Computer Science Journal of Moldova, vol.**3**, no.1(7), 1995

Meta–generation of syntax–oriented editors

N.Shvets K.Chebotar

Abstract

A method for automatic generation of syntax-oriented editors (SOE) for high level programming languages is presented. It is based on a special template definition metalanguage. The SOE functional environment including the operations with source files and internal representation of the programs in form of abstract syntax tree was implemented as an independent modular structure. As a result of target language metadescription processing the SOE for this language is generated by a special preprocessor. Depending on user's experience level (novice, advanced, etc.), generation of various SOE for the same language by changing a level of descriptions of phrases and templates is also possible.

Introduction

Syntax–oriented editor (SOE) is a tool for program development based on hierarchical structure of the high–level programming languages (HLPL) [1–3]. Its design pursues a few goals. Primarily, its purpose is to increase the users labour productivity and to guarantee a minimal level of errors when typing the text of a program. Secondly, it may be used for tutoring if designed properly. There are some related aspects, too.

Both main goals are achieved by automatic expansion the HLPL metaconcepts marked by the user and due to the presence of operations on structured texts and operations delivering an additional information about the language to the users who are not familiar with it.

The most important thing is that all these operations do not depend on the language factually and that the minimal level of errors

can be regulated.By the appropriate organization of the editor even graphical languages can be treated, too [3]. All this is achieved easily if the information necessary for SOE functioning may be extracted immediately (and automatically) from the common syntax description (metasyntax) of the language. Under these conditions we get the SOE generator which gives possibility to work uniformly with any HLPL in the editor environment when it is provided with an algorithm processing the metadescriptions mentioned.

1 The HLPL preprocessing

The aim of preprocessing is to build informational tables controlling the SOE, from the target language definition in terms of a special metalanguage. The latter is based on BNF (Backus normal forms) and is called the template definition language (TDL). Besides information which is characteristic for BNF, the metalanguage includes some additional data, namely:

- quantitative data for quick and convenient generation and use of the tables
- metaconcepts characteristics
- data used to visualize the program being generated.

Metaconcepts characteristics are:

- type (template is an intermediate concept, phrase is a terminal one)
- kind (simple template, template–option, templates–lists of two sorts).

The terminal metaconcepts possessing simple structure are being typed, as a rule, by the user on the display. Their level may be controlled by means of metalanguage. After processing it we can have different tables for the same programming language, controlling the

speed of the program generation and minimal level of errors, simultaneously. So, if a concrete language and a level of phrases are fixed, the tables obtained may be considered as parameters and then the simple SOE becomes a parameterized one.

The syntactic tables being a result of preprocessing, are represented by files. One of them is a set of records containing full information about the structure of each template and relations between the templates (for quick search). The other is a set of strings representing metaconcepts names and separators. One more file formed as part of preprocessing contains texts specifying the structure of the standard functions calls and standard identifiers. These files, besides providing the correct functioning of SOE, are used to supply a user–novice with additional information about the language structure (help–services).

The structure of TDL is the following:

⟨language_metadescription⟩ ::= ⟨number_of_metaconcepts⟩
⟨metaconcept_description⟩
⟨number_of_metaconcepts⟩ ::= ⟨integer⟩
⟨metaconcept_description⟩ ::= ⟨numeric_information⟩
⟨metaconcept_name⟩
⟨metaconcept_structure⟩
⟨numeric_information⟩ ::= ⟨template_numeric_description⟩
⟨screen_layout_descriptions⟩
⟨numeric_template_description⟩ ::= ⟨number_of_alternatives⟩
⟨number_of_components⟩
⟨number_of_sons⟩
⟨number_of_screen_layout_descriptions⟩
⟨number_of_alternatives⟩ ::= ⟨integer⟩
⟨number_of_components⟩ ::= ⟨integer⟩
⟨number_of_sons⟩ ::= ⟨integer⟩
⟨number_of_screen_layout_descriptions⟩ ::= ⟨integer⟩
⟨screen_layout_descriptions⟩ ::= ⟨template_layout⟩... ⟨template_layout⟩
⟨template_layout⟩ ::= ⟨component_layout⟩... ⟨component_layout⟩
⟨component_layout⟩ ::= ⟨line_shift⟩⟨column_shift⟩
⟨line_shift⟩ ::= 0 | 1
⟨column_shift⟩ ::= 0 | ⟨integer⟩

⟨metaconcept_structure⟩ ::= ⟨component_description⟩...
... ⟨component_description⟩
⟨component_description⟩ ::= ⟨component_type⟩⟨component_kind⟩
⟨component_type⟩ ::= ⟨simple_component⟩| ⟨option⟩| ⟨list⟩
⟨simple_component⟩ ::= (
⟨option⟩ ::= [
⟨list⟩ ::= #
⟨component_kind⟩ ::= ⟨keyword⟩| ⟨phrase⟩| ⟨template⟩
⟨keyword⟩ ::= K ⟨string_of_keywords_and_separators⟩
⟨phrase⟩ ::= F ⟨metaconcept_name⟩
⟨template ::= T ⟨metaconcept_name⟩
⟨metaconcept_name⟩ ::= ⟨⟨identifier⟩⟩

2 SOE functioning

The work done by the SOE is the following:

1. operations on files such as
   - viewing text file
   - reducing it to the abstract syntax tree
   - creating a new file for program generation
   - calling an old file containing the result of the previous editing session
   - formatting the texts represented in free format
2. setting the mode of SOE functioning:
   - autosave period selection
   - saving history of the session
   - setting the depth of the stack of operations
3. structured program generation/editing
4. unstructured program editing
5. quitting the editor under the following requests:
   - to save the intermediate result generation
   - to generate the program in terms of target language

- to confirm the exit or to return to the editor environment.

As all systems of the kind, the SOE–program is menu–driven. The main menu and its submenues used in SOE correspond to the operations mentioned. After setting the mode of functioning and the initial file choice all the work is done in the editor environment (item 3). This work has a visible part (the SOE–user interface: screen, cursor) and an internal one. Invisible for the user, the second part is based on the program representation known as abstract syntax tree (AST). It is used to implement the screen interface, relating cursor movements with the program text and allowing in this way the basic operations application. A linear representation of the AST is designed for saving an intermediate (not finished)program in a file.

The initial visible object being the object for processing when a program is being generated/edited, is the virtual screen created as a result of a file choice. Relations with the syntax tree vertices corresponding to the current state of the screen are set through two types of a cursor: the normal cursor and the structural one. The normal cursor may be replaced by the mouse to accelerate access to a node. The structural cursor goes through empty (not yet generated) nodes and inserted ones (to make the latter visible if they are not hidden) by the users request. Empty nodes are highlighted and nonempty ones are marked by the normal cursor.

Two basic actions applied to the program text in the editor environment are node generation and editing.

Node generation (or insertion) occurs for simple templates immediately, and for alternatives it is preceded by the corresponding menu displaying. Every insertion causes the related window setting and cursors movements correspond further to the coordinates of the window. The final insertion returns the resulting screen state. Exit from the window with some nongenerated nodes may return the screen to the initial state. For the case the special operation is introduced to renew the screen by the user's desire.

Phrase generation is done by typing the appropriate text, as it was mentioned above. There is an operation for phrases selection from those

introduced before. Analysis of the phrase syntax is done immediately after phrase generation.

The number of sons for the templates–lists is unknown in advance and is determined during the process of generating/editing the program. To create new node as list element there exist a special operation requesting if it is subsequent or previous to the current one. The same is done for the templates–options: they are present on the screen, but may be generated or suppressed by desire of the user. A special operation is introduced to restore them.

Node editing is made by means of operations mark, copy, delete, move. A stack of nodes is used to restore the structure of a program in the case of erroneous deleting or moving the subtree during the current session. Its depth may be regulated by the user.

The other kind of editing implemented as independent work and called after the program has been generated, is based on two simple operations on strings characteristic for the text editors, namely, cut and paste. It serves for the purposes of displaying the text of the final program in the manner preferable by the user.

To minimize the size of the final program text on the screen, the operation of packing / unpacking is introduced (the hidden nonempty nodes appear and disappear).

Comments are generated as phrases–options.

A few words are to be said about controlling the speed of program generation. There is a correlation between the level of the user and the degree of detail of metaconcepts. For experienced programmers the latter may be diminished so that the maximal speed could be achieved. A special item in the main menu is present to edit the metadescription of a language and to generate the tables controlling the SOE functioning which are appropriate for this user.

Conclusion

The SOE structure described above isn't complete or closed. At the present time it is under implementation on the principles of object–oriented programming more adequate for the description of the SOE

structure. It is supplemented by a number of other useful and convenient features (for example, context analysis of templates/phrases, parallel hierarchies of procedures and data declarations, more convenient screen interface etc.).

It means that the similar SOE may be used as a base for integrated environments creation which allow passing from the program correct generation to the full syntax structure check–up and to run intermediate programs (prototypes) in the process of the program development.

References

[1] M.V.Zelkowitz. A small contribution to editing with a syntax directed editor, ACM SIGSOFT SIGPLAN Symposium for Practical Software Development Environments, Pittsburgh, PA, April, 1984, p.1–6.

[2] T.W.Reps, T.Teitelbaum. The synthesizer generator: a system for constructing language–based editors. N.Y.: Springer–Verlag, 1988.

[3] V.Lextrait, X.Ceugniet. NEXUS: The meta–generation of versatile graphical multi–user structure editors using generalized attribute grammars. CHI9 Workshop on structure editors, Seattle, April 1990.

N.Shvets,
K.Chebotar,
Institute of Mathematics,
Academy of Sciences of Moldova,
5 Academiei str., Kishinev,
277028, Moldova
e–mail: {22nata,chebotar}@math.moldova.su

Received 13 April, 1995