John C. Paolillo

Language, the Internet and access: do we recognize all the issues?

1. Introduction

The Internet is so ubiquitous and global that it is now widely regarded as a significant site of contact among people of all linguistic and cultural backgrounds. Representatives of language groups from the smallest to the largest see Internet access as a major policy objective, as do governments and policy makers worldwide. Policy discussions and academic research tend to focus questions of representation and dominance at either the global or local level, systematically neglecting a range of related social and technical issues such as the central role of written language, the linguistically non-neutral implementation of technical protocols, programming and markup languages and the social infrastructure governing the technical aspects of the Internet. When these are taken into account, it becomes clear that the current Internet promotes an emerging global diglossia, in which exclusively English is used for technical purposes, and all other languages are merely "content", without full range of use. Recognition of this issue points toward a need to emphasize truly language-neutral technologies for the Internet.

Public awareness of Internet language issues has focused on the "use of languages on the Internet". This notion is often not fully explicated, but three principal concerns are discussed more frequently than others: (1) the ability to produce web pages in whatever language is at issue (typically for language preservation), (2) the ability to use different languages to name Internet hosts and (3) the various adaptations of users to technical systems that were not designed for their own languages. The first of these is addressed through the Unicode effort and the World Wide Web recommendations process. The second of these dominated the agenda of the United Nations World Summit on the Information Society (held in Geneva 2003 and in Tunis 2005), resulting in the establishment of the Internet Governance Forum (IGF), and the modification of the Domain Name System, regulated by the US non-profit corporation ICANN, to allow International Domain names. The third is the subject of intense research among academics around the world, resulting in collections such as Danet/Herring (2007), Androutsopoulos (2006) and Wright (2004), among many other individual publications. However, the technical situation behind the first two issues remains disconnected from the academic research on Internet language use, and does not get the full examination it deserves, given its importance to the status of languages on the Internet.

In this plenary, I argue that the technical issues of language use on the Internet require a closer look. We examine three core Internet technologies: the representation of text in Unicode, the naming of Internet hosts in the Domain Name System, and the programming and markup of websites using HTML and web scripting languages. All three are shown to have a bias toward English, with other languages experiencing greater costs that could otherwise be technically unnecessary. This situation is interpreted in terms of the ethics of computer system design, and the sociolinguistic phenomenon of diglossia, to illuminate the ways in which technical design choices reflect extrinsic social values. We conclude with a discussion of principles for the remediation of technical language bias.

2. Use of text

The Internet is primarily a text-based environment, so the use of the Internet for communication presupposes and the ability to represent written language text is a basic requirement. In fact, information technology is heavily dependent on text in general, and so it follows that languages without written representations, or in particular, languages without "machine readable" written form, are at a disadvantage in their ability to use IT.

The principal effort to address this issue is the Unicode project, led by a consortium of technology companies (eight full members), government agencies (four, primarily South Asian), supporting and associate members (24) and individual members (102). The Unicode project began in the late 1980s as an effort to reconcile what were then vendor-specific, incompatible ways of representing non-English text (Becker 1988). The consortium officially incorporated in 1991, and by 1993 the Unicode standard was formally merged with ISO-10646 (a formerly competing encoding for text); it is currently in version 6 (Unicode 2011).

Unicode recognizes three problems relating to the representation of multilingual text: the problem of *encoding* and processing written language in machine-readable form, the problem of *rendering* machine-readable text in human-readable form, and the problem of *inputting* text into a computer by people. Unicode only addresses the encoding issue, and does not specify the rendering or input methods that should be used, permitting vendors (e.g. Unicode's commercial software company members) to compete to fill these needs. This delimitation of scope has delayed full Unicode implementation for many languages, especially those using non-Latin orthographies.

Unicode's solution to the encoding problem was to develop a universal encoding (hence "Unicode") for all written languages by assigning each distinct textual *character* in each writing system a unique binary number, or *code-point*. The notion of character in Unicode is an abstraction lacking a universally fixed definition.[1] It is implemented differently for different writing systems, e.g. Latin letters with diacritics are typically different characters, as are circle-enclosed Latin letters, but Han (Chinese) characters in Chinese, Japanese and Korean are expected to be unified across all three languages, even where these characters have divergent written shapes (and no circle-enclosed versions are recognized). Code points are fixed and finite in number. Unicode is limited to 1,114,112 unique code-points, so no more than this number of characters can ultimately be recognized without significant revision; 109,499 code points are actually defined at present.

Unicode code points are broken up by numerical range into *planes* (of which there are 17) and blocks (units of up to 256 characters). Each plane has a maximum of 65,536 code points. The lowest-numbered plane (zero) is known as the *Basic Multilingual Plane* (BMP).

[1] "*Abstract character*: A unit of information used for the organization, control, or representation of textual data. When representing data, the nature of that data is generally symbolic as opposed to some other kind of data (for example, aural or visual). Examples of such symbolic data include letters, ideographs, digits, punctuation, technical symbols, and dingbats. An abstract character has no concrete form and should not be confused with a glyph. An abstract character does not necessarily correspond to what a user thinks of as a 'character' and should not be confused with a grapheme [...]." (Unicode 2011, 66).

Outside of the BMP, the currently defined planes are the *Supplementary Multilingual Plane* (for historical scripts and musical and mathematical symbols), the *Supplementary Ideographic Plane* (for additional Chinese, Japanese and Korean ideographic symbols), the *Tertiary Ideographic Plane* (reserved for additional ideographs) and the *Special Use Plane* (for application-specific purposes). The remaining 13 planes are not presently assigned, but reserved for future use.

The allocation of the BMP, by script and block, is illustrated in Figure 1. The largest share of the BMP is taken up by East Asian Languages: there are 111 blocks used for "unified CJK", which is the Chinese character set common to Chinese, Japanese and Korean. Identifying this set was a major preoccupation of Unicode in the 1990s. In addition to this, there are 55 blocks for Southeast Asian languages, 11 for South Asian scripts, 10 for Latin, 4 for Cyrillic, and one for Greek and Coptic. Other assignments are made for African, American, Middle Eastern and linguistic scripts.

The lowest block of the BMP contains the most commonly used Latin characters; this single block is generally sufficient for Western European languages, being the same as the 1989 specification ISO-8859-1 (Latin-1). The map of the character assignments in this block is given in Table 1. This block consists of assignments for upper and lower case Latin characters, including vowels with diacritical marks, special symbols common in Euro-American print usage, and 52 "control codes", most of which are presently unused, but are retained for compatibility with older encodings. The lower half of Block 0 is identical to a 1963 standard known as US-ASCII or simply ASCII, the most commonly used character set prior to Unicode; ASCII is only adequate for representing US English.

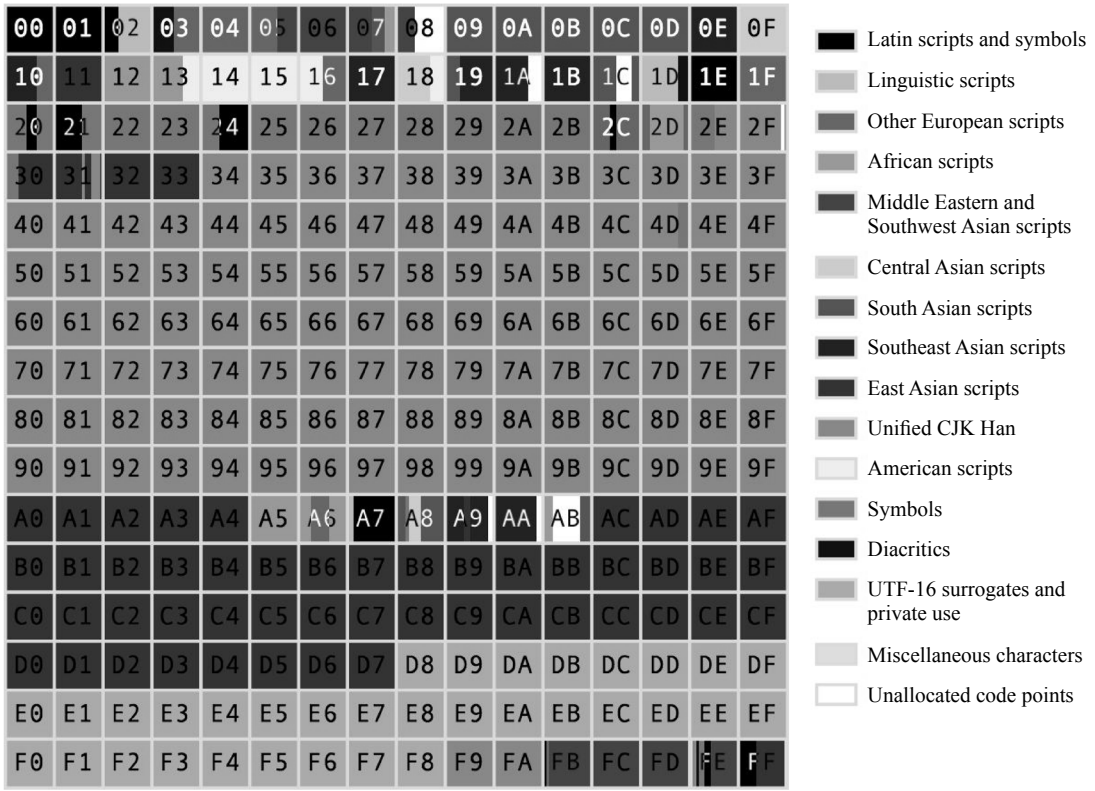

Figure 1: The Basic Multilingual Plane of Unicode
(public domain image: http://en.wikipedia.org/wiki/File:Roadmap_to_Unicode_BMP.svg)

One might be tempted to argue from the existing Unicode character assignments that most of the existing Unicode standard is devoted to the needs of Chinese, Japanese and Korean, and it therefore exhibits a bias toward those languages. This would be a superficial view, however, because the actual bias depends on other technical aspects of Unicode's use. Unicode has been designed to preserve a continuity of standards from ASCII to ISO-8859 to the present Unicode, thereby privileging the encoding of US English. In terms of code-point assignments, this advantage is small, but has broader ramifications.

Unicode offers three options for representing text: UTF-32, a 32-bit fixed-width encoding, UTF-16, a 16 bit variable-width encoding, and UTF-8, an eight-bit variable-width encoding. Fixed width encodings require the same amount of space for each character, regardless of its location in the Unicode system. Variable-width encodings allow some characters to be encoded in less space, while others take up more. In UTF-8, the most common Unicode text encoding, characters of 7-bit ASCII are encoded using the value of the code-point in a single eight-bit byte, whereas other Unicode characters (including Latin with diacritical markings) are encoded using 2, 3 or 4 bytes. For efficiency, variable-width encodings should encode more frequently used characters in shorter sequences; for UTF-8 it appears that the design assumption was that Unicode would be most commonly used for US-English. The advantage cited, however, is that any pre-existing ASCII text remains valid in Unicode, without modification.

	ISO-8859-1 (Basic Latin)															
	US-ASCII															
	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
0	NUL	DLE	SPC	0	@	P	`	p	XXX	DCS	NBSP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q	XXX	PU1	¡	±	Á	Ñ	á	ñ
2	STX	DC2	„	2	B	R	b	r	BPH	PU2	¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	NBH	STS	£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t	IND	CCH	¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	NEL	MW	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	SSA	SPA	¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	,	7	G	W	g	w	ESA	EPA	§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x	HTS	SOS	¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y	HTJ	XXX	©	¹	É	Ù	é	ù
10	LF	SUB	*	:	J	Z	j	z	VTS	SCI	ª	º	Ê	Ú	ê	ú
11	VT	ESC	+	;	K	[	k	{	PLD	CSI	«	»	Ë	Û	ë	û
12	FF	FS	,	<	L	\	l	|	PLU	ST	¬	¼	Ì	Ü	ì	ü
13	CR	GS	-	=	M	]	m	}	RI	OSC	SHY	½	Í	Ý	í	ý
14	SO	RS	.	>	N	^	n	~	SS2	PM	®	¾	Î	Þ	î	þ
15	SI	US	/	?	O	_	o	DEL	SS3	APC	¯	¿	Ï	ß	ï	ÿ

Table 1: Character Assignments in Block 00 of Unicode

The significance of this design choice, while greater than that of numeric code-point assignments, might still appear small. For most European languages using Latin characters, the extra difficulty of encoding diacritics is small, and it does not affect the core fifty-two Latin characters shared with English. The encoding would be no more complex if diacritics were encoded as separate characters. For scripts such as Arabic script, Cyrillic or Greek, the situation is different, affecting each and every character in a text. Texts in such scripts require approximately twice as much space to store and incur twice as much transmission cost as comparable texts in Latin.

An even greater cost to international text is the same one that Unicode sought to preserve for English: any previously encoded text for something other than US-ASCII is simply not valid Unicode. This includes ISO-8859-1, and the many (non-standard) eight-bit encodings of Arabic and Cyrillic, alongside others (e.g. East Asian and South Asian encodings). This, alongside the many years' lag in support for input and rendering of the same scripts, imposes significant legacy-text conversion costs on non-English and non-Latin script use (Hardie 2007; McEnery/Xiao 2005).

3. ASCII and the Internet: domain names

Another major area of contention over the use of languages online has been in the naming of Internet hosts. Internet host names are maintained by the Domain Name Service (DNS), a system conceptualized in 1981 to replace the original file-based directory scheme known as HOSTS.TXT (Rader 2001). The DNS was conceived as a way to maintain a global, distributed directory system within a delegated model of hierarchical control. As a global system, it was intended to index every host on the Internet; as a distributed system, it was intended to permit the index to be updated in a responsive and timely fashion, to avoid the cumbersome centralized management that had been a problem with HOSTS.TXT.

The DNS was established in 1985 under a US Department of Defense contract for the ARPANET, the precursor to the Internet. DNS authority was essentially delegated by sub-contracts under the Department of Defense and the National Science Foundation from 1985 to 1997; in 1995, Network Solutions Inc. (NSI) operated the DNS and began charging fees for host registration. Because of this commercialization and the monopoly control of whoever was to run the DNS, DNS authority was heavily contested until 1998, when a new legal structure was forged, and the US Department of Commerce forced the transfer of DNS authority from NSI to a new body, the Internet Corporation for Assigned Names and Numbers (ICANN), which runs the DNS and other aspects of the Internet to this day (Rader 2005).

On a technical level, the DNS is a set of protocols for the resolution of host names, which serve as human-readable mnemonics for computers on the Internet. Nothing in the Internet protocols requires that a host be named this way; an Internet host need only have an Internet Protocol number (or IP number, a 32 bit number that serves as a computer's address, also administered by ICANN). The sole technical role of the DNS is to translate conveniently remembered mnemonics into IP numbers, which can be done (arbitrarily) by any Internet host designed to use the DNS protocols. On a political level, the design-

ers of the DNS intended that there be only one such service, centrally managed and controlled.[2] Because of this, the DNS and the naming of Internet hosts have enormous commercial and political significance, involving the legal use of commercial trademarks, freedom of political speech and minority rights.

Under the DNS protocols established in 1985, hostnames are limited to a strict subset of seven-bit ASCII characters: the 26 lower case Latin letters a to z (with no diacritics), the digits 0 through 9, and hyphen. Clearly, this technical limitation favours US English at the expense of all other languages; the lack of Latin characters with diacritics being a long-standing rub (Pargman/Palme 2009). From the time that ICANN assumed control over the DNS, international pressure over this issue from governments and community groups intensified, leading to the key concession at the UN World Summit on the Information Society (WSIS) 2003 and 2005, where ICANN agreed to establish the Internet Governance Forum (IGF). An outcome of the WSIS and IGF processes was the implementation of Punycode, a seven-bit encoding of Unicode suitable for use on the DNS. The intent of Punycode was to permit the use of Unicode for the naming of Internet hosts, to respond to the need for "internationalized domain names" (IDNs); in 2009, a process for registration of top-level internationalized domain names was initiated by ICANN.

As it relies on Unicode, the Punycode implementation of IDNs has all of the same problems as text encoding in Unicode, but actually in an even more severe form, as Punycode is limited to bytes corresponding to the subset of ASCII already used by the DNS. Thus, Punycode is a variable width encoding in which the lower case characters of ASCII are unchanged, and other characters require two more bytes to encode. For example, the Greek word παράδειγμα becomes hxajbheg2az3al in Punycode, and the domain name παράδειγμα.gr becomes xn--hxajbheg2az3al.gr. The result of a Punycode encoding is thus, not in general human-readable without considerable application support.

Figure 2 is an attempt to indicate how this might affect different languages for which information is available; the pages for "What is Unicode" in each of the available script encodings as of November 2007 were converted to Punycode word-by word, and heatmap histograms were plotted. Each column in Figure 2 represents a distinct language/script, with brighter areas indicating higher frequency. Hence we can see that English has fairly short typical word lengths (under ten ASCII characters), whereas Albanian, Finnish, French, Slovenian and Turkish all have words of more than 20 ASCII characters in length (indicated by the horizontal white line). Hence the number of characters required to encode a typical hostname in these languages can be expected to be greater.

These requirements do more than make hostnames longer in non-English scripts; they also impose upper limits on what a valid hostname can be that are sharply lower for non-English languages than for English. The DNS requires that hostnames consist of dot-separated fields, that the fields not be longer than 63 ASCII characters, and the total length of the name not be longer than 255 ASCII characters, including all field-delimiting dots and

[2] Users, institutions and ISPs can configure their computers to use an alternative "DNS root" from that overseen by ICANN. Many alternate roots do exist, and some were instrumental in increasing pressure on ICANN for internationalizing domain names. ICANN, however, has considerable financial incentive to downplay the significance of alternate roots, to avoid what it calls "fragmenting the Internet" (Kahn 2006).

top-level domain (e.g. .com, .net, country code domain such as .fr, or internationalized top-level domain, such as .рф [Russian Federation]). As a consequence, some desired domain names are likely to be impossible to register. A three-word name in a language where typical words encode as 20 or more ASCII characters would raise this problem, whereas English does not encounter it until six-word combinations are used for fields. Once a large number of single-word names are registered, as happened quite quickly with English, longer names are required; IDNs are likely to experience this as well, and the ICANN will eventually have to face an IDN issue from these field and hostname length limits. Since these limits come from obsolete data types,[3] it is not clear why ICANN has worked so hard to keep them in place.

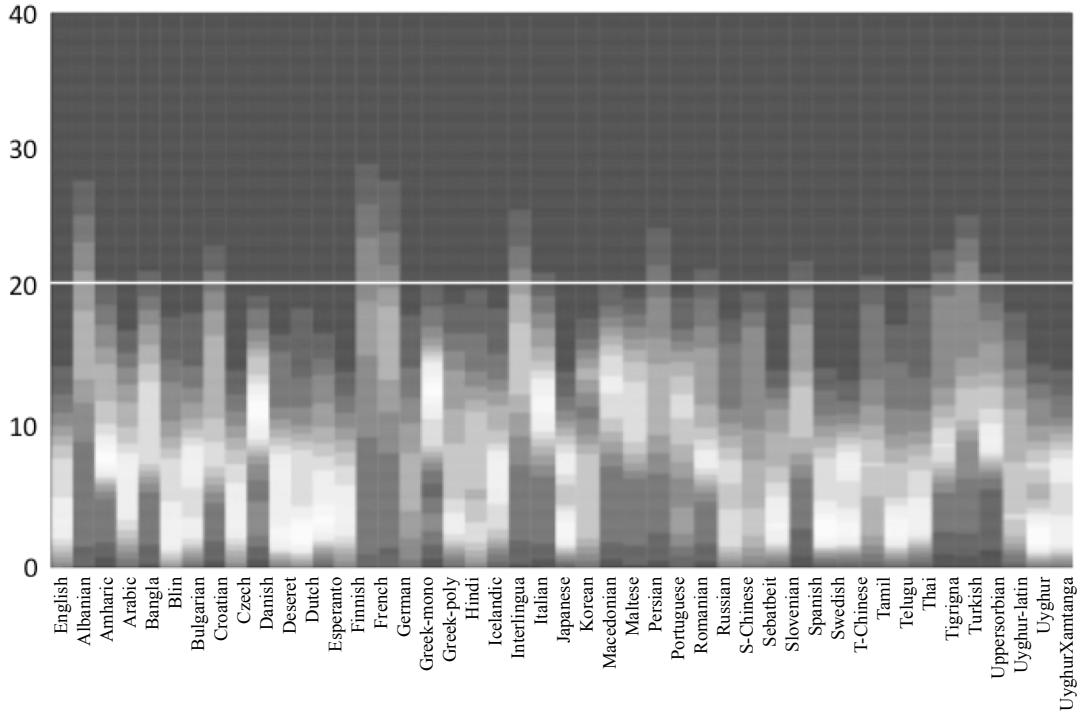

Figure 2: Typical Punycode word lengths in the text "What is Unicode" for 45 language-script combinations

The DNS is hardly alone among Internet protocols in uniquely privileging ASCII. Most other technical protocols do the same. The WHOIS protocol, which provides public access to the assignment of IP numbers, and basic directory services identifying parties legally and technically responsible for each Internet host, is entirely based upon ASCII (Daigle 2004). SMTP, the data-exchange system at the heart of all email, has been extended to permit messages to have content from any character set, but headers and other information used directly by SMTP remain in ASCII (Klensin 2001). Even the HTTP protocol underlying the World-Wide Web requires command, error and header information to be transmitted in ASCII or ISO-8859-1 (Berners-Lee et al. 1996). Others could be mentioned as well. What is remarkable about ICANN's management of the DNS is the slow pace of its adaptation in the face of intense public demand and mounting international pressure (Mayer-Schönberger/Ziewitz 2006). Should the remaining protocols ever become internationalized, we can expect similar difficulties with them as well.

[3] The 255-character limit suggests a Pascal data type originally defined in 1971.

4. Web markup and programming

Another major role of language is found in the markup and programming of web sites, as contrasted with the development of content; markup refers to the HTML, XML, RDF and other markup vocabularies that are used to format web page content, and to indicate something about its semantics for whatever applications use it. While the content may be in whatever language is desired for the target audience, markup must be in the formally specified vocabularies used for markup. Similarly, web programming is the writing of processing algorithms to be used either by the server, before data is delivered to the client, or by the client program, to dynamically control the presentation of data to the user. Web programming is accomplished in formally specified scripting languages.

Purpose	*Name*	*Data*	*Identifiers*	*Keywords*
General data definition	XML	Unicode	Unicode	Unicode
Formatting text	HTML	Unicode	Unicode	ASCII
Markup definition for XML	XML-DTD	Unicode	ASCII	ASCII
	XML Schema	Unicode	Unicode	ASCII
Transformation of XML to other formats	XSLT	Unicode	Unicode	ASCII
Server-side programming	Python	Unicode	Unicode	ASCII
	Ruby	Mainly ASCII	ASCII	ASCII
	Perl	Mainly ASCII	ASCII	ASCII
	PHP	Mainly ASCII	ASCII	ASCII
Client-side (browser) programming	JavaScript	Unicode	ASCII	ASCII
	ECMAScript	Unicode	ASCII	ASCII
Database query language	SQL	Mainly ASCII	ASCII	ASCII

Table 2: Support for Unicode in web markup and programming languages

Because of the Unicode effort, the international spread of the Internet and the development of web standards under the Worldwide Web Consortium (W3C), Unicode support is now seen as a requirement for most programming languages. Currently, the web markup languages HTML and XML support Unicode. All of the major web scripting languages, whether on the server side like Perl, PHP, Python, Ruby, and others, or on the client side, like JavaScript and ECMAScript, now support Unicode, although this means different things for different markup or programming languages.

Table 2 lists some common markup and programming languages, describes their functionality, and indicates the level of their support for Unicode. The "Data" column indicates the encoding permitted as data for each language; "Identifiers" indicates the encoding permitted for function, variable or other identifier names; "Keywords" indicated the encodings permitted (required) for special keywords defined in the language. From Table 2 we can see that only XML is defined to permit Unicode in all three. However,

this appearance is a little deceptive, because XML is used to represent data; the XML data formats themselves are defined in either XML-DTD or XML Schema, using ASCII keywords. Among the server programming languages, full support for Unicode data is rare; when it exists it is generally enabled by extensions, and secondary to ASCII or other 8-bit encodings. All the languages use ASCII keywords; it is worth examining these in more detail.

The Appendix lists keywords for several markup and programming languages: the data-formatting languages HTML and XML-DTD, the server scripting languages Python, Ruby and PHP, and the compiled languages Fortran, C and C++. These keywords represent computational features in each language; i.e. they are that part of the code, apart from mathematical operators etc., which is actually interpreted in terms of computer instructions, and has computational semantics. While the keywords vary by language, what is remarkable is that all of these keywords come from English words, phrases or abbreviations.

```
#!/usr/local/bin/cpython
# answer = raw_input('Do you think the Chinese language has value? (Yes / no)')
回答 = 读入('你认为中文程式语言有存在价值吗？(有/没有)')
# if answer == 'yes':
如 回答 == '有':
        # print 'Well, let's work together!'
        写 '好吧, 让我们一起努力!'
# elif answer == 'no':
不然 回答 == '没有':
        # print 'Well, not as a programming language'
        写 '好吧,中文并没有作为程式语言的价值.'
# else:
否则:
        # print 'Please give serious consideration before answering.'
        写 '请认真考虑后再回答.'
```

Figure 3: A Chinese Python program. The comment lines in English gloss the function of the Chinese-language code (www.chinesepython.org)

The English-lexified nature of computer programming is so entirely deep-seated that it is often treated by computer scientists and professionals as unremarkable. The number of computer programming languages is thought to exceed the number of human languages,[4] but it is English which is the overwhelmingly the parent language of the codes actually used to program computers. Yukihiro Matsumoto, the Japanese author of Ruby, chose to use English keywords for his creation. A few projects exist to "translate" programming languages into Chinese and other languages, but the remark most often made about them, including by their authors, is that they are strange, as can be seen from Figure 3. Such projects seldom gain traction; the Chinese Python project, for example, appears to have been abandoned. Native-language programming is a cause sometimes taken up for educational purposes, but it is generally assumed to have no other practical significance.

[4] The HOPL website (http://hopl.murdoch.edu.au/) currently catalogs 8,512 programming languages, whereas the Ethnologue (http://www.ethnologue.com/) lists 6,909 living human languages.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      lang="en">
  <head>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8" />
      <base href="http://www.efnil.org/mission-1" /><!--[if lt IE 7]></base><![endif]-->
    <meta name="generator" content="Plone - http://plone.org" />
    <link rel="kss-base-url" href="http://www.efnil.org/mission-1" />
      <script type="text/javascript"
              src="http://www.efnil.org/portal_javascripts/SubSkins/event-registration-cachekey3888.js">
      </script>
      <script type="text/javascript"
              src="http://www.efnil.org/portal_javascripts/SubSkins/fckeditor-cachekey6635.js">
      </script>
      <style type="text/css"><!-- @import url(http://www.efnil.org/portal_css/SubSkins/base-cachekey5313.css); --></style>
      <style type="text/css" media="screen"><!-- @import url(http://www.efnil.org/portal_css/SubSkins/projectprogressstyle-cachekey0528.css); -
-></style>
   <link rel="kinetic-stylesheet" type="text/css"
         href="http://www.efnil.org/portal_kss/SubSkins/at-cachekey0389.kss" />
  <link rel="kinetic-stylesheet" type="text/css"
        href="http://www.efnil.org/portal_kss/SubSkins/resourcecmfnotification-cachekey9316.kss" />
<title>Mission &mdash; European Federation of National Institutions for Language</title>
    <!-- Internet Explorer CSS Fixes -->
    <!--[if IE]>
        <style type="text/css" media="all">@import url(http://www.efnil.org/IEFixes.css);</style>
    <![endif]-->
    <link rel="author"
          href="http://www.efnil.org/author/varadi"
          title="Author information" />
    <link rel="shortcut icon" type="image/x-icon"
          href="http://www.efnil.org/favicon.ico" />
    <link rel="home" href="http://www.efnil.org"
          title="Front page" />
    <link rel="contents" href="http://www.efnil.org/sitemap"
          title="Site Map" />
    <link rel="search"
          href="http://www.efnil.org/search_form"
          title="Search this site" />
    <!-- Disable IE6 image toolbar -->
    <meta http-equiv="imagetoolbar" content="no" />
  </head>
  <body class="section-mission-1 template-document_view"
        dir="ltr">
    <div id="visual-portal-wrapper">
      <div id="portal-top">
        <div id="portal-header">
```

Figure 4: A fragment of the HTML in the home page of the EFNIL website (www.efnil.org)

Whatever its presumed status, the net effect of the English lexis is that the majority of the human-readable information associated with a web page tends to be English-based, requiring a technical English education to use, modify and otherwise appreciate. This is evident in EFNIL's own website: each page of the site, whatever the content language, is bracketed by about eight pages of HTML code like that in Figure 4 (alongside another five pages of JavaScript). This is true in spite of the fact that the Python-based Plone content management software system used by the EFNIL website was probably chosen for its superior Unicode support.

5. Diglossia in computer systems

In sociolinguistic terms, the situation with programming and markup on the web is a diglossia (Ferguson 1959): a situation in which two or more language varieties are used side-by-side for different functions. On the Internet, the “High” variety (English, in this case) is the exclusive code of all technical functions, while the “Low” varieties (all other languages) are used only as content, i.e., for everyday communication. This diglossia is global in scope, but concerns a specific set of technical functions in which English alone is used.

This situation is not a necessary part of computer programming. Keywords for programming and markup are actually stand-ins for abstract operations implemented by the computer, and any other replacements would work just as well. The abstract operations are no more naturally expressed in English than in any language: English programming keywords are often used in meanings only remotely related to their English meanings (e.g., “for” and “while” do not function as temporal prepositions, but indicate a computation

performed some number of times). The Unicode project maintains the Common Locale Data Repository (CLDR), which consists of lists of internationalized keywords for other domains: dates, language names, currencies, measurement systems, territories, time zones, etc.; a unified list of programming and markup terms would not be very long, compared to the number of language names, for example (the longest keyword list in the Appendix is HTML, with 211 terms). Moreover, many programming languages, such as C++, already use preprocessors that perform replacements similar to what is required for language localization. Technical extensions of this nature may not be trivial, but they are also not onerously difficult.

Nor is this situation a simple consequence of voluminous literature – e.g., technical documentation – in English. Such technical documentation can readily be found in many other languages; technical documentation is often localized when technologies are adopted (e.g., projects for localizing Linux, which mostly means translating documentation), but computer code almost never is. Moreover, English is not just the preferred language for programming; it is also the favoured language of scientific research and many international functions. Other linguists have noted the same English diglossia in other domains (Crystal 1997; Graddol 1999; Phillipson 2003), further connecting it to linguistic bias (Phillipson 1992, 27, 104).

This situation is, rather, an outcome of the historical development of computer and Internet technology in the post World War II era. The shift of the centre of scientific and engineering innovation from Europe to the US occurred at a time when computing technology was still largely nascent. Many innovations in computing were led by US-based industrial, military and academic research efforts; meanwhile the developing telecommunications infrastructure, on which computing has long depended, became cantered in the US. The rapid adoption of computing and Internet technology in the 1980s and 1990s took place in a context in which US-English was already favoured by a number of other factors. In spite of the significant contributions of many European scientists to computing (Bauer 2002; Tomayko 2002) and even Internet protocols, the commoditization of computers and their coupling to the telecommunications network largely benefitted US English.

Friedman/Nissenbaum (1996) examined bias in computer systems, and developed a set of categories for describing biases and identifying appropriate remedies. They recognize three kinds of bias: Pre-existing, technical and emergent bias. Pre-existing bias arises from circumstances outside the technical system, via attitudes toward, conflict with or oppression of groups of people; such circumstances tend to be institutionalized in the social system or culture. Technical biases exist when the technical systems themselves have some built-in bias. Emergent biases arise through the interaction of a technical system with the people using it in some particular context. Many times, emergent biases are the consequence of taking a technical system out of its original context and deploying it elsewhere. Evaluations that were not intended in the design of the technical system become part of its use in the new context, resulting in bias.

Different types of bias require different types of remedies. Pre-existing biases can only be addressed through social changes and are usually not the direct responsibility of computer system designers. Technical biases are part of the technologies themselves, and

therefore require technical changes, and addressing them is the responsibility of designers. Emergent biases involve properties of both the technology and the society in which it is used; they are sometimes the responsibility of designers, but they often involve interactions that can be difficult to anticipate.

As we have seen, there is a clear technical bias toward English in the core Internet technologies. Yet Internet technologies, such as the DNS and ASCII text encoding, were adopted from an US-English context (the ARPANET) by countries where English is not generally spoken; this suggests that an emergent bias. The development of Unicode and other technologies are technical efforts to address this emergent bias. At the same time, pre-existing bias is present in the attitudes of members of the technical professions, whether they be those emphasize backward-compatibility with ASCII at the expense of other encodings, or those that treat English as the only suitable medium for technical aspects of computing.

The English bias of the Internet is thus deeply inscribed in the Internet technologies themselves, but more than a mere technical bias; it has complex causes, and simple remedies are unlikely to successfully address it. At the same time, it should be clear that an important locus of these biases is in the community of engineers developing computing and Internet technologies. The organization operating the DNS (delaying the implementation of IDNs for fifteen years) and the technology consortia creating Unicode (deciding that only ASCII would be a directly supported legacy encoding) and HTML (originally from CERN in Geneva, Switzerland) all belong to the same international community of researchers, academics and technologists; they hold meetings to discuss these technologies in North America, Europe, Asia and elsewhere, and citizens of EU member countries are prominently represented among the participants. For both technical and social reasons, a program to address linguistic bias in Internet technologies should begin with these people.

6. Designing a program for change

The need to address the issue of English bias and diglossia in the Internet technologies should be evident. Diglossias are rarely stable, and tend to develop toward one of two outcomes: inter-generational language shift to the High code (English) or replacement of the High code by the Low code in High functions (i.e., the use of non-English languages for programming computers). The first outcome is favoured when individuals perceive greater rewards, economic or otherwise, in using the High variety. But to the extent that the European Union values its language diversity, the second outcome is the more desirable one. Since the technologies of the Internet impose high costs for non-English languages, maintaining Internet infrastructure, web sites, and language-localized information resources costs more in countries where English proficiency is less prevalent.

A program to address the US-English bias needs to recognize certain principles. First, it must acknowledge the importance of written language on the Internet. Much has been made of the prospect for voice, video and translation applications to assist Internet users in connecting across language barriers. However, any assessment of these technologies must be guarded. The dream of machine translation is as old as the origins of modern

computing in World War II military code breaking, but despite many rosy predictions and heavy research expenditures over the years, its promises remain unfulfilled. Video cannot be expected to do much better than television and film have already done in crossing language barriers; these technologies are much better at delivering powerful cultures to large audiences than they are at promoting the interests of minorities. As for voice, as long as the gateway technologies are programmed and configured via written text, voice telephony will only be an application.

Second, as suggested immediately above, the accessory role of specific enabling technologies needs to be recognized. The text-encoding problem targeted by Unicode is such a technology, as is the DNS, WHOIS and many others. These technologies should have high priority for internationalization, as that is the only real guarantee of access to the technologies. In some cases, as Unicode, and the IDNs of the DNS, internationalization is well underway, although its exact form may not be all that is desired. Others, like computer programming, need urgent attention. It will not be possible for international citizens to innovate Internet technologies in their own languages without learning computer programming. Since the web programming and markup languages are an important entry-point for computer programming, including among primary school age children, internationalization of these technologies should be a high priority.

Third, internationalization of the Internet requires social change in large social institutions, and that change needs to be coupled with other interests to be successful. Many of the core Internet technologies like the DNS, WHOIS, mail, etc., were designed very poorly from the perspective of security considerations. The DNS, for example, is susceptible to various kinds of attacks that direct users to incorrect hosts, and ICANN faces increasing public pressure to update the DNS for security. Coupling internationalization concerns with security is actually a necessity for both: many of the DNS security concerns come from the combined, sometimes conflicting assumptions of Unicode and ICANN when applied to IDNs.

Fourth, changes in the core technologies of the Internet will require significant institutional support. EFNIL is one such institution, but many others need to be engaged as well, from the many organizations overseeing different aspects of the Internet (ICANN, the IP registries ARIN, RIPE, APNIC, the Internet Engineering Task Force, etc.), to the national telecommunications regulators of member states, to technical consortia such as the Unicode Consortium and the W3C, to commercial entities manufacturing and marketing computers and Internet software; the list is long. Engagement of these institutions can happen in different times and places and by different means: via workshops sponsored at technology conferences for the WWW, Unicode, Internet networking, etc., via international for a such as the Internet Governance Forum and via institutional membership for EFNIL or member organizations in technology consortia. Corporations and other organizations tend to move in directions where there are clear rewards (e.g. access to markets), so there is an important role for regulation and market incentives as well.

Finally, the changes described here are likely to take some time to implement. The Internet itself was created with significant institutional support, but that support was more like a trickle over a long period of time than a sudden flood: packet-switching, the basis

for all Internet communications, was first envisioned in the early 1960s; it took nearly ten years to have an operating network, and another twenty to bring that same network to academia. Internationalizing the Internet probably will not take as long; for the sake of the future of the languages of Europe and the rest of the world, it also *needs* to be more rapid. Significant groundwork has been laid, which can continue to be improved as long there are people and institutions who persist in demanding its improvement, and who apply their efforts strategically.

7. References

Androutsopoulos, J. (ed.) (2006): Sociolinguistics and computer-mediated communication. Themed issue, *Journal of Sociolinguistics* 10.4, 419-438.

Bauer, F. (2002): Pioneering work on software during the 50s in Central Europe. In: Hashagen, U./Keil-Slawik, R./Norberg, A.(eds.): *History of computing: software issues*. Berlin: Springer Verlag, 11-24.

Becker, J.D. (1988): *Unicode 88*. Palo Alto, CA: Xerox Corporation. Reprinted 1998 by the Unicode Consortium. Internet: http://unicode.org/history/unicode88.pdf.

Berners-Lee, T./Fielding, R./Frystyk, H. (1996): *Hypertext Transfer Protocol – HTTP 1.0*. Internet: www.ietf.org/rfc/rfc1945.txt.

Costello, A. (2003): *Punycode: A bootstring encoding of Unicode for Internationalized Domain Names in Applications* (*IDNA*). RFC-3492. IETF Network Working Group. Internet: http://tools.ietf.org/html/rfc3492.

Crystal, D. (1997): *English as a global language*. Cambridge: Cambridge University Press.

Daigle, L. (2004): *WHOIS Protocol Specification*. RFC-3912. IETF Network Working Group. Internet: http://tools.ietf.org/html/rfc3912.

Danet, B./Herring, S.C. (2007): *The multilingual Internet*. Oxford: Oxford University Press.

Faltstrom, P./Hoffman, P./Costello, A. (2003): *Internationalizing Domain Names in Applications* (*IDNA*). RFC-3490. Internet: http://tools.ietf.org/html/rfc3912.

Ferguson, C.A. (1959): Diglossia. In: *Word* 15, 325-340. Reprinted in: Ferguson, C.A. (1996): *Sociolinguistic perspectives: Papers on language in society, 1959-1994*. Ed. by Thom Huebner. Oxford: Oxford University Press, 25-39.

Friedman, B./Nissenbaum, H. (1996): Bias in computer systems. In: *ACM Transactions on Information Systems* 14 (3), 330-347.

Gillam, R. (2002): *Unicode demystified*. New York: Addison-Wesley.

Graddol, D. (1999): The decline of the native speaker. In: Graddol, D./Meinhof, U. (eds.): English in a changing world. In: *AILA Review* 13, 57-68.

Hardie, A. (2007): From legacy encodings to Unicode: the graphical and logical principles in the scripts of South Asia. In: *Language Resources and Evaluation* 41, 1-25.

Kahn, R. (2006): *Remarks from the opening session of the first Internet Governance Forum*. Athens, Greece. Internet: www.intgovforum.org/cms/IGF-OpeningSession-301006.txt.

Klensin, J. (2001): *Simple Mail Transfer Protocol*. RFC-2821. Internet: www.ietf.org/rfc/rfc2821.txt.

Loewis, M. van. (1997): *Internationalization and nationalization*. *Proceedings of the Sixth International Python Conference*, *San Jose, CA, October 1997*. Internet: www.python.org/workshops/1997-10/proceedings/loewis.html.

Mayer-Schönberger, V./Ziewitz, M. (2006): *Jefferson rebuffed: The United States and the future of Internet governance*. (= Kennedy School of Government Faculty Research Working Paper Series, RWP06-018). Cambridge, MA: Harvard Kennedy School of Government. Internet: http://ksgnotes1.harvard.edu/Research/wpaper.nsf/rwp/RWP06-018.

McEnery, A.M./Xiao, R.Z. (2005): Character encoding in corpus construction. In: Wynne, M. (ed.): *Developing Linguistic Corpora: A Guide to Good Practice*. Oxford: AHDS, 47-58.

Pargman, D./Palme, J. (2009): ASCII imperialism. In: Lampland, M./Star, S.L. (eds.): *Standards and their stories: How quantifying, classifying, and formalizing practices shape everyday life*. Ithaca, NY : Cornell University Press, 177-199.

Phillipson, R. (1992): *Linguistic imperialism*. Oxford: Oxford University Press.

Phillipson, R. (2003): *English-only Europe? Challenging language policy*. Oxford: Routledge.

Rader, R.W. (2001): *One history of DNS*. Internet: www.byte.org/one-history-of-dns.pdf.

Tomayko, J.E. (2002): Software as engineering. In: Hashagen, U./Keil-Slawik, R./Norberg, A. (eds.): *History of computing: software issues*. Berlin: Springer Verlag, 65-76.

Unicode (2011): *The Unicode Standard, Version 6.0.0*. Mountain View, CA: The Unicode Consortium. Internet: www.unicode.org/versions/Unicode6.0.0/.

Wright, S. (ed.) (2004): Multilingualism on the Internet. Themed Issue, *International Journal on Multicultural Societies* 6.1. Internet: www.unesco.org/shs/ijms.

8. Appendix: Keywords sets of several common markup and programming languages

HTML (91)	A, ABBR, ACRONYM, ADDRESS, APPLET, AREA, B, BASE, BASEFONT, BDO, BIG, BLOCKQUOTE, BODY, BR, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, H1, H2, H3, H4, H5, H6, HEAD, HR, HTML, I, IFRAME, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MAP, MENU, META, NOFRAMES, NOSCRIPT, OBJECT, OL, OPTGROUP, OPTION, P, PARAM, PRE, Q, S, SAMP, SCRIPT, SELECT, SMALL, SPAN, STRIKE, STRONG, STYLE, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TITLE, TR, TT, U, UL, VAR
(120)	abbr, accept-charset, accept, accesskey, action, align, alink, alt, archive, axis, background, bgcolor, border, cellpadding, cellspacing, char, charoff, charset, checked, cite, class, classid, clear, code, codebase, codetype, color, cols, colspan, compact, content, coords, data, datetime, declare, defer, dir, disabled, enctype, face, for, frame, frameborder, headers, height, href, hreflang, hspace, http-equiv, id, ismap, label, lang, language, link, longdesc, marginheight, marginwidth, maxlength, media, method, multiple, name, nohref, noresize, noshade, nowrap, object, onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onload, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreset, onselect, onsubmit, onunload, profile, prompt, readonly, rel, rev, rows, rowspan, rules, scheme, scope, scrolling, selected, shape, size, span, src, standby, start, style, summary, tabindex, target, text, title, type, usemap, valign

XML-DTD (22)	ELEMENT, DOCTYPE, IGNORE, INCLUDE, SYSTEM, PUBLIC, #PCDATA, ANY, EMPTY, CDATA, NMTOKEN, NMTOKENS, ID, IDREFS, ENTITY, ENTITIES, NOTATION, #REQUIRED, #IMPLIED, #FIXED, xml:space, xml:lang
Python (31)	and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield
Ruby (38)	alias, and, BEGIN, begin, break, case, class, def, defined, do, else, elsif, END, end, ensure, false, for, if, in, module, next, nil, not, or, redo, rescue, retry, return, self, super, then, true, undef, unless, until, when, while, yield
PHP (71)	Abstract, and, array(), as, break, case, catch, cfunction, class, clone, const, continue, declare, default , do, else, elseif, enddeclare, endfor, endforeach, endif, endswitch, endwhile, extends, final, for, foreach, function, global, goto, if, implements, interface, instanceof, namespace, old_function, or, private, protected, public, static, switch, throw, try, use, var, while, xor, __CLASS__, __DIR__, __FILE__, __LINE__, __FUNCTION__, __METHOD__, __NAMESPACE__, Language, constructs, die(), echo(), empty(), exit(), eval(), include(), include_once(), isset(), list(), require(), require_once(), return(), print(), unset()
Fortran (41)	assign, backspace, block data, call, close, common, continue, data, dimension, do, else, else if, end, endfile, endif, entry, equivalence, external, format, function, goto, if, implicit, inquire, intrinsic, open, parameter, pause, print, program, read, return, rewind, rewrite, save, stop, subroutine, then, write
C (31)	auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, static, struct, switch, typedef, union, unsigned, void, volatile, while
C++ (16)	bool, catch, class, delete, friend, inline, new, namespace, operator, private, protected, public, tempate, this, throw, try, template