INFORMATICA, 2010, Vol. 21, No. 3, 321–338

An Approach to Formalize Metainformation of Software Localizable Resources

Valentina DAGIENĖ, Tatjana JEVSIKOVA
Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: dagiene@ktl.mii.lt, tatjanaj@ktl.mii.lt

Received: May 2010; accepted: August 2010

Abstract. Software localization is one of important tasks to insure a successful computer user experience. The paper discusses how localization of the software dialog text can be accelerated and how to raise the quality of software product localization. We also discuss the main features and common structure of localizable software resources, their formats and preparation for localization. As a result, we suggest applying a modified formalism of attribute grammars to describe localizable resources, taking the graphical user interface as a basic grammar structure, localizable strings and their parts as terminal symbols, and using the attributes to add important metainformation and context to the resources. The main principles of creation of such attribute grammars are presented.

Keywords: software localization, software internationalization, localizable resources, metainformation of localizable resources, user interface adaptation, attribute grammars.

1. Introduction

The concept of *software localization* is defined differently by different authors. In this paper, we will stick to the localization as a process of modifying software to be suitable for a particular cultural and linguistic environment. The success of this process depends directly on how software and its resources are prepared for localization, i.e., internationalized.

The two main parts of software localization, identified in most research literature on the topic (e.g., Esselink, 2000; O'Sullivan, 2001; Yang, 2007) are: (1) software adaptation to the target locale (adjustment of character encoding, number formats, date and time formats, document templates, etc.); (2) translation and adaptation of user interface (menu items, button labels, check boxes, error messages, etc., including online help and documentation).

The problems of software adaptation to the target locale are usually solved by means of formal locale definitions (e.g., ISO/IEC 15897, 1999; ISO/IEC 14652, 2004; ISO/IEC 9945-2, 2003; Unicode, 2007); they have also been discussed in detail in the papers (Jevsikova, 2006; Dagienė, Jevsikova, 2009). The translation and adaptation of graphical user interface is more complicated due to a large amount of the user interface text (called "text strings", or just "strings" in this context) and the lack of context of text

strings. As a consequence, the localization quality is obtained going through iterations of the user interface text translation, testing, and correction. Existing methods of machine translation (e.g., Sepesy Maucec, Brest, 2010) can be applied to prepare draft version of long texts (e.g., software documentation) translation, but specifics of user interface text strings (short text or phrases without context) do not let us prepare an accurate translation in an automated way.

Our aim is to find a solution, how to provide a localizer with contextual information on localizable resources, and, as a result, to help raise the quality of localizations and reduce localization testing expenditure. Therefore, in this paper, the ways of separation of user interface text strings and their representation formats are analyzed (Sections 2 and 3), and an approach to include metainformation on the context of text strings, based on a modified attribute grammar, is proposed (Section 4).

2. Structure of Localizable Resources

One of the main steps during software preparation for localization is separation of the localizable resources from the software source code, i.e., externalization of all texts, graphics, videos, sounds, and locale-specific parameters used in the program. Most of the large software producers create their own methods of separation of such resources from the program source code. Open source software development initiatives have their own adopted methods. However, all the existing methods present localizable resources without the context of their usage in the graphical interface of software, or with slight and not extensive contextual information. The process of development, separation and delivery of localizable resources is shown in Fig. 1.

During Step 1 (Fig. 1) the context of all the texts and other resources is known. After the separation of resources from the source code, the context (or its major part) is lost. In Step 3 the separated resource files are delivered to localizers. They do not receive information on the context of the resources. A very important part of contextual information is lost between Step 2 and Step 3. The separated localizable resources are delivered in the default format of the resources separation method, or transformed into another (e.g., simple textual or intermediate) format.

As a result of the lack of context of user interface text strings, the percent of the text strings translated does not correspond to the real localization work completed. Some models are proposed that allow us to evaluate localization work done in relation with the

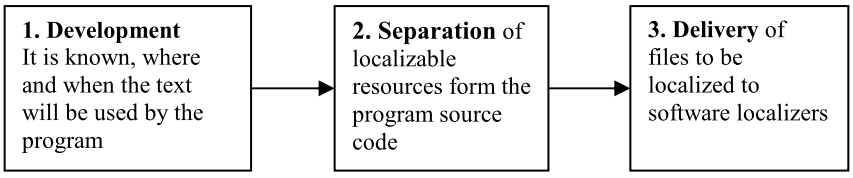

Fig. 1. Software preparation for localization: a scheme of the main stages.

percent of strings translated, e.g., 90% of the translated strings mean that only about 42% of localization work is done (Dagienė, Grigas, 2006).

A detailed analysis of the main wideused localizable resource separation methods and formats (RC, RESX, GNU Gettext, Java Resource Bundles, Mozilla, XLIFF, and PHP) has shown that, despite the variety of separation methods and formats existing today, most of them present textual localizable resources in a similar way: a simple database of key-value pairs (key is an identifier of a text string; value is a text string to be shown on the screen or to specify a preference of the program). Additional information can be added through localization comments. Exceptions are RC, RESX dialog section formats, and XLIFF format.

RC and RESX formats do have some means to present contextual information on dialog and menu sections, but do not present any contextual information on the string section.

XLIFF format has many advantages, comparing with any other format. However, it inherits all the shortcomings of other formats, because it is used as an intermediate format, to which data is converted from other existing formats. So, the main contextual information is not present in fact. XLIFF is a universal format not only for interface strings, but also for long texts with enough context (e.g., documentation) to translate. Therefore, it has a lot of tools to handle formatting and other information that is not so important for software interface strings.

GNU Gettext is the only format of all before analyzed ones that brings tools to handle the forms of words, written after the number.

3. Context in Localizable Resources

Formats of localizable resources are similar in the presentation of general textual localizable resources as a set of text string identifiers and text strings pairs. Text strings here are not only texts and messages, shown on the screen during program's runtime, but also values of some parameters, e.g., font, character encoding names, dialog box sizes, adaptation of which can make the functional effect on the program.

Localizers, working with textual localizable resources, face the following main problems:

1. They deal with separated words or phrases without the context of use in the graphical user interface of the program.
2. The context of a localizable text is obtained while running the program and searching for a particular text string, or examining the program source code (if the software license allows doing it).
3. A part of dialog texts can be noticed only in rare and exceptional situations (e.g., errors, interaction with other programs), that are very difficult or impossible to model in practice.
4. Fusional languages (languages in which one form of a morpheme can simultaneously encode several meanings; Payne, 1997) face a lot of problems in choosing

the correct form, part of speech, etc., deciding from a short English phrase without the context of use (most of software today is localized from English). For example, *File*, noun and verb, are written in the same way, but in Lithuanian it is *failas* (noun), *įtraukti* [į aplanką] (verb); *login* is *prisijungti* (verb), *prisijungimas*, *prisijungimo vardas* (noun). Another problem is terminology variations, slang and metaphors.

The analysis of software programs (web browsers, email clients, virtual learning environments, etc.) has shown that more than 50% of textual localizable resources are strings of 1 to 4 words (it means that they will probably lack context, important for localization). Localization would be accelerated if the context (linguistic, as well as user interface) was provided.

Some contextual information can be included into localization comments, folders and localizable resource file names, using the existing formats of localizable resources. Unfortunately, providing localization comments, naming folders and files depend on a software developer, it is not a systematized way of providing contextual information. Therefore, an approach, aimed to include contextual information into localizable resources and help to solve the issues mentioned above, is presented in the next section.

4. An Approach to Formalize Metainformation of Localizable Resources Using Attribute Grammar

The approach, proposed in this paper, is based on attribute grammar $AG = \langle G, A, R\rangle$ formalism, consisting of three components:

1. Context-free grammar G.
2. Finite set of attributes A.
3. Finite set of semantic rules R.
   Here $G = \langle N, T, P, S\rangle$:
   1. N is a finite set of non-terminal symbols;
   2. T is a finite set of terminal symbols;
   3. P is a finite set of grammar productions $X \rightarrow \alpha$, where $X \in N$, $\alpha \in (N \cup T)^*$ (set of all strings, comprised of non-terminal and terminal symbols, including the empty string ε).
   4. S $(S \in N)$ is a starting non-terminal symbol.

Such attribute grammar formalism has been proposed by Knuth (1968) four decades ago for specifying and implementing the (static) semantic aspects of programming languages, but remains still often used in many fields where relations among structured information play a central role. Some examples include general software engineering, databases, distributed programming, logic programming, static programming, visual programming, and natural language interface. As far as authors of this paper know, attribute grammars have not been used before to formalize contextual information of software localizable resources.

The main steps of the algorithm of the approach as well as the main aspects of attribute grammar formalism modification are presented below.

4.1. *Main Aspects of Attribute Grammar Preparation*

Formalization of localizable resources metainformation is started by describing a context-free grammar, according to the main principles:

1. Context-free grammar G is written for a particular program, taking into account the graphical user interface structure of a program and relating its elements with the corresponding localizable strings.
2. The grammar has two main parts, based on the usage of non-terminal and terminal symbols: representation of the structure of software graphical user elements, and representation of localizable strings and their structure.
3. Usage of non-terminal symbols:
   3.1. A non-terminal for each graphical user interface element (control), according to the specifics of the program.
   3.2. A non-terminal for a whole string from localizable resources.
   3.3. If a resource string has a parameter(s) inside, a non-terminal is used to address each parameter. The decision to do so is based on the result of investigation of localizable resources of software that about 10% of localizable strings have one or more parameters inside. Usage of a parameter (variable) is a potential source of errors for most of fusional languages. The production with a parameter symbol on the left-hand side has one or more non-terminals on the right-hand side, representing resource strings, planned to be used instead of a parameter, or ε (empty string), if the value of the parameter is obtained dynamically during the program runtime.
   3.4. If the menu element or another control uses the text, composed of several resource strings, then in the grammar derivation tree these strings should appear alongside, introducing separate non-terminals for the control and each string.
4. Terminal symbols are resource strings or resource string segments. If a string doesn't have parameters, then all the string is a terminal symbol, if there are some parameters, then segments between the parameters are taken as terminal symbols.

The next steps are extension of a context-free grammar to the attribute grammar:

5. Grammar symbols are augmented with attributes, which are used to present information, important from the localization point of view. Selection of attributes is based on the analysis of localization errors and features of the language. The list of common attributes is presented in the next section.
6. The rules to calculate the values of attributes (semantic rules) are defined.

The software graphical user interface strings are related to the interface elements (controls). We distinguish the common cases, how localizable text strings can be shown in the graphical user interface. This affects the choice of grammar symbols (Steps 1–4, described above). The elements are presented graphically as fragments of the grammar derivation tree. The main cases are the following:

1. A whole string, shown on a simple control. A case, where the string has no parameters inside, is presented in Fig. 2a. There is one string segment F, node C

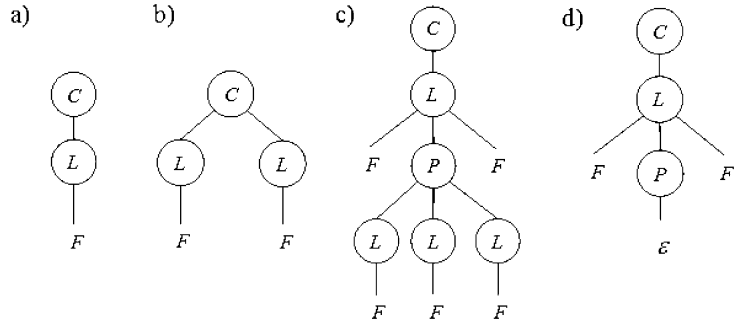

Fig. 2. Grammar derivation tree fragments, according to various cases of text string display in controls.

corresponds to the control non-terminal symbol, node L is a whole localizable text string non-terminal symbol.

2. Compound strings. Each string has its own name in localizable resources, i.e., the strings are presented separately in the resources, but shown together in the graphical user interface. Despite that such a method of using compound strings causes many problems during localization and is unacceptable (it can tolerate making errors when localizing to some languages, when it is needed to adjust grammatical forms of words or phrases, presented in different stings), it is quite usual in localization practice. A case where two composed strings are shown in the same control is presented in Fig. 2b.
3. Localizable string, having one or more parameters inside. There are several cases:
   a. A string from the localizable resources is used instead of the parameter during the program runtime (e.g., one string from the possible group of strings, meant for use instead of the parameter, depending on the situation). We assume that a string, used as a value of the parameter, has no parameters inside. Then, a group of strings to be used as values of the parameter corresponds to a non-terminal P(Fig. 2c).
   b. A value, not existing in the localizable resources, is used instead of the parameter during the program runtime. The value is dynamic, e.g., the number of objects created by the user, user name and surname, taken from the registration database. Then, an attribute is needed to bring more information to the localizer: what type of data is the parameter value. Fig. 2d shows a case where there is one parameter with a dynamically used value inside.
4. A group of several strings from the localizable resources is shown on a control, depending on the context arising during the program runtime. Such a case is sometimes used when software developers try to avoid using parameters inside the localizable strings: a separate string is created for each case of the context. This case is included into the grammar in the same way, as case 2 (Fig. 2b), therefore, additional information whether the strings are compound or dynamically used is presented by means of attributes.

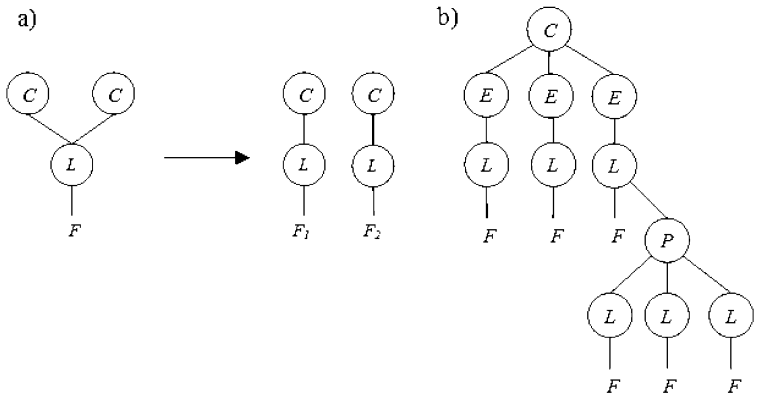

Fig.3. Grammar derivation tree fragments: (a) the same string is used in several controls, and (b) complex control.

5. A string from the localizable resources is used in several controls (e.g., as a button label and as an item of a context menu). When including such a case into the grammar, the string is duplicated (Fig. 3a), the attributes are compared. If the attributes of duplicated strings match, then one instance of a string can be kept in the localizable resources. If the attributes do not match, it means that the string should be localized in one way for one control, and in another way for another control. In this way, we can detect internationalization bugs during grammar preparation. Such a bug can be fixed including more copies of the string into localizable resources. In this case, creation of a grammar can also help minimize the number of string instances in localizable resources. Sometimes software developers try to avoid the bug mentioned above by including into resources too many instances of the same string, for each different control to be used in. While analysing the attributes of the grammar, we can decide whether we need an instance of a string in a particular case, or not.
6. A complex control (i.e., a control which shows more than one string, according to its purpose, e.g., drop-down or any other list). Dealing with any simple element E of a complex control C, one of the cases, discussed above can be used. Below we present a case where a complex control has three simple elements (E), one of which has a string with a parameter (Fig. 3b). Non-terminal L corresponding to the whole string from localizable resources, is introduced keeping the same grammar preparation principle, because theoretically more than one strings can be used in one simple element (E), e.g., as alternative strings.

If software uses command keys and access keys for the controls (e.g., menu, button commands), then it is reasonable to change that keys during localization, so that they fit the localized command name (e.g., *Print*, access key P, in Lithuanian localization would be *Spausdinti*, access key S). Our research has shown that about 17% of all localizable strings of web browser "Mozilla Firefox" and email client "Mozilla Thunderbird" are command keys and access keys. Therefore, the grammar should include symbols for the access key and command key, if any.

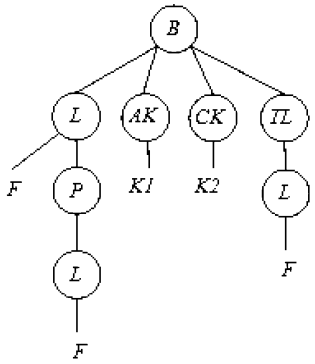

Fig. 4. Derivation tree fragment, corresponding to a button with an access key, command key, and a tooltip.

User interface controls may also have tooltips to explain commands in more detail than the name, displayed on the control does. Therefore, if there are any tooltips, it is reasonable to include the corresponding symbol into the attribute grammar. Below, a fragment of the grammar derivation tree, corresponding to a button control, is presented, where a button B has a name (L), made of two strings taken from the localizable resources (one of the strings is included through the parameter P), an access key (*AK*), a command key (*CK*), and a tooltip (*TL*) (Fig. 4).

Here, B corresponds to a button, L is the whole text string to be shown on the button (a name of button command), *AK* is an access key (*K1* is an actual letter, marking that key), *CK* is a command key (*K2* is an actual symbol, marking that key), *TL* is a tooltip, P is a parameter, F is a terminal symbol (string segment, some times coinciding with a whole string).

Figure 5 shows a fragment of the grammar derivation tree for a hypothetical software dialog box.

The greyed area corresponds to a part of the grammar graphical user interface. Another part is a string part. Grammar symbols, written in bold, correspond to the terminal symbols (localizable strings, e.g., “Archive”, “Web pages”, “Name:”). One of such strings has a parameter (P).

4.2. *The Main Attributes*

In order to formalize metainformation of localizable resources, attribute grammars have been augmented with new tools:

- A notion of the primary attribute has been introduced. The set of attributes of a grammar can be expressed as $A(X) = S(A) \cup I(A) \cup E(A)$. Here $S(A)$ is a set of synthesized attributes, $I(A)$ is a set of inherited attributes, and $E(A)$ is a set of primary attributes. Primary attributes are assigned not only to terminal symbols, but to non-terminal symbols as well.
- The values of primary attributes are provided externally, e.g., using the attribute questionnaire which is filled by the software developer or experienced localizer.

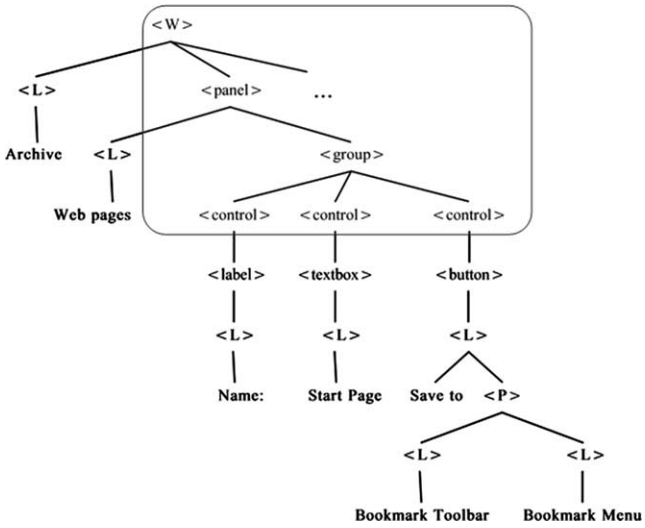

Fig. 5. Grammar structure of localizable resources: GUI and strings parts.

- The scope of attributes passing through the attributed derivation tree has been controlled, limiting it to the surrounding of a particular node.

All the attributes are expressed independently of the language by means available in the English language, so that the author of software is able to assign the values. Thus, most of the attributes and their values are common to fusional alphabetic languages.

The list of some attributes, used to extend the context-free grammar, is presented below.

4.2.1. *Noun/Verb Phrase*
This attribute is used to avoid an error arising from the same spelling of English nouns and verbs. The attribute is especially important to short strings. For example, the same phrase *Open file* in the menu context would have a verb form, but in a dialog box, the title would have the noun form in Lithuanian. When assigned automatically, the attribute is recommendatory.

4.2.2. *Controlling Word of a Phrase*
This attribute marks the word that controls the forms of other words of a phrase. In fusional languages it can help select the correct case of surrounding words.

4.2.3. *String Description*
The attribute is provided for strings, having special characters, abbreviations, unclear semantics or rare error messages.

4.2.4. *First Letter (Capital or Small)*
In English, capital letters are usually used in the middle of the phrases and sentences, e.g., months, day names, titles. In other languages, e.g., in Lithuanian, only the proper

nouns, abbreviations in the middle of a phrase are usually written in a capital letter. In other cases, a small letter is used.

4.2.5. *Form*
A word form, expressed by the question it answers to, e.g., *who*, *what*, *whom*, *where*, *when*, or a preposition: *to*, *from*, *in*. *Who* and *what* questions are divided into two groups: *subject* and *object*.

4.2.6. *Term*
The attribute is designed to specify the meaning of homonymous terms, marking the number of a meaning according to the agreed dictionary. For example, the term *key* has at least four meanings in Lithuanian: (1) *klavišas*, (2) *raktas*, (3) *kodas*, (4) *šifras*, etc. The dictionary can be a glossary, designed for a particular program's localization needs, or a common dictionary of computer terms.

4.2.7. *Internal Function*
A link to the internal function, implementing a particular command. In some cases it is needed to better understand the meaning of the command.

4.2.8. *String Identifier*
An identifier of a string in a resource system. It is used as an auxiliary attribute to keep a set of strings to be substituted instead of the parameter, compound strings or alternative strings.

4.2.9. *Width or Height*
An attribute, used to define the width or height of a dialog box or its control. It is used to calculate whether there is enough place to display the translated string.

4.2.10. *Indicator of a Compound String*
Several strings, used to compose a single user interface string, are included in the grammar tree alongside. Since the compound strings and alternative strings have the same formal description, the attribute is used to indicate whether the strings are composed. The attribute is also used to calculate the length of the composed string.

4.2.11. *Data Type*
If a string has a parameter, it is useful to know what type of data will be written instead of it during the program runtime. The attribute will help to choose the forms of words surrounding the parameter, e.g., if the number of an object is inserted instead of the parameter, then it is necessary to adjust the forms of a noun, written after the parameter; if the parameter is a person's name, then it is necessary to foresee a change in the forms and gender of surrounding words. It is also an indicator for software developers to include a component to adjust such a change in forms.

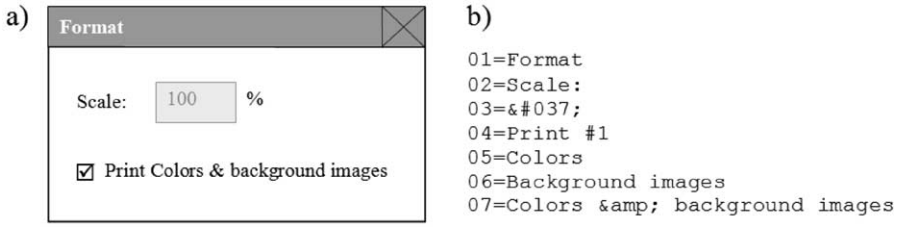

Fig. 6. Fragment of a dialog box (a) and the corresponding localizable strings (b).

4.2.12. *Tracking of Access Keys and Command Keys*
This group of attributes is designed to keep control of access and command keys, changed during the localization: check of duplicated keys in the scope.

The next step is a definition of semantic rules (functions, depending on the values of other attributes or values, submitted externally). A part of attributes is calculated; another part of attributes is submitted externally, e.g., via a questionnaire. Semantic information on localizable resources is spread over the entire attributed derivation tree, but not collected in the attributes of a root symbol of a tree.

4.3. *An Example*

In this section, we provide a small example of formalization of metainformation of localizable resources of a dialog box fragment (Fig. 6a) and a corresponding set of dialog strings (Fig. 6b). This example does not include all types of contextual information and attributes, but lets us demonstrate, how the attributes can help to choose the right grammatical forms in localizable strings.

The corresponding context-free grammar is presented below.

$G_W = \langle N_W, T_W, P_W, \text{W\_fragment}\rangle$, where

$N_W = \{\langle\text{W\_fragment}\rangle, \langle\text{control}\rangle, \langle\text{caption}\rangle, \langle\text{text\_box}\rangle, \langle\text{check\_box}\rangle, \langle\text{L}\rangle, \langle\text{P}\rangle\}$ is a set of non-terminal symbols;

$T_W = \{\text{F}\}$, F $\in$ {Format; Scale:; %; Print; Colors; Background images; Colors & background image} is a set of terminal symbols (% represents a percent sign in localizable resources);

P_W is a set of productions:

$P_W = \{\langle\text{W\_fragment}\rangle \rightarrow \langle\text{L}\rangle\langle\text{control}\rangle\{\langle\text{control}\rangle\}$

$\langle\text{control}\rangle \rightarrow \langle\text{caption}\rangle$

$\langle\text{control}\rangle \rightarrow \langle\text{text\_box}\rangle$

$\langle\text{control}\rangle \rightarrow \langle\text{check\_box}\rangle$

$\langle\text{caption}\rangle \rightarrow \langle\text{L}\rangle\{\langle\text{L}\rangle\}$

$\langle\text{text\_box}\rangle \rightarrow \langle\text{L}\rangle\{\langle\text{L}\rangle\} \mid \varepsilon$

$\langle\text{check\_box}\rangle \rightarrow \langle\text{L}\rangle\{\langle\text{L}\rangle\}$

$\langle\text{L}\rangle \rightarrow (F|\langle\text{P}\rangle)\ \{(F|\langle\text{P}\ \rangle)\}$

$\langle\text{P}\rangle \rightarrow \langle\text{L}\rangle\{\langle\text{L}\rangle\} \mid \varepsilon$

$\langle\text{L}\rangle \rightarrow F\}$;

$\langle$W_fragment$\rangle$ is a starting non-terminal symbol.

Table 1
Attribute list, used as an example

Attribute name	Attribute description/corresponding section of the paper	Attribute type	Attribute data type
desc	Section 4.2.3	↑←	Text string
noun_verb	Section 4.2.1	↓←	Text string
c_type	Type of a control, e.g., "button"	↓←	Text string
d_type	Section 4.2.11	↓←	Text string
order	Display order of dialog controls (names of several controls can form a single phrase or sentence)	↓←	Number
form	Section 4.2.5	↓←	Text string
id	String identifier in a resource system (auxiliary attribute)	↑←	Text string
idlist	List of identifiers of related strings (e.g., possible parameter values)	↑	Text string array
letter	Section 4.2.4	↑←	Boolean
term	Section 4.2.6	←	Text string
proper	Indicates, whether a noun is a proper name or not	←	Boolean
ctrl_word	Section 4.2.2	↓←	Text string

Names of non-terminals in productions are written in brackets $\langle\ \rangle$ to distinguish them from non-terminal simbols.

The next step is to assign attributes (a list of attributes is presented in Table 1). ↑ marks a synthesized attribute, ↓ means that the attribute is inherited, ← marks a primary attribute, provided externally, e.g., by the developer using a questionnaire. Arrow combinations mean that the attribute is provided externally, and then is passed up/down the grammar derivation tree.

The attribute grammar for the chosen example (a fragment of dialog box) is presented in Table 2. P_n in Table 2 corresponds to the grammar production number in the first column, the second column presents a production (written in bold) and semantic rules, written in pseudo code.

In the right-hand side of the productions we use EBNF syntax, presenting not predefined number of symbol occurrences (e.g., $\langle$L$\rangle\{\langle$L$\rangle\}$). Therefore, we should agree on how we mark attributes of any occurrence of the symbols.

In semantic rules, we use the attribute names presented in Table 1, after the names of grammar symbols (without brackets), separating them by a full stop, e.g., *L.letter*. An index in the attribute name means ith occurrence of the symbol in the production, e.g., L_i.order ($i = 1, 2, \ldots, n$, where n is number of symbol occurrences in the production). The statement $X_i.a = b$ means that attribute a of all the occurences of symbol X in the procuction, must be assigned value b.

The values of external attributes, e.g., *term*, are provided in a questionnaire (a fragment of the questionnaire is presented in Table 3. In Table 3 we use $P_{i.j}$ to mark the productions: production number $i = 1, 2, \ldots, 10$, and production application number

Table 2
Attribute grammar for a dialog box fragment

Production No.	Production and semantic rules
P_1	⟨**W_fragment**⟩ → ⟨ **L**⟩⟨**control**⟩{⟨**control**⟩}
	W_fragment.desc = L.desc
	L.noun_verb = NOUN
	L.letter = 1
P_2	⟨**control**⟩ → ⟨**caption**⟩
	caption.order = control.order
P_3	⟨**control**⟩→ ⟨**text_box**⟩
	text_box.order = control.order
P_4	⟨**control**⟩ → ⟨**check_box**⟩
	check_box.order = control.order
P_5	⟨**caption**⟩→ ⟨**L**⟩{⟨**L**⟩}
	L_i.c_type = "caption"
	L_i.order = caption.order
P_6	⟨**text_box**⟩ → ⟨**L**⟩{⟨**L**⟩} | ε
	L_i.c_type = "text box"
	L_i.order = text_box.order
P_7	⟨**check_box**⟩ → ⟨**L**⟩{⟨**L**⟩}
	L_i.c_type = "check box"
	L_i.order = check_box.order
P_8	⟨**L**⟩→ (**F** | ⟨**P** ⟩) **{(F** | ⟨**P**⟩)}
	P_i.ctrl_word = L.ctrl_word
	P_i.noun_verb = L.noun_verb
	L.letter:
	if L.order $>$ 1 and L.proper = 0
	then L.letter = 0
	else L.letter = 1
	F_i.id = L.id
	F_i.c_type = L.c_type
	F_i.c_type = L.c_type
P_9	⟨**P**⟩ → ⟨ **L**⟩{⟨**L**⟩} | ε
	P.letter:
	if P.proper = 1
	then P.letter = 1
	else P.letter = 0
	P.idlist:
	add(P.idlist, T_i.id)
	L_i.d_type = P.d_type
	L_i.form = P.form
P_{10}	⟨**L**⟩→ **F**
	F.id = F.id
	F.noun_verb = L.noun_verb
	F.form = L.form

Table 3
A fragment of a questionnaire, used to assign the values of external attributes

Production	Attribute name	Question	Answer/ options	Value examples, comments
$P_{1.1}$	control$_i$.order	Which is the order number of a control? $1, 2, \ldots, n$ if strings of the corresponding controls are related (form a single phrase/sentence), 0 if strings are not related.		0 1 2
...	...	...	...	...
$P_{8.j}$	T.noun_verb	Is the text string a verb phrase (VERB) or a noun phrase (NOUN)? Use value NA if impossible to specify.	VERB NOUN NA	
$P_{8.j}$	T.ctrl_word	Which word is a controlling word of a string? Use value NA if impossible to specify.		Print NA images
$P_{8.j}$	T.proper	Is the text of a string a proper name, an abbreviation, or starts with a proper name or abbreviation?	Yes No	
$P_{8.j}$	T.id	Please provide a string identifier.		01 02
$P_{8.j}$	S$_i$.term	If a string/string segment has an ambiguous term, please provide the number of its meaning. If one string has more than one ambiguous term, then provide meaning values, separating them by a semicolon. If there is no ambiguous term in a string, leave the field empty.		format (1) scale (2) certificate (2); key (4)
...	...	...	...	...

$j = 1, 2, \ldots, n$ to assign attribute values to every application of the production, n is the number of production i applications.

The corresponding attributed derivation tree with actual attribute values is presented in Fig. 7.

Dealing with the real software localizable resources, the approach presented here can be applied in the following steps:

1. A software component (or a whole program) is split into a set of relatively autonomous parts, corresponding to the sets of graphical user interface related elements. The size of components is not essential, the main aspect is that localizable strings of the component are useful to analyze together.

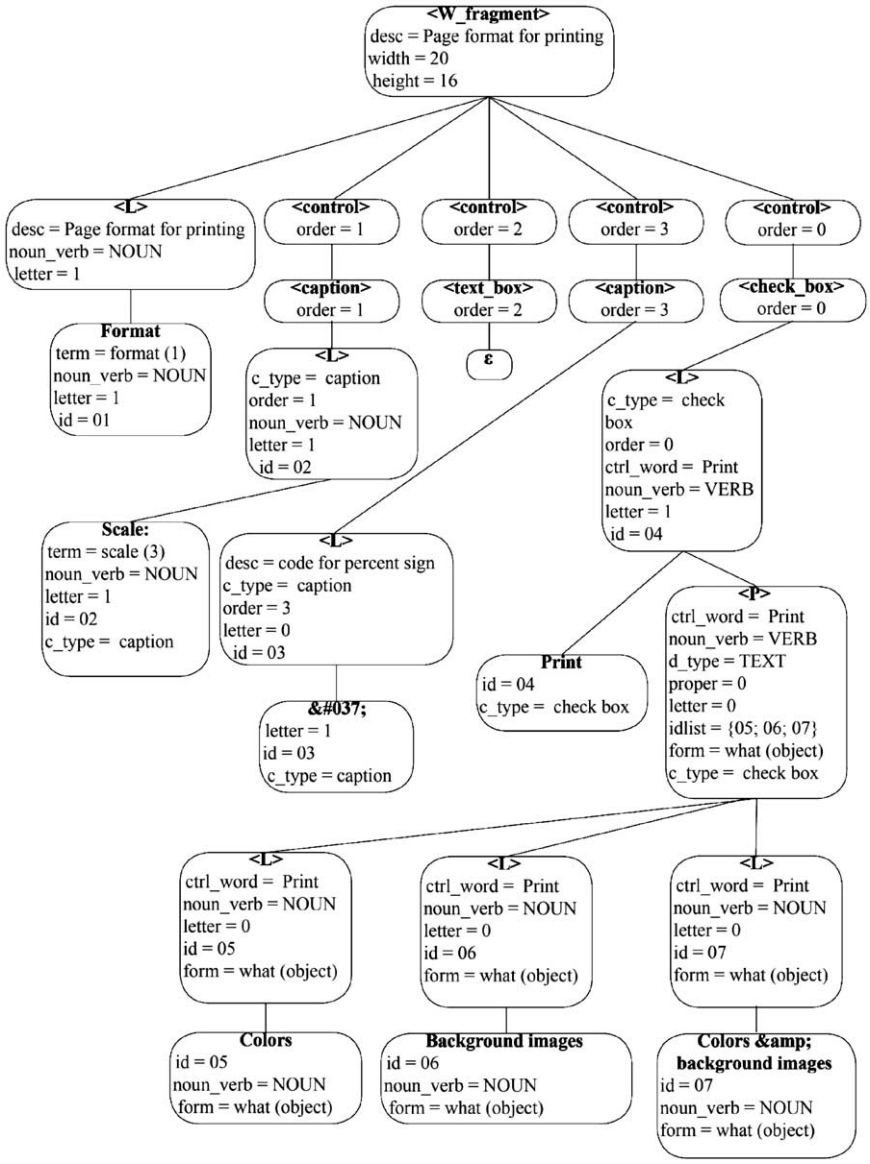

Fig. 7. Attributed derivation tree of localizable resources of the example.

2. A context-free grammar is prepared for each component or part of a component, as described in Section 4.1.
3. Every context-free grammar is extended into an attribute grammar, assigning attributes to each grammar symbol and defining semantic rules. The attribute list and semantic rules are prepared according to the software specifics.
4. Attribute grammars are combined into a whole.
5. Using the prepared attribute grammar, localizable resources are translated, and the results written into files.

5. Conclusions

The analysis of existing localizable resources formats has shown that there are not enough means to provide metainformation on the localizable resources that would help to choose the correct translation of strings (usually, short texts) in the localization and prepare localized products of a high quality. Therefore, an approach to include metainformation into localizable resources has been proposed. The approach presented is not tied to any particular technology (it can be considered as a matter of implementation). The main novel aspects of the approach are as follows:

1. It extends attribute grammars by three main novelties: (a) attribute grammars are augmented with a new attribute type; (b) computed attributes are complemented by some attributes, entered from outside; (c) the attribute context scope is controlled.
2. It presents localizable resources in a systematized way and helps to notice internationalization errors in the early software development (or even specification) stage.
3. It includes into localizable resources information that describes their context and is useful to raise the quality of localization: the place where the resource string will appear on the screen, relations between the related strings, and semantic information of each text string.
4. The approach is suitable not only for static user interface strings, but also for dynamic strings (formed during program runtime) as well.

As far as the attributes are expressed by the means not related to a specific language, the approach, presented here, will be suitable to most fusional European languages.

References

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

Dagienė, V., Jevsikova, T. (2009). Cultural elements in internet software localization. In: Lenca, P., Brézillon, P., Coppin, G. (Guest Eds.) *Revue d'intelligence artificielle. Human-Centered Processes – Current Trends*, Vol. 23, No. 4/2009. Hermes–Lavoisier, pp. 485–501.

Esselink, B. (2000). *A Practical Guide to Localization*. Benjamins.

ISO/IEC 15897:1999 (1999). *Information Technology – Procedures for Registration of Cultural Elements*.

ISO/IEC 9945-2:2003 (2003). *Information Technology – Portable Operating System Interface (POSIX)*. Part 2, System Interfaces.

ISO/IEC 14652:2004 (2004). *Information Technology – Specification Method for Cultural Conventions*.

Jevsikova, T. (2006). Internationalization and localization of web-based learning environment. In: R. Mittermeir (Ed.) *Informatics Education – the Bridge Between Using and Understanding Computers. Proc. ISSEP 2006, Lecture Notes in Computer Science*, Vol. 4226. Springer, Berlin, pp. 310–319.

Knuth, D.E. (1968). Semantics of context-free languages. *Math. Syst. Theory*, 2(2), 127–145.

O'Sullivan, P. A. (2001). *Paradigm for Creating Multilingual Interfaces*. Doctoral dissertation. University of Limerick.

Payne, T. E. (1997). *Describing Morphosyntax: A Guide for Field Linguists*. Cambridge University Press, Cambridge.

Sepesy Maucec, M., Brest, J. (2010). Reduction of morpho-syntactic features in statistical machine translation of highly inflective language. *Informatica*, 21(1), 95–116.

Unicode, Inc. (2007). *Unicode CLDR Project: Common Locale Data Repository*. http://unicode.org/cldr/ [accessed on 2010-07-30].

Yang, Y.X. (2007). Extending the user experience to localized products. In: Aykin, N. (Ed.) *Usability and Internationalization, pt 2, Proc. Global and Local User Interfaces. Lecture Notes in Computer Science*, Vol. 4560. Springer, Berlin, pp. 285–292.

V. Dagienė is the head of the Department of Informatics Methodology at the Institute of Mathematics and Informatics as well as a professor at the Vilnius University. She has published over 100 scientific papers and the same number of methodological works, has written more than 50 textbooks in the field of informatics and ICT for high schools (part of them is written together with co-authors). She has been working in various expert groups and work groups, guiding the activity of a Young Programmer's School, for many years, she has been organizing the Olympiads in Informatics among students. Valentina Dagienė has also been engaged in localization of software and educational programs, e-learning, and problem solving. She is a national representative of the Technical Committee of IFIP for Education (TC3), a member of the Group for Informatics in Secondary Education (WG 3.1) and for Research (WG 3.3) of IFIP, a member of the European Logo Scientific Committee, and a member of the International Committee of Olympiads in Informatics. She is an Executive Editor of international journals "Informatics in Education" and "Olympiads in Informaticsy".

T. Jevsikova is a researcher in the Department of Informatics Methodology at the Institute of Mathematics and Informatics as well as a lecturer at the Vilnius University. She received her PhD in computer science from Vytautas Magnus University and Institute of Mathematics and Informatics. Her main research interests include software localization, cultural aspects of human-computer interaction, e-learning, and standards. She is the author (or a co-author) of more than 15 scientific papers, several methodological books and dictionaries of computer science terms.

Programinės įrangos lokalizuojamųjų išteklių metainformacijos formalizavimas

Valentina DAGIENĖ, Tatjana JEVSIKOVA

Programinės įrangos lokalizavimas – vienas svarbesnių veiksnių kompiuterių taikymo srityje. Straipsnyje nagrinėjama, kaip galima būtų paspartinti programinės įrangos dialogų tekstų vertimą ir pagerinti lokalizacijų kokybę. Siūloma pasiremti atributinėmis gramatikomis ir jomis aprašyti lokalizuojamuosius išteklius, per atributus įtraukiant lokalizavimo požiūriu naudingą kontekstinę informaciją. Aptariamas programinės įrangos išteklių parengimas lokalizavimui, lokalizuojamųjų išteklių struktūra ir ypatumai. Pateikiami lokalizuojamųjų išteklių formaliosios gramatikos sudarymo principai.