-10121-X.
6. Lucovsky, Mark (2000-08-09). Windows: A Software Engineering Odyssey.
7. Paul Thurrott's History of Windows Server 2003: The Road To Gold.
8. http://crlam.free.fr/unicode_support.html

Порев Г.В. **Рекомендации по многоязыковой обработке данных в одноранговых сетях**
Показано, что при разработке программного обеспечения для работы в одноранговых сетях необходимо принимать во внимание корректную обработку различных письменных систем. Такими системами являются, например, идеографические, символы в которых не могут быть адекватно представлены устаревшими 8-битными кодировками. Показано, что использование стандарта Unicode в этих случаях будет полезным.

Порєв Г.В. **Рекомендації щодо багатомовної обробки даних в однорангових мережах**
Показано, що при розробці програмного забезпечення для роботи в однорангових мережах необхідно брати до уваги коректну обробку письмових систем. Такими системами є, наприклад, ідеографічні, символи в яких не можуть бути адекватно представлені застарілими 8-бітними кодуваннями. Показано, що використання стандарте Unicode в цих випадках буде вигідним.

Надійшла до редакції
25 жовтня 2007 року