УДК 81'373.45
DOI 10.17223/19996195/32/1

PROGRAMMING LANGUAGES IN TRANSLATION – A TRANSLATOR'S GLANCE

Radoslaw Gajda

Abstract. The aim of this article is to find the answers to some questions referring to a programming language and especially the ones concerning the functioning of graphical elements of user interfaces, translating the comments in the source code of programmes, documentation comments and also the translation of the comments in markup code.
Keywords: Technical language; IT; software; language trap; problems with translation.

Introduction

The aim of this article is to find the answers to some questions referring to a programming language and especially the ones concerning the functioning of graphical elements of user interfaces, translating the comments in the source code of programmes, documentation comments and also the translation of the comments in markup code.

It is worth starting the dissertation concerning this topic with the definition of the term the programming language, according to the ways it is used in the specialist dictionaries.

According to the definition in Słownik Języka Polskiego (The Dictionary of the Polish Language) a programming language is a tool used to formulate programmes for computers, a sort of formal language whose syntax determines the rules of how a programme is written in the way which is both unambiguous and easy to analyse and it is the semantics which attributes the interpretation to the particular programmes and determines the results of the operations of a programme written in a programming language [1. P. 473].

A programming language is also «a system of signs adjusted to write programmes for digital computers» [Ibid].

Annalysis

Among many translators dealing with translations in the field of computer studies prevails the conviction that too little attention is paid to teaching specialist technical language while teaching foreign languages at academic level. While working on the texts referring to computer studies they encounter serious problems since they have not been confronted with technical texts either in the secondary school or at university so they are not able to use the previous experience in this area. An additional difficulty is caused by the fact that students still have not mastered Russian terminology in particular discipline including computer studies. Beginner students find understanding specialist texts difficult due to the lack of Russian terminology and generally due to the fact that their knowledge of some terms in general is not sufficient. To prove this, it is enough to say that while translation practices, it is even the text in students' native language that forms the hindrance which they cannot cross; as a result there is no way we could expect a proper translation which complies with all the rules.

As I mentioned in the introduction, this outline concerns some aspects of translating computer studies texts including the source code. At this point it is worth pointing out that in the literature of the subject specialist language is not fully independent from general one. The grammatical (referring to syntax or morphology) part of the model of a specialist language constitutes mostly a subset of the grammatical (referring to syntax or morphology) part of the module of general language [2. P. 21–22].

Nevertheless, there are certain syntactic or morphological phenomena which are typical for a particular specialist language and which are alien to general language.

The lexical part of the model of a specialist language comprises a set containing a part of lexemes belonging to the lexical part of a module of a general language and an external part of lexemes which do not belong to it.

Indeed, the grammatical (syntactic) part of the model of a programming language does not agree in any excerpt with grammatical (syntactic) part of the model of general language.

Both the programming language (as a feature of a particular man) and the specialist language (as a feature of a particular man) oppose functionally to other specialist languages and other programming languages [3. P. 129].

According to Małgorzata Kornacka both programming languages and the types of languages such as of Law or Economics can be called specialist languages. In such case the whole theory of specialist languages used so far needs to be revised. Programming languages as independent beings from the general language can be qualified as a specialist language. The type of languages of Law or Economics as the non-independent beings from the general language could be called specialist sublanguages. In such case the so far existing theory of specialist languages becomes the theory of specialist sublanguages [Ibid. P. 17].

As W. Zmarzer and J. Lukszyn state, two functions can be attributed to lexemes of programming languages: 1) instrumental function – as a professional tool, 2) didactic function – a function of organising the didactic process [4. P. 73].

However, the didactic function of terms differs from the didactic function of the lexemes of programming languages. Both specialist languages and programming languages are used in the didactic process which aim at using some language skills. While teaching foreign languages the language of grammar (or widely linguistics) is used. Mastering this language is to make learning foreign languages easier since it means acquiring the language in which the information about the studied language is provided.

To teach programming one uses for example the Pascal language. Acquiring this language is to facilitate learning other programming languages because it means mastering the skill of making algorithms.

Specialist languages, as opposed to programming languages, are also used in the didactic process, which aims at developing non-language skills.

As it is emphasized by Lukszyn some typical features of specialist languages include:

a) the openness of terminological lexicons to borrowings, first of all from the world languages;

b) the tendency to focus on classical languages as the source of words while creating new units;

c) the predilection for making terminological systems international;

d) the predisposition for conceptual transformations as a result of various interdisciplinary connections [2. P. 41].

According to Piwko programming languages constitute a very diverse and large group. There are many types of them, (ex. object-oriented, procedural, functional ones, etc. – however seldom do we find clear types, i.e. only purely object-oriented or only procedural ones, etc.) and each of them has its own features. Nevertheless, all of them have two features in common – each language provides some data which can be displayed on the screen (of a monitor or any other appliance or device) and in each of them one can use some comments or commentaries. In this part of the article some methods of distinguishing the above mentioned structures will be presented in order to attribute their localization.

Another things which have to be remembered are so called special characters. These are special sequences of characters, which compiler or interpreter treats in a special way. They are used most of all to insert into the string such characters which do not exist on the keyboard or which cannot be written directly into the contents of the string. It is the straight quotation marks (code 0022 from Unicode) that can be used as an example of such a character. It cannot be written directly into the string since then it would be treated as the ending of this string, ex.:

```
string x = "A text "in quotation marks"";
```

This definition of the variable is wrong and it would cause a mistake of compilation. Below you can find its corrected version with a use of special characters:

```
string x = "A text \"in quotation marks\"";
```

If the contents of the above variable was to be displayed on the screen, the result would be as follows:

```
A text "in quotation marks".
```

Special characters always start with the character of a backslash (\) which is followed by one more character or a few ones. For instance, we can often find the special character, so called a character of a new line, which as a name indicates means entering a new line. Let us take the below excerpt of a code in the language *C++* as an example:

```
cout <<"Welcome, \n John Smith";
```

Following the instruction we would find the displayed result:

```
Welcome,
John Smith
```

A translator working with a source code has to know special characters and know their usage in order to take them into consideration in an appropriate way in his or her translation. For instance, the special character of a straight quotation marks (\") in Russian translation would probably be replaced by directly written pair of characters «и».

There are also some programming languages, where in order to denote a character string one uses other characters, ex. in the language dBase it is possible to use square brackets – [ ].

It is also worth remembering that quotation marks can also be used for some other purposes rather than denote character strings, ex. in order to indicate the text value consisting of more than one word, such as for instance the name of the font.

When it comes to translating some names of the elements of the graphic interface of a user, translators usually do not have to deal with the source code of the given programme (unless they translate a book or some other text in the field of programming). Usually they receive a list of words and expressions which they are to translate into the target language.

In such lists you can often find the symbol & whose usage is not clear to lots of translators. This character is a programmers' way to designate so called accelerators, in other words the dedicated keys for the particular element of the interface.

If a letter is proceeded by the symbol &, the very same letter will be underlined in the ready programme and it means that this particular element of the interface can be activated simply by pressing just this key. For example, if in the toolbar of a Firefox Internet browser a letter P is underlined in the menu's *File*, it means that if activate the toolbar and press the key P, the contents of the menu *File* will be displayed with various options with own accelerators.

It is worth remembering what the symbol & is used for in such contexts and to use it in accordance with logic or the guidelines of the translation provider.

Equally important as to what appears on the screen is the translation of the comments or commentaries which are found in the source code. Generally speaking, they can be classified in two ways. One can distinguish one line and multiline / block comments and using other criteria: «regular» comments and documentary comments. One line comments are these which are to be found fully in one line and multiline can take up any number of lines.

On the other hand, documentary comments are especially marked blocks of text which after writing the programme are picked up by a special algorithm and written in the form of a document (ex. HTML) which serve as a documentation of the particular programme.

There are many ways of denoting comments and all of them are characterized by the fact that they require the use of a special character or a special sequence of characters indicating the beginning of a comment and in case of a multiline one, also pointing out its end.

Probably most often found type of comments are so called comments in the style of the *C* Language. In this language (and also among others in the languages *C++, PHP, C#* and *Java*) to indicate one line comments the sequence of marks // is used. Below you can find an example of a code with the comments which could be written in this language:

```
// Integer variable named x containing the value 10.
int x = 10;
```

A sample of the translation into the Russian language:

```
// Целая переменная, именуемая x, содержит ценность, равную 10.
Int x = 10;
```

The block comment in the language *C* (and also in among others *C++, PHP, C#* and *Java*) is indicated with the sequence of marks /* (the beginning) and */ (the end):

```
/* Integer variable
named x containing
the value 10 */
int x = 10
```

A sample of the translation into the Russian language:

```
/* Целая переменная
именуемая x
содержит ценность, равную 10*/
int x = 10;
```

In other programming languages different ways of denoting comments can be found. For instance, in the language *PHP* apart from the two mentioned types of comments there is also a possibility to mark the one line comments with the help of the mark #. Similar case concerns the language *Ruby* and for example the configuration file of the server Apache*httpd.conf.* However, for instance, in the language *Visual Basic* one line comment is indicated with the help of the mark of the apostrophe (').

The documentary comments differ from the ordinary ones most of all in the fact of being taken from the source code by a special programme and processed into the form of a document which is used in the function of the documentation of the given programme. Therefore, they are often more complex and marked in a special way in order to let them be distinguished from the regular comments.

An example can be provided from the language *Java* where the documentary comments are indicated with the use of the sequence of marks/** (the beginning) and */ (the end). Visual Basic, on the other hand, uses three single quotation marks ("'), and the end the comment ensues together with the end of the line. In the language *C#* three slashes (///) are used with similar ending as in *Visual Basic*.

By the term extensible markup languages we usually understand the languages HTML / XHTML and XML. In reality the majority of all the translations of this type concerns just the documents in these languages. However, it is worth remembering that there are more extensible markup languages. Many of them have been formed based on the language *XML*. It means they have their own precisely specified syntax, yet they are governed by the rules from *XML*. The most important difference between the languages XHTML and HTML is also connected with this. The first one comes directly from the language XML, whereas the latter one from the language SGML. In reality both of them are very similar to each other because, to put it short XML is a simplified version of the language SGML invented in order to enable an easy exchange of data between various programmes operating on different platforms.

Extensible markup languages most often mean to us the various versions of the languages HTML and XML. Recognising what is to be translated in documents of this type is very easy and when we use such support software aiding the translation of the Trados type we do not have to think about it at all. These programmes automatically choose for translation only what is necessary.

Each extensible markup language, as the name indicates consists of a determinate (HTML) or undeterminate (XML), it means any sets of elements. Most of elements consist of two marks- an opening and the closing ones (the elements consisting of one mark do not have the contents in the same way as the elements consisting of two mark-ups, yet they can still have some attributes. They come as follow:

<name-element's attributes'>element's contents</element's name>.

What is to be translated is most of all what is found between the opening mark and the closing one, so in the quoted scheme it would be the words «the contents».

However, sometimes it may be necessary to translate the content of some attributes. It refers mostly, but not only to the language HTML and its versions. For instance, below you can find a mark inserting a picture called holidazs.jpg onto a web page:

<img src="holidazs.jpg"alt="Photo of me and my cousin at the countryside"/>

Despite the fact that the above element does not have the real content, it contains an attribute alt. It is used to determine the text which is to be displayed instead of the picture, if it is going to be unavailable for some reason (ex. when the owner deletes it by mistake). This text has to be translated into the Russian language:

<img src="holidazs.jpg"alt="Я и мой племянник в деревне"/>

Another place where one can find a text to be translated is the content of some so called meta names, and to be precise – meta characters <meta name=”keywords”/> and <meta name=”description”/>. The first of them is applied to denote key words, which used to be implemented by the Internet search engines while sorting out the websites. (At present due to dishonest use of this character by the Internet web-designers it has lost its practical usage, nevertheless is still in use.) The latter one, however, contains the description of the Internet website and this description is displayed by the Google search engine while showing the results of the search.

These characters look like:

<meta name=”keywords” content=”list of comma-separated keywords”/>

<meta name=”description content=”description of the page”/>

The above characters could be translated into the Russian language in this way:

<meta name=”keywords” content=”список ключевых слов, разделенных запятой”/>

<meta name=”description” content=”описание”/>

The so far given examples refer to the language HTML only. In case of the language XML the situation seems to be a bit more complicated since in this language there are no constant fixed set of elements. The person creating a document in this layout himself defines a set of elements and their usage. Therefore, it is not always obvious the contents of which attributes ought to be translated and which not. It is worth pointing out that the rules governing the formation of the elements and translating their contents are the same as in case of the language HTML [5. P. 59].

As far as it concerns the comments un extensible markup languages the situation is less complicated. To mark them a sequence of marks <!— (beginning) and —>(the end) is used, ex.:

<!—A paragraph-->

<p>A paragraph of text< /p>

An example of a translation into Russian:

<!--Абзац-->

<p>Абзац с текстом</p>

To conclude this part, the comments in the language CSS are worth mentioning. This language does not belong to extensible markup languages, nevertheless it is inseparably connected with them. It is used to denote the style and the layout of the elements on the websites. This language uses the earlier described comments in the style of the language

C–/*...*/:

/* Rule that turns the color of all paragraphs red*/

p {color: red}

A sample translation into Russian:

/* Правило, которое окрашивает цвет всех параграфов в красный цвет*/

p {color: red}

Additionally, CSS can contain so called generated content which is inserted onto the website automatically by a search engine. For example, in order to place a text *Once upon a time* into the beginning of each paragraph it is possible to write such a rule of CSS (using so called dot AND element – :before):

p:before { content:"Once upon a time"}

Its translation can be as follows:

p:before {content: "Однажды"}

Dot AND element :after has the similar usage as : before, with this difference that it is always placed after the element which it refers to. It is worth remembering these two dot AND elements since CSS code is almost always written in a different file than the website itself and added only with the help of a special instruction. If it is so and the dot AND elements are used in this code it will be necessary to find them in an additional file and to translate them properly.

Let us notice that in practice the code which is correct from syntax point of view does not have to be correct semantically and it is the key element for the translator. Of course, an analogy to natural languages can be found here.

It is the semantics of the particular programming language that defines precisely the meaning of each symbol and its function in the programme. The semantics is mostly defined in words since it is impossible to express most of its issues in any formal way. Some semantic mistakes can be noticed immediately in the first steps of processing the code of the programme, ex. an effort to refer to a non-existing function of the programme. Nevertheless, some others can appear later during the performance.

A translator of similar texts has to be aware of the fact that in English texts it is possible to find often the constructions of the type: *x must be equal to or less than y*, so literally it means:

x равен или меньше чем y.

However, this literal translation is awkward and not correct from grammar point of view since you cannot say *равен чем.* Is it possible to translate it in a better way? Of course, it is. It is necessary to use a simple procedure.

Instead of translating literally, it is possible to convey the meaning of the sentence and rather than to translate it, it is better to write simply in Russian what the author had meant.

In the sentence given above it is known that x can be equal to y and that it can be also less than it. If x may be less than y or equal to it, we can use the reverse statement and write that

x cannot be more than. In this way we get a neat translation of the sentence of the text: x must be equal to or less than y: не может быть больше чем y.

Below I am going to provide more real examples of the sentences containing this construction:

The number of elements in the initializer must be equal to or less than the number of elements specified for the array (The C++ Programming Language, 4th edition, Bjarne Stroustrup).

Число элементов в инициализаторе должно равняться или быть меньше, чем количество элементов, заданных для матрицы.

If the total amount you received for the item is equal to or less than the total cost of buying and selling it, then you had no income (www. nbcnews.com).

Если общая полученная вами сумма за отдельный предмет равняется или менее общей стоимости его покупки и продажи, в этом случае вы не получаете дохода.

The stock's required return must be equal to or less than 5%.

Необходимый по основному капиталу доход должен равняться или быть менее 5%.

In texts in the English language, especially in the field of IT, it is possible to encounter the expression optimize away. Since there is no such concise expression in the Russian language, it may cause some problems to translators. Before we proceed to some suggestions referring to how to translate it, let us first analyse what it exactly means to optimize away. The meaning of each element of the analysed expression seems to be obvious: optimise – оптимизировить [Ibid. P. 95], away – oddalić [Ibid. P. 96]. So it refers to removing something away in order to optimize it. Everything becomes clear after studying a couple of examples:

A volatile specifier basically tells the complier not to optimize away apparently redundant reads and writes (The C++ Programming Language, 4th edition, Bjarne Stroustrup).

Can a C++ complier optimize away code when dealing with pointers? (stackoverflow.com).

Will Perl 6 optimize away useless assignments? (www.perlmonks.org).

It is possible to guess the meaning of the expression to optimize away based on the first from the above given examples: *not to optimize apparently redundant reads and writes.* It can be concluded that the interesting us activity causes getting rid of «apparently redundant reads and writes», therefore to optimize away means to eliminate, to get rid of by means of optimization.

So, the above given example can be translated into the Russian language as follows:

Переменный признак, в основном, указывает компилятору не оптимизировать сразу несомненно чрезмерное считывание информации и её запись.

Может ли компилятор C++ сразу оптимизировать код, когда работает с указателем?

Сразу ли язык Perl 6 оптимизирует бесполезные отчисления?

Based on the above given considerations and examples, it is possible to conclude that the good equivalents for the English expression to optimize away is: *удалить путём оптимизации, исключить при оптимизации, оставить (не удалять) при оптимизации.*

Of course the analyse provided here refers to IT context and precisely the programming one. Each language uses a particular set of data and therefore it is necessary to divide the data into particular types defined according to their properties and the operations which in which they are to be implemented.

English programming texts are characterised by the fact that everywhere unusual expressions, words or even whole sentences can be found. One of them is an English expression, or a term – *one –past-last element.*

At first, a person who is not a programming specialist might think that this must be some kind of an awkward expression or even a programmer's mistake. However, it is a fully correct expression which has to be included into purely programming terminology.

In all programming languages, so called data structures are used in order to store the information, ex. integers (integral numbers) or floating-point character strings or more complicated types of data.

Since particular portions of information kept in these structures are just called elements so it is already known what the last part of the described expression means – it is the element of structure of the data, ex. an integer or a floating-point or anything else.

The elements stored in the data structure (in reality the way of storing may be more complex, but here I avoid these issues for the purpose of simplicity, because they do not have any significant role this context) may be arranged in continuous way, it means there might be neither space between the particular elements nor the elements taken from other structures [5. P. 152].

Assuming that data structure is shown as a one – dimensional array, the designate of the expression *one-past-last* element is the space/place behind the last element of the continuous structure.

To sum it up, it is possible to translate the expression *one-past-last* element as место за последним элементом (в структуре данных).

Summary

As Lukszyn claims the basic function of a specialist language is an instrumental one. According to him, the functions of a specialist text analysed from the point of view of a sender and / or a receiver of a message form a fundamental triad. These are:

1) the functions of collecting (the data);
2) the function of conveying (the message);
3) the function of knowledge improvement.

As it is emphasized by the author of *Uniwersalia tekstów specjalistycznych*, a successful implementation of the first function means, among others, the necessity to re-create the proper theoretical context in order to distinguish the quantum of knowledge [3. P. 44].

The function of conveying the specialist information assumes its updating with the reference to the target receiver of the message. The knowledge improvement function stems out of the nature of professional communication which fundamental task is to be in search for new formulas of knowledge [Ibid]. As the author writes further, it is the coherence of a specialist text which enables re-creating the net of semantic connections which are typical for its conceptual elements. The semantic net is expressed in the hypersyntactic organisation of a text. The semantic cohesion of a specialist text is shown in such phenomena as:

1) subject progression;
2) individual sentences being synsemantic;
3) multiple repetitions of the same terms;
4) anaphora and cataphora [Ibid].

The presented here outlook of several difficulties in translating some programming languages, according to the author, enables showing the mistakes made by students and translators as opposed to the results of what is taught at foreign language courses at universities both as optional and linguistic ones [6–28]. It seems also that the remarks and conclusions reached while discussing particular mistakes may be useful while changing and reforming the curricula of the translation subjects of linguistic studies. I do not advocate coming back to the Grammar Translation Method, nevertheless it is necessary to support the concept of popularizing the need to teach technical (IT) translation since this ability will come useful to translators in their practical activities.

References

1. ***Bańko M.*** Inny słownik języka polskiego. Warszawa, 2000.
2. ***Grucza F.*** O językach specjalistycznych jako pewnych składnikach rzeczywistych języków ludzkich. Warszawa, 1994.
3. ***Kornacka M.*** Języki programowania jako języki specjalistyczne [w] Języki specjalistyczne. Problemy technolingwistyki. Warszawa, 2002.
4. ***Zmarzer W., Lukszyn J.*** Teoretyczne podstawy terminologii. Warszawa, 2001.
5. ***Piwko Ł.*** Komputer w pracy tłumacza. Warszawa, 2012.
6. ***Bojar B.*** Słownik encyklopedyczny terminologii języków i systemów informacyjno-wyszukiwawczych. Warszawa, 1993.
7. ***Chomsky N.*** Syntactic structures. Berlin, 2002.
8. ***Lukszyn J.*** Tezaurus Terminologii Translatorycznej. Warszawa, 1993.
9. ***Milowski M.*** Przekład tekstów informatycznych na język polski. Warszawa, 2012.
10. ***Sielicki W.*** Trudności leksykalne i gramatyczne w tłumaczeniu tekstów technicznych z języka rosyjskiego na polski i z języka polskiego na rosyjski, [w:] Interferencja w procesie przekładu językowego. Wrocław, 1974.
11. ***Szulc A.*** Słownik dydaktyki języków obcych. Warszawa, 1997.
12. ***Voellnagl A.*** Jak nie tłumaczyć tekstów technicznych. Warszawa, 1973.
13. ***Abramsky S., Jung A.*** Domain Theory // Handbook of Logic in Computer Science. 1994. Vol. III.
14. ***Rojas Raúl et al.*** Plankalkül: The First High-Level Programming Language and its Implementation. Institut für Informatik, Freie Universität Berlin, Technical Report B-3, 2000.
15. URL: http://terminologia.pl
16. ***Антонова Т.В.*** К вопросу о классификации явлений языковой асимметрии (на примере русского и английского языков) // Язык и культура. 2015. № 3 (31).
17. ***Ваулина Е.Ю.*** Информатика. М., 2005.
18. ***Герасимова Н.И.*** Прагматическая адаптация IT-текстов информационного дискурса при переводе. Тюмень, 2015.
19. ***Ефремова Т.Ф.*** Новый словарь русского языка. Толково-словообразовательный. М., 2001. Т. I–II.
20. ***Комиссаров В.Н.*** Теория перевода (лингвистические аспекты). М., 1990.
21. ***Кузнецов С.А.*** Большой толковый словарь русского языка. СПб., 2008.
22. ***Мильруд Р.П., Карамнов А.С.*** Подходы к классификации английских модальных глаголов // Язык и культура. 2008. № 3.
23. ***Нелюбин Л.Л.*** Толковый переводческий словарь. М., 2003.
24. ***Раренко М.Б.*** Основные понятия переводоведения (Отечественный опыт) : терминологический словарь-справочник. М., 2010.
25. ***Сьерра К., Бейтс Б.*** Изучаем Java. М., 2012.
26. ***Соснина Е.П.*** Введение в прикладную лингвистику. Ульяновск, 2012.
27. ***Фримен Э., Батес В., Сьерра К.*** Паттерны проектирования. М., 2011.
28. ***Prowit M., Szarski J.*** Большой русско-польский политехнический словарь. Warszawa, 1968.

Radoslaw Gajda, Doctor of Humanities in Linguistics, Faculty of Philology, Pedagogical University of Cracow (Cracow, Poland). E-mail: radgajda@wp.pl

DOI 10.17223/19996195/32/1