INFORMATICA, 2007, Vol. 18, No. 3, 363–374
© 2007 *Institute of Mathematics and Informatics, Vilnius*

Internationalization of Compilers

Valentina DAGIENĖ
Vilnius University, Faculty of Mathematics and Informatics
Naugarduko 24, 03225 Vilnius, Lithuania
e-mail: dagiene@ktl.mii.lt

Rimgaudas LAUCIUS
Institute of Mathematics and Informatics
Akademijos 4, 08663 Vilnius, Lithuania
e-mail: rimga@ktl.mii.lt

Received: January 2007

Abstract. Internationalization of compilers and localization of programming languages is not a usual phenomenon yet; however, due to a rapid progress of software and programming technologies it is inevitable. The new versions of wide used programming systems already allow using the identifiers written in the native language, and partially supports Unicode standard, but still have many internationalization deficiencies.

The paper analyses the main elements of internationalization of compilers and their localization possibilities. According to contemporary standards, existing practices of software internationalization and tendencies there are given recommendations how compilers should be internationalized. The paper gives arguments of the importance of localization of lexical elements of the programming languages, and presents solutions that enable to solve the problems of portability of programs developed using localized compiler as well as problems of compiler's compatibility with other compilers.

Key words: localisation, internationalisation, design of compilers, translation, open source.

1. Introduction

Beginning with the last decade most of the software developers conceived the importance of the localization of their products. This is one of the most efficiently ways of expanding the software market and getting more revenues. Based on the statistics of the USA software developers, their localized software market exceeds 60% and is constantly growing. When localizing software the needs of a particular cultural divide are taken into account. This increases its usability and therefore people all over the world give a priority to it contrary to non-localized software (Esselink, 2002).

Software localization is a complex task. This process requires a contribution not only from a provider of localization services, but also from the software developer. The expenditure and success of localization strongly depends on the software internationalization level which is the prerogative of developer.

Localisation Industry Standards Association (LISA) defines internationalization as follows: “Internationalization is the process of generalizing a product so that it can handle multiple languages and cultural conventions without the need for re-design. Internationalization takes places at the level of program design and document development“.

The study on internationalisation that has been performed in Finland (Immonen and Sajaniemi, 2003) reveals that the majority of Finish software producers, who took participation in the survey, understand the importance of internationalization for their software however most of them restrict themselves only to establishing features for its text translation. Features for localization of the rest of software elements are established only by several developers and only in rare cases. The main reason is rather a high percent of investments necessary for internationalization, – it amount up to 10–20% of software production expenses.

During the interviews different developers pointed on to various reasons that determine that high percent of expenditure for internationalization. The main reasons, however, were named technical, such as: the lack of methods and standards for developing the internationalized software, the problems of character coding, and insufficient support of internationalization by programming tools, and so on (Immonen, Sajaniemi, 2003).

Software internationalization is part of its development process, so an internationalization of software development tools is a very important factor (Dagiene, 2006). If the tools are not internationalized then it is impossible to create internationalized software using them or otherwise this process requires additional investments. For example, it is obvious that a programmer will face difficulties in developing internationalized software, if the programming tools do not support usage of a multilingual text.

Most of researchers present internationalization process as additional to software development process and tries to improve it (Young, 2001; Sullivan, 2001). But most of the internationalization problems may be effectively solved only by internationalizing software development tools. By that may be achieved that development of internationalized software would be as natural process as that of non-internationalized.

The investigations revealed that the majority of contemporary programming tools are insufficiently internationalized. It could by partly explained by the fact that internationalization itself is rather a new phenomenon. The delayed interest in the internationalization of compilers, compared with the usual software, was determined by other factors as well. For instance, stagnation: programmers are frequently ill disposed towards any novelties (Trillo, 1999).

The greatest part of software is being developed by using compilers. The principal modern compiler consists of three main parts: 1) translator, 2) runtime library, and 3) linker (Fig. 1). A translator translates a program written in human readable programming language into program meant for linker. Linker uses this program and fragments of code prepared in advance and stored in runtime library to generate the final products, generally these are executable files or libraries.

In the paper, we shall overview the issues of translator’s internationalization, actually without touching issues of internationalization of runtime library and linker; namely, we are going to analyse these issues: data encoding, implementation of locale elements, internationalization of lexical elements.

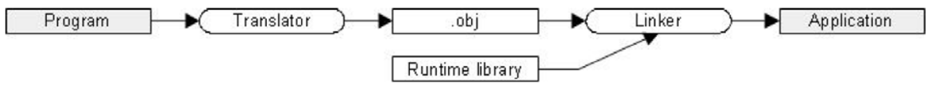

Fig. 1. Compilation process.

2. Data Encoding

Data encoding is one of the most important issues of compiler internationalization. The features of compiled program to process data properly depend on that as well as the possibility to input source code of program containing elements written in native language or even multilingual text. Improperly selected encoding method may even cause data loss during its encoding from one code table to other. Because of that the right choice is to use universal encoding method.

There are 6,912 known living languages in the world (Gordon, 2005) however most of software is developed in the English language. Based on statistics, the initial language of more than 80% of localization projects is English (Esselink, 2002). This is the reason of insufficient software internationalization as well. English language almost do not require internationalization, hence when designing software the needs of other people are often disregarded. It is often supposed to be the problem of localizers but not developers. Such attitude is wrong.

The most frequent error when creating software is an improper choice of 8 bit data encoding. That is indirectly determined because of use of 8 bit encoding by the most of the wide used compilers. It is possible to encode only 255 characters by 8 bits, which is sufficient for encoding script characters of the English and many other languages by using different code tables, but far from all languages (ISO 8859). There are scripts that use several hundreds and even thousands of characters (e.g., Chinese, Japanese). That is why the internationalized compiler should use universal method for data encoding.

Such universal method is proposed in Unicode standard. The Unicode standard was created and is further improved by the consortium under the same title name in 1991. It defines quite a large code table (including 111412 code units) which encompasses all the characters used in the world.

The Unicode standard base code table is equivalent to the ISO 10646 standard subset UCS-2. Differently than the ISO 10646 standard, which only describes the set of characters, the Unicode standard presents in addition, the rules and specifications related with the standard implementation, e.g., combined character sequence normalization, bidirectional text display specifications, and the like. Due to this, the Unicode standard became more wide known and used. It is also implemented by the majority of modern operating systems (OS).

The Unicode standard presents three character encoding methods: UTF-32, UTF-16, and UTF-8. The digit in the title of method means the number of bits allocated to express a code unit. It suffices only 32 bits in order to express all the Unicode characters by separate code units. Meanwhile, by using other methods part of characters is expressed by sequences of code units. It suffices 16 bits to express the characters present in the base multilingual plane of Unicode code table by separated code units. Characters beyond this limit are encoded using sequences of code units from surrogates' area. Most often there is no need to do that because characters beyond base plane are employed very rarely. 8 bits are sufficient to code only ASCII character, while the remaining ones are coded by sequences of 2–6 code units. By mutually comparing the Unicode methods with respect to implementation in software, one can find both merits and demerits, whereas in terms of internationalization they are equivalent.

With respect to compiler, data can be divided into two types: external and internal. The source code, configuration, messages, log files and like are considered as external data. The data used in the compiler memory, e.g., lexical elements, messages, and like, are regarded as internal data. In terms of internationalization it is not important which of the Unidode encoding methods is chosen for encoding the external data. When reading data it makes no difficulty to recode it. However, it is reasonable to choose a method for encoding external data coincident with that of internal data encoding. In that case, a syntax analyser will operate more efficiently.

In order to code internal data, it is expedient to select UTF-16 method because it is implemented by most modern OS APIs. Therefore using OS services there will be no need to recode them.

As an example it may be presented several contemporary programming systems such as *Visual Studio 2003 .NET*[1], *Delphi 2005 .NET*[2], *Java Studio 8*[3] that are using Unicode encoding already.

3. Implementation of Locale Elements

The subset of software user's environment dependent on the customs of language and culture is defined by a locale. Locales may be implemented in different ways. One can rely on generally accepted international standard – POSIX (IEEE Standard 1003, ISO/IEC 9945), C++ (ISO/IEC 14882), FDCC-set (ISO/IEC 14652), or de-facto standards (these models of locales are not formally standardized, but very important): *Windows*, *Java* locales. We should also mention here the standard (ISO/IEC 15897) for registration procedure of cultural elements. That defines the registration of locale elements described not only in a formal form (based on the POSIX or computer readable format, e.g., XML), but also in an informal way, by natural language (Dagienė, Laucius, 2004). But it is a bit obsolete. Unicode technical standard #35 "Locale Data Markup Language" is considered as more actual at this time. The data prepared according to this standard may be submitted to "Common Locale Data Repository" and is publicly available.

Standard models of locales define a generalized set of cultural elements. The main parts that are included by the majority of locales are:

[1]Visual Studio 2003 .NET, Windows, .NET Framework are the registered trademarks of Microsoft Corporation.

[2]Delphi 2005 .NET is a registered trademark of Borland International, Inc.

[3]Java Studio 8, Java are the registered trademarks of Sun Microsystems, Inc.

1. Language and country. In what language a user keeps a dialogue with software. The names of language and country and their translations into English.
2. Characters encoding, classification, and collation. Which characters are letters, numbers, or punctuation marks? Has a language got capital and lower-case letters and how are they mutually converted? What is collation order of characters?
3. Formats (of numbers, currency, date and time, addresses, telephone numbers, etc.). How are the integer and real numbers, money sums, dates, and time represented? In what order are addresses and telephone numbers written?
4. Calendars, time zone. Which calendar is used? Are the week days showed in it and in which way? Information on summer/winter time.
5. Measurement units, paper formats. Which measure units are used in measuring length, weight, sound, speed etc?

The key elements that identify a locale are language and country. For instance, despite that USA and United Kingdom use same language they have different locales, because their cultural conventions are different. A similar situation is in Finland where two different state languages (Finnish and Swedish) are used, so two different locales are needed. To identify locales in a computer a combination of language and country codes defined by the standards ISO 639 and ISO 3166 is usually used. RFC 3066 and XPG4 documents define such identification of locales and it is widely applied. Though, e.g., in the *Windows* locales are identified by special digital constants LCID.

A compiler as any other software also depends from a locale. It may use date, time, number and other formats when formatting messages. Messages may contain grammatically changeable parts (e.g., quantitative nouns). Differently from usual software compilers may include the code fragments of standard functions, operations, data types' realization and etc. that may be locale dependent as well. These code fragments is being included into produced binaries during compilation hence the internationalization of compiler may impact the internationalization of produced software as well. That makes the internationalization of compilers a bit harder in compare with the internationalization of usual software. It may be called a meta-internationalization in some way.

Information stored in standard locales is not always sufficient. For example, a great many of additional cultural dependent elements may be used in office programs: date, address, and other formats, templates of documents, and the like. In such cases, one has to look for other ways of realisation of additional elements (Laucius, Dagienė, 2004). It might be necessary also in those cases, if the program runs on various platforms, because most of them use different locale models.

4. Internationalization of Lexical Elements

The difference of internationalization of compilers in comparison with internationalization of usual software is that programming language syntax requires internationalization as well. Because of singularity of compilers' functional logic usual internationalization methods do not suit for its internationalization.

Lexical elements of the programming language may also be regarded as additional elements of locale. Though official descriptions of programming languages typically omit localization capabilities of lexical elements, however, often there is a provided possibility that they might be selected during the programming language implementation during compiler design. In such a case, lexical elements can be selected so that they would correspond to the language and culture of particular locale. In fact, that would be programming language localization (Grigas, 2000).

Localization of programming languages is not yet a usual phenomenon there exist very few internationalized compilers that allow it. One of the programming languages most known and used in the world which supposed to be localized is LOGO. It is used in general education schools in Lithuania for teaching algorithms students of lower grade. Localization gives an advantage to it against non-localized programming languages, because the vocabulary of a localized language is clearer and simpler, it can be easier memorized and learned. The elements of a program written in the native language are more natural and easier conceivable.

We seek similar aims localizing another programming system – Free Pascal. It is used for teaching algorithms in higher grades in Lithuanian schools of general education. Free Pascal compiler is not sufficiently internationalized hence some problems were faced in the process of its localization. Since it is open source compiler, we decided to internationalize it ourselves and investigated the ways of doing that. The investigation revealed a lot of useful information related with compilers' internationalization (Laucius, Dagienė, 2001).

There is an important relation between the human thinking and language. A man uses language in order to explicitly express what he has in mind. Analogously when developing program a programmer is also thinking in constructions and concepts of the language he is programming. That is why it is of utmost importance that these concepts (denoting various lexical elements) where as close to the native language as possible.

Language localization is especially important for the peoples who have different than Latin script writing system (e.g., Cyrillic, Arabic, etc.). In that case, to write lexical elements denoted by Latin characters they have to use additional, according to they locale, Latin input methods. Text typesetting problems may arise even for the peoples that use the Latin script writing system, but do not use all the letters of English alphabet, e.g., there no "w, q, x" letters in the Lithuanian alphabet (LST 1852).

The locale should include the following lexical elements: reserved words, operators, number, punctuation marks, and standard names. With a view to localize them properly, the compiler has to support the Unicode standard. For example, Unicode Standard Annex #31 describes syntax rules for formation of the identifiers of programming languages. Based on them identifiers can be formed from the Latin character as well as from other scripts characters (Cyrillic, Greek, Chinese, etc.) (Unicode, 2005).

Reserved words. In many programming languages English words or their abbreviations serve as basic words. There is an opinion that these should play a unification role among different programming languages and possibly they have not to be localized. However, as Grigas' research has shown, a variety of reserved words among different languages is very great indeed (only 3,3% of common reserved words were used in 13 programming languages, while 56% are used only in particular languages), so that this opinion has no ground (Grigas, 2000).

Internationalization of the reserved words is not complicated as they are only symbols denoting programming language syntax. Therefore, it suffices to implement a mechanism into a compiler that would dynamically load localized reserved words from a certain locale.

Operators. A part of operators in programming language are taken over from mathematics and are international. However, there are some exceptions, e.g., some peoples use different symbols of division and multiplication.

The majority of programming languages were created on the basis of ASCII code table that has only several principal symbols of operations. Therefore part of the symbols were encoded in pairs of characters (:=, <>, and so on) or were denoted by names (*div*, *or*, *and*, and so on). After internationalizing a compiler, there will appear an opportunity to proceed to natural operators' notation by mathematical symbols (Table 1). However, in order to avoid possible text typesetting problems, it is reasonable to implement mathematical symbols as synonyms to the former ones.

Numbers. Figures of digits just like script characters may differ in different writing systems (Table 2). Therefore their localization features should be foreseen as well.

Formats of writing numbers also pose some problems, e.g., punctuation mark of decimal fraction. For instance, in the USA, a period plays its role, while a comma is used in most of European peoples. In programming languages a comma typically performs the punctuation function of list elements. Therefore, the use of comma as a punctuation mark of a decimal fraction will result in ambiguity. For instance, the note 0,12 can be treated both as a decimal fraction and as a list of two numbers. The note 0, 12, however, is unambiguous, since there is an interval after a comma which is inadmissible in writing

Table 1

Examples of operators

Operation	Pascal operators	Mathematical operators
Assignement	:=	← (U+2190)
Ordinal division	Div	÷ (U+00f7), \ (U+0092)
Multiplication	*	· (U+2219), × (U+00d7)

Table 2

Examples of digits

Latin	Arabic	Thai
5	٥ (U+0665)	๕ (U+0E55)
7	٧ (U+0667)	๗ (U+0E57)
9	٩ (U+0669)	๙ (U+0E59)

Table 3

Examples of digits grouping

Nation	Examples	Remarks
British Germans	1,234,567,890.12 1.234.567.890,12	Groups by 3 digits. Groups are set off by various marks, dependent on a locale.
Swiss	1’234’567’890,12	
Lithuanian	1 234 567 890,12	
Chinese, Japanese	12 億 3456 万 7890.12 or 12 億 3,456 万 7,890.12	Groups by 4 digits, separated by ideographic marks. Sometimes even thousands may be set off.
Hindu	1,23,45,67,890.12	Groups by 3 digits in the first group, and in 2 in the next ones, separated with commas.

numbers. Thus this problem could be solved by extending the syntax of a programming language according to which the first note could be treated only as number (Grigas, 2000).

Grouping digits also gives rise to similar problems. Digits are grouped differently in various cultures. Various symbols may also play the part of a punctuation mark between groups. As usual, programming languages do not provide any features for grouping digits, despite that after expanding their syntax a little, it would not be difficult to introduce digit grouping.

Punctuation marks. Along with the punctuation marks for a decimal fraction and grouping, there exist other groups of punctuation marks that require internationalization (e.g., an interval may be basic, connective, ideographic, etc.). These marks are not numerous so their implementation is nor very complicated.

A part of punctuation marks in programming languages also play the role of lexical elements. For instance, parentheses serve for grouping of elements, comments, etc., apostrophes denote string constants. It should be noted that in different locales different marks may be used to denote the same. For example, commas may have various flourishes, they can be written upper or lower, etc. (e.g., “English”, “Lithuanian“) (Unicode, 2005).

Standard names. The standards of programming languages describe not only the rules of program syntax and semantics, but also they present a collection of means that facilitate their realisation, such as standard data types, constants, functions, and modifiers. Other than the reserved words, these are not fixed and can be described anew. Therefore it is possible to install rather a simple mechanism that would re-declare these names, by localized names.

It is a little more complicated to localize names of standard functions. In various compilers they are realised in a different way, so it is difficult to offer a universal mechanism for their internationalization. In most compilers, however, the standard functions are denoted by internal names, while the real names are only lexical symbols. In those cases, they can be internationalized rather easily.

Directives. The names of directives usually are not described in the standards of programming languages. They are meant for controlling the compilation process, so they are closely connected with the realisation of a particular compiler. Therefore the right to

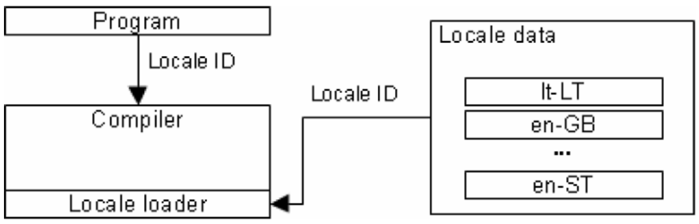

Fig. 2. Locale data of compiler.

choose their names belongs to a developer, and usually directives are described in the documentation of compiler. They are also very important and frequently used construction of programming language therefore they should be internationalized as well.

5. Framework of Internationalization of Lexical Elements

In order to allow complete localization of lexical elements, two possibilities should be foreseen during the internationalization of compiler: 1) to replace them and 2) to attribute synonyms to them. The possibility of using synonyms may solve possible text typesetting in some cases (e.g., in typesetting a synonym of a mathematical symbol that is missing in the computer keyboard).

Lexical elements are easily defined in a formal way; therefore their internationalization is not a complicated task. Lexical elements can be treated as customary formal data of locale. In this way, one can solve the problem of a portability of programs by loading data of different locales dynamically at runtime of compiler. So compiler will become independent from a particular locale and will be able to compile programs meant for various locales (e.g., indicating the program locale by directives or compiler switches). The figure below (Fig. 2) illustrates the translation process. According to the main platform locale or compiler switches, a compiler loads the data of a respective locale.

It is possible to retain the compatibility with previously developed programs by using a fictional locale (in the figure it is conditionally denotes as “en-ST”) that would include lexical elements corresponding to the standard functioning of the compiler, so as if the compiler were not expanded in this case as well.

In order to retain the compatibility with other compilers of the same programming language or merely to translate source code into another locale, one can employ a preprocessor that would translate source code between locales.

6. Conclusions

Software internationalization is part of its development process, so internationalization of software development tools is a very important factor to it. Internationalization and further localization of compilers can improve software development process and its internationalization level.

Internationalization of compiler differs from usual software internationalization because it additionally encompasses internationalization of lexical elements of programming language. The localization of lexical elements may cause problems of source code portability that can be solved by applying methods presented in this paper.

The most important task of the internationalization of compilers is the implementation of Unicode encoding method. The features of compiled program to process data properly depend on that as well as the possibility to input source code of program containing elements written in native language or even multilingual text. It also provides features for internationalization of lexical elements.

The localization of programming language gives an advantage to it against non-localized programming languages, because the vocabulary of a localized language is clearer and simpler, it can be easier memorized and learned. The programs written in the native language are more natural and easier conceivable.

References

Dagienė, V., R. Laucius (2004). Internationalization of open source software: framework and some issues. In T. Boyle, P. Oriogun, A. Pakstas (Eds.), *2nd International Conference Information Technology: Research and Education*, pp. 204–207.

Dagienė, V., G. Grigas (2006). Quantitative evaluation of the process of open source software localization. *Informatica*, **17**(1), 3–12.

Esselink, B. (2002). Localization and translation. *Computers and translation*. John Benjamins Publ. Comp., 67–87.

Gordon, R. G. (2005). *Ethnologue: Languages of the World*. 15th edition. Dallas: SIL International.

Grigas, G. (2000). Programavimo kalbų leksikos elementų analizė mokymo požiūriu. *Informatika*, **1**(35), 45–67.

Hall, B. (2005). Developing software with internationalization in mind. *MultiLingual Computing & Technology*, **16**(3), 10–13.

Hall, V., Hudson, R. (1997). *Software without Frontiers*. Chichester: Willey & Sons.

Immonen, J., J. Sajaniemi (2003). *Software Globalisation in Finland: A State-of-the-Practice Survey*. University of Joensuu.

Laucius, R. (2003). Lokalės, jų sandara ir ypatumai. *Informacinės technologijos*, Konferencijos pranešimų medžiaga. Kaunas, Technologija, **I**, 1–7.

Laucius, R., V. Dagienė (2001). "Free Pascal" panaudojimas informatikos kursui. *Lietuvos matematikos rinkinys*, **41**, 267–271.

Pat O'Sullivan (2001). *A Paradigm for Creating Multilingual Interfaces*. Dissertation, University of Limerick.

Trillo, N.G. (1999). The cultural component of designing and evaluating international user interfaces. In *Proceedings of the 32nd Hawaii International Conference on System Science*. Hawaii.

Unicode (2003). *The Unicode Standard*, Version 4.0. Boston, MA, Addison-Wesley.

Young, E. (2001). *A Framework for the Integration of Internationalization into the Software Development Process*. Dissertation, South Dakota University.

V. Dagienė is head of the Department of Informatics Methodology at the Institute of Mathematics and Informatics as well as professor at the Vilnius University. She has published over 100 scientific papers and the same number of methodical works, has written more than 50 textbooks in the field of informatics and ICT for high school (part of them is written together with co-authors).

She works in various expert groups and work groups, guides the activity of a Young Programmer's School, for many years, she has been organizing the Olympiads in Informatics among students. Recently she is engaged in localization of software and education programs, e-learning, and problem solving.

She is national representative of the Technical Committee of IFIP for Education (TC3), member of the Group for Informatics in Secondary Education (WG 3.1) and for Research (WG 3.3) of IFIP, member of the European Logo Scientific Committee, member of International Committee of Olympiads in Informatics. She is an Executive Editor of international journal "Informatics in Education".

R. Laucius is PhD student (finishing) in the Department of Informatics Methodology in Institute of Mathematics ans Informatics. His dissertation is on topic "Internationalization of compilers". The results of his dissertation have been published in 8 scientific papers. His research field is software and compilers internationalization and localization as well as programming methodology. He has developed the internationalized software for teaching programming that is used in Lithuanian schools. He is lecturer in Vilnius University. Most of the subjects he lectures are related with programming.

Transliatorių internacionalizacija

Valentina DAGIENĖ, Rimgaudas LAUCIUS

Transliatorių internacionalizavimas ir programavimo kalbų lokalizavimas dar nėra įprastas reiškinys, tačiau sparčios programinės įrangos ir programavimo technologijų pažangos dėka jis tampa vis labiau būtinas. Naujausios populiarios programavimo sistemų versijos (Delphi 8, Visual Studio 2003) leidžia naudoti nacionalinius žymenis, įvairiomis kalbomis užrašytus kintamųjų, procedūrų ir kt. vardus. Straipsnyje argumentuojama programavimo kalbų leksikos lokalizavimo svarba, nagrinėjama, kaip turėtų būti internacionalizuotas transliatorius, kad jį būtų galima lokalizuoti kuo mažesnėmis sąnaudois. Pateiktas spendimas leidžia spręsti lokalizuoto transliatoriaus bei juo sukurtų programų perkeliamumo ir suderinamumo su kitais transliatoriais problemas.