IMPLEMENTATION OF INTEGRATED DEVELOPMENT ENVIRONMENT AND COMPILER FOR GPL

Kalpesh B Lad, Dr. Bankim Patel

ABSTRACT

Computer programming is important today, simply because technology has taken over the globe for over a decade now. Several new programming languages are coming at a regular interval; usually supporting programming in English language. Very limited worked were available to support programming in regional language. In this paper, authors have presented development of Programming Language (GPL) that support programming in regional language Gujarati.

Keywords: Programming Language, Compiler, Integrated Development Environment, Scanner, Parser, Unicode

1. INTRODUCTION

Computer programming is an exact science in which all properties of a problem as well as consequences of executing it in any given environment can be find out from the text of the program itself by means of purely deductive reasoning [1, 2, 3]. Computer programming is important today, simply because technology has taken over the globe for over a decade now. Having computer programming skills opens many doors when seeking a job [4, 5].

Popular programming languages support programming through English language only. That is working knowledge of English is required when doing programming using any high level programming language [21]. However, country likes India facing problems of illiteracy specifically in rural area and hence, unable to take benefit of computer based technological achievement. Now a day 72% of Indian population resides in rural areas; out of which 52% literacy in the national language [6, 9]. Very few programming languages are developed, which supports programming in regional language. In south India people have tried to develop programming languages and tools in regional language to develop programming skill but were not of much success.

Authors have made an attempt to develop programming language in which programming can be done using Gujarati language. The purpose of this is to provide GPL interface to develop logic as well as programming skill of the student. Implementation explanation of Programming Environment with IDE and various phases of compiler are as bellows [19, 20].

2. PROGRAMMING ENVIRONMENT

A Programming Environment is prone to poor usability problems due to the rich functionality offered through its User Interface [12, 13]. Learner may have difficulties in understanding many of the features provided in it and may have a hard time in locating the appropriate menu elements in the beginning. Improving the usability of a Programming Environment is more of an art than a science as it involve human factors [14]. Different learner will have different usability expectations and usage patterns. As usability is a software quality attribute, authors have decomposed it into following five attributes [16, 17]:

1. Learnability – To measures the ease of learning functionalities.
2. Efficiency – To measures the ease of use and the level of productivity attainable by the learner.
3. Memorability – To measures the ease of remembering the functionality.
4. Low Error Rate – To measures how the environment supports learners in making errors less as much as possible.
5. Satisfaction – To measures how the learners enjoy using it.

Based on these authors have tried to develop a programming environment architecture, which provides a simplified user interface for programming in Gujarati with robust, existing, open source compiler. Also it has been designed and built-in based on plug-in architecture of Linux platform. Each plug-in represents a logical module, which in turn depend on other plug-ins. There are various components integrated in Programming Environment, which support Unicode for regional language Gujarati. It also supports major facilities and utilities, which are common to most of Programming Environment.

GPL is influenced not only by the good characteristics but also by the quality and availability of the whole set, which includes Integrated Development Environment (IDE), Compilers and built-in tools. Architecture overview of GPL is given in following fig. 1.

As shown in the figure 1 the highest-level of Programming Environment consists of following two components.

- An Integrated Development Environment (IDE)
- A Programming Model

An IDE is a source code editor with various features such as Syntax Highlighting Code, Editing Assist, Navigator View, Multi-lingual Text Editing, Multiple Window Handling and Output Window. It also integrated with other tools and utilities such as Document Statistics as well as Search and Replace. In short,

proposed IDE generates code that will be well structured with easy approach. Whereas components of Programming Model is like Lexical Analyzer, Parser, Unicode Handling Library and Code Generator.

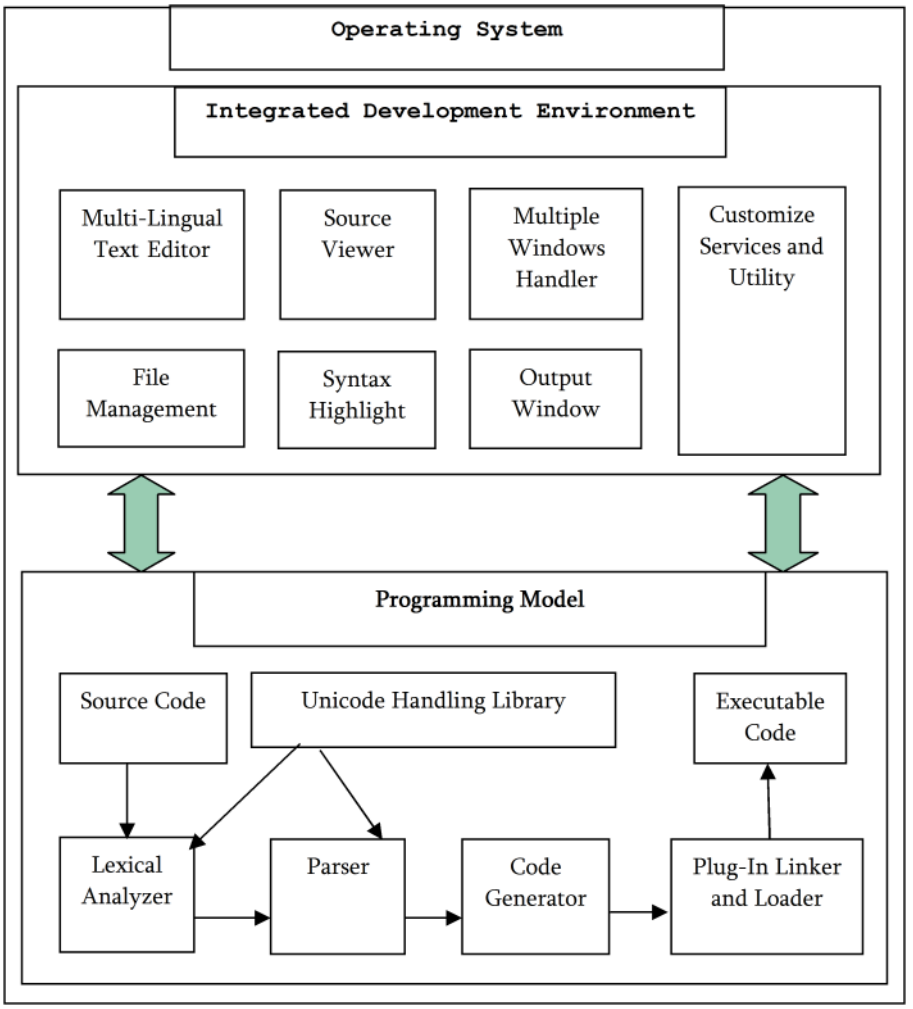

Fig. 1. Components of Programming Environment

3. INTEGRATED DEVELOPMENT ENVIRONMENT

The IDE present an environment in which all activities related to development are carried out. The term "environment" is used to describe an integrated collection of components that assist the learner in developing or maintaining source code artifacts. It provides a number of useful tools and features like

Search-replace, Viewing Toolbar and Status bar, Color selection for background, Normal and Selected text, Convert selected text to UPPER, Lowercase, Capitalize and Invert Case. Also it provides the entire standard menu, dialog, editing, and mouse support, as well as all of the standard shortcuts to which the users of GUI environments are accustomed.

One of the goals with proposed IDE is to help learner to transfer their logic to conventional-style computer syntax. Apart from this, the other one is to simplify and speed the development process through a tighter integration of tools. It is very difficult for beginners to learn different interfaces for programming. Hence, development environment address through the design of an effective multilingual, providing interactive services for the program development and maintenance. It has the entire interface and messages in Gujarati language. Following are some of the functionalities implemented as a part of IDE.

1. **Multi-Lingual Text Editor** – The multi-lingual text editor supports Unicode, ANSI and UTF-8. It is designed to help user to read and edit text or text file(s) in an own language like Gujarati, Hindi, Marathi, English, Chinese, Japanese, etc. It supports all languages supported under Linux platform. Following are major futures for multi-lingual text editor.
    1. Automatic Indenting
    2. Finding and Replace Text
    3. Cursor Motion
    4. Cut and Paste Text

The functions developed for above purpose are as under:

1. The function gedit_view_set_auto_indent() is for automatic indenting purpose.
2. The functions gedit_document_find() and gedit_document_replace_selected_text() are for finding a text and replacing a new text on old text respectively in currant document.
3. The gedit_document_get_cursor() and gedit_document_set_cursor() are for getting cursor position and setting cursor position in current document.
4. The gedit_view_copy_clipboard() and gedit_view_paste_clipboard() are for text copy and paste functionality.

2. **File Management:** Editor provide all functionalities related to text file creation, edition and printing. For this the File menu displayed on top of editor window. Authors have developed various functions for achieving this tasks. Some of them are listed below:

1. The gedit_document_new() function for creating an empty buffer in the editor for a new text file.
2. The function gedit_document_load() displays a source code file in the editor window for viewing and editing, after displaying a dialog to ask for the file to open.
3. The gedit_document_save() and gedit_document_save_as() functions save the contents of current document on secondary storage hard disk.

3. **Source Viewer** – This component responsible for document presentation and editing. It abstracts away many of the base functionality common to source code editors, thus offering an extensive base for customization of common editor features. It supports wrapping lines of code to fit the current window size as well as numbering each line of code. Navigate quickly to a numbered source line with the Go to Line item in the Edit menu. The line number is typed into the data entry area of the window. It also shows the line number in source code at the starting of line and set the tabulation width. Following functions are developed for source viewer component:

    1. The gedit_view_show_line_numbers() function for managing line number displayed beside in the source code.
    2. The function gedit_view_set_tab_size() for setting the width of tabulation in characters.
    3. The gedit_view_set_wrap_mode() function for wrapping lines of code to fit the current window size.
    4. The gedit_document_goto_line() for setting the cursor on specific line number.

4. **Syntax Highlighting** - Syntax highlighting will helpful to understand code faster and to identify errors more quickly. This concept has been implemented in proposed programming language, which can be activated by selecting Programming Language in the menu. If it recognizes the computer language that then highlighting rules are available for that language, it will highlight text and maintain the highlighting automatically as type. This features turn make all control flow keywords appear green and all quoted text as light blue.

    Authors have used GtkSourceView for rendering text and it maintains the language definitions for highlighting. To developed gpl.lang xml file, which have all the reserve keywords and special symbol of GPL and store into /usr/share/gtksourceview-2.0/language-specs directory.

5. **Multiple Window Management** - IDE supports opening of multiple files into different tabs. It also allows opening multiple windows on the screen. Several functions for cascade windows and managing most recently used files list have been developed, which are listed under:
    1. The gedit_mdi_child_new(), gedit_mdi_child_set_closing() and gedit_mdi_child_get_closing() functions developed for MDI management like creating new file and closing tab.
    2. The functions gedit_get_active_window(), gedit_get_active_document(), gedit_get_active_view() and gedit_get_top_windows() developed for activating window.
    3. The gedit_file_close_all() developed for closing all current open file and gedit_file_open_recent() function for managing last open file.
6. **Output Window** – Compiler will report compilation errors in the output window. It will be prompted at the bottom of main window. The compiler also reports warning. Following functions are developed for achieving this task:
    1. The gedit_output_window_new() for creating new output window.
    2. The function gedit_output_window_clear() for clearing the content of output window.
    3. The gedit_output_window_append_line() for inserting content into output window.

Programming Model

The Programming Model is the brain of the Programming Environment and it is responsible for managing the structured and representations used when defining and executing programs. It consists of a lexical Analyzer, Parser, Code Generator, Linker and Loader.

Lexical Analyzer - It transforms a stream of Gujarati characters into tokens. A token is a categorized block of Gujarati text. The block of Gujarati text corresponding to the token is known as a lexeme. A lexical analyzer processes lexemes to categorize them according to function, giving them meaning and ignore comments. This assignment of meaning is known as tokenization.

Tokens are frequently defined by regular expressions, which are understood by a lexical analyzer generator Flex. For example regular expression for set of integer literal is [0-9]+, where as for real literal [0-9]+.[0-9], for string literal \"[^\n]*["] and for the set of identifiers [f-¿>y-yk][f-¿>y-yk 0-9]*.

Lexical analyzer verifies and identifies the reserve keywords of GPL and according report errors, if any. Lexical analyzer has been developed under Linux operating system. It reads the GPL source code file, which consist of Gujarati Language statements. All regular expressions defined in GPL.lex file. Compiling it with the command flex GPL.lex results in the production of the file lex.yy.c, which defines the C function yylex(), which scans the input file an returns the next token.

Unicode Handling Library - Unicode has started to replace ASCII, ISO 8859 and EUC at all levels. It allows programs to utilize any of the character sets. UTF-8 is a serialization method for Unicode that is the de facto standard for encoding Unicode on Linux operating system. Ulrich Drepper's GNU C library glibc has featured since version 2.2 full multi-byte locale support for UTF-8. All Linux distributions come with glibc 2.2 libraries. Using it user interface to generate messages in Gujarati language were developed. Also interaction at program execution time and send it to the stdin of the foreground process. Similarly any output of a process on stdout is sent to the terminal in Gujarati. So the prerequisite of learning a foreign language is removed and the social and monetary benefits of computer technology are more easily realized. Several functions are developed for these purposes. Some of them are as under.

- wcs_to_int() convert the wchar_t string into long integer.
- int_to_locale() convert the Long Integer number into multicharacter Gujarati string.
- wcs_to_float() convert the wchar_t string into C Double number.
- float_to_locale() convert double number into multicharacter Gujarati string.

Parser - A parser is one of the components in a compiler. It is the process of matching grammar symbols to elements in the input data, according to the rules of the grammar. It breaks data into smaller elements, according to a set of rules that describe its structure and checks for correctness of syntax and builds a parse tree. Grammars are written in Backus-Naur Form. The developed parser is a general-purpose parser that converts a grammar description for LALR(1) context-free grammar. It generates a bottom-up parser and parse tree using Bison, which traces a rightmost derivation in reverse by starting with the input string and working backwards to the start symbol. It tries, by shifts and reductions, to reduce the entire input down to a single grouping whose symbol is the grammar's start-symbol. Grammar, precedence and associatively of operators are in grammar rule section of Bison GPL.y file. The precedence of operators are defined as

%left '+' '-'

%left '*' '/'

There is a main() routine which calls the function yyparse(), which is the driver routine for the parser and function yyerror() which is used to report on errors during the parse. Compiling the Bison file with the command *bison -vd GPL.y* causes the generation of two files file.tab.h and file.tab.c. The file.tab.h contains the list of tokens is included in the file which defines the scanner. The file file.tab.c defines the C function yyparse() which is the parser. Grammar rules have an action made up of C statements. Each time the parser recognizes a match for that rule, the action is executed. The task of most actions is to compute a semantic value for the grouping built by the rule from the semantic values associated with tokens or smaller groupings. The C code in an action can refer to the semantic values of the components matched by the rule with the construct $n, which stands for the value of the nth component. The semantic value for the grouping being constructed is $$.

Code Generator- Authors have defined a hypothetical machine, with instruction set and architecture convenient for the execution of programs of the source language. The action of the interface routines will be to translate the source code into an equivalent sequence of operations for the hypothetical machine. The hypothetical machine means stack machine consists of a program store C and a data store S. There are four registers, an instruction register IR, which contains the instruction which is being interpreted, the stack top register T, which contains the address of the top element of the stack, the program address register PC, which contains the address of the next instruction to be fetched for interpretation, and the current activation record register AR, which contains the base address of the activation record of the procedure which is being interpreted.

Each instruction consists of three fields, an operation code and two parameters. Various functions for code generation using stack machine are developed. Some of them are as under:

- data_location() for reserves a data location.
- gen_label() for returning current offset.
- gen_code() is developed for generating code for current location.
- back_patch() function for generating code for reversed location.

REFERENCES:

1. Lawson, Stephen, The Importance of Computer Programming Skills to Educational Researchers, www.eric.ed.gov
2. Knuth, Computer Programming as an Art, CACM, www.paulgraham.com/knuth.html
3. Zsuzsanna Papp, Peter Szlavl, Laszlo Zsako(2008) - ICT teaching methods – programming Languages, Eotvos University, Annales Mathematicae et Informaticae, pp. 163–172
4. Lawson, Stephen, The Importance of Computer Programming Skills to Educational Researchers, www.eric.ed.gov
5. Storey, Sanseverino, German (2003)– Adopting GILD: An integrated learning and development environment for programming, 3rd International Workshop on Adoption-Centric Software Engineering ICSE 2003
6. Policy Matters, A National Policy for ICT in Indian Education , September-2007, www.digitallearning.in
7. McIver, Conway - Seven Deadly Sins of Introductory Programming Language Design, Proceedings of the 1996 International Conference on Software Engineering: Education and Practice
8. Raju Kumar (2008), Convergence of ICT and Education, Proceedings of world academy of science, engineering and technology, volume 30, ISSN 1307-6884
9. Digital Review for Asia Pacific, ICT Profile – India, www.apdip.net/projects/dig-rev/info/in/
10. Balendu Shrivastava, Citius, Altius, Fortius (Faster, Higher, Stronger) Internet In India- I-Cube-2008, Internet & Mobile Association of India
11. National Academy of Sciences, Being fluent with information technology, Washington, DC: Author, 1999, p. 48.
12. Ira P. Goldstein, Daniel G. Bobrow (1980), " Description For a Programming Environment", AAAI-80 Proceeding