World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:7, 2007

System of Programs for Rapid Development and Execution of Palm OS Applications

Mihai Ciocarlie, Marcela-Simona Atanasoae, and Horia Ciocarlie

***Abstract*—**We present the development of a system of programs designed for the compilation and execution of applications for handheld computers. In introduction we describe the purpose of the project and its components. The next two paragraphs present the first two components of the project (the scanner and parser generators). Then we describe the Object Pascal compiler and the virtual machines for Windows and Palm OS. In conclusion we emphasize the ways in which the project can be extended.

***Keywords*—**Compiler design, Palm OS applications, rapid application development, virtual machine.

I. Introduction

THIS paper describes a system of programs intended to be used for rapid development of applications that can be executed on handheld computers with Palm operating system.

The project has two major components:

- a compiler for a subset of Object Pascal language that generates virtual object code [1]
- virtual machines that execute the object code [2] on a Windows platform as well as on a Palm OS platform [3].

This system of programs allows the development of applications that does not involve the use of the applications programming interface (API) of Palm OS. This interface is rather complex, as it is described in detail in Palm OS Programmer's Reference, a document that contains 2360 pages [4].

The implemented language includes besides basic elements (simple and structured types, definition of procedure and functions, etc.) the following advanced features:

- object oriented programming
- exceptions
- file operations.

The entire system of programs was realized using object oriented design techniques [5]. The programming language that was chosen for implementation is C++ [6].

For the development of the compiler we decided to use scanner and parser generators that we developed as part of this project.

Authors are with Computer and Software Engineering Department, Faculty of Automation and Computers, Politechnica University of Timisoara, Romania.

II. The Scanner Generator

The input of the scanner generator is represented by a text file that contains a set of regular expressions which describes the source language atoms and generates the corresponding transitions table and the C++ code of the scanner.

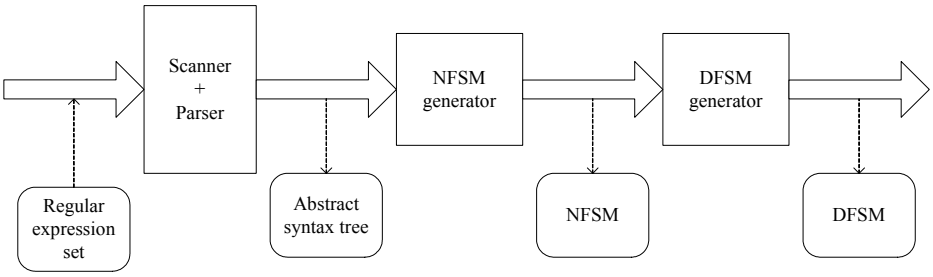

Fig. 1 The structure of the scanner generator

The steps of generation follow below (fig.1):

- the generator creates first an explicit abstract syntax tree corresponding to the regular expressions set from the input file;
- then it creates the corresponding non-determinist finite state machine (NFSM) based on the abstract syntax tree from above;
- finally, the generator transforms the above NFSM into the equivalent determinist finite state machine (DFSM) represented by a transitions table and generates the C++ code of the scanner.

III. The Parser Generator

The input of the parser generator is represented by a text file that contains the productions of the grammar and generates C++ code corresponding to the syntactic analysis as well as to the construction of the complete syntax tree.

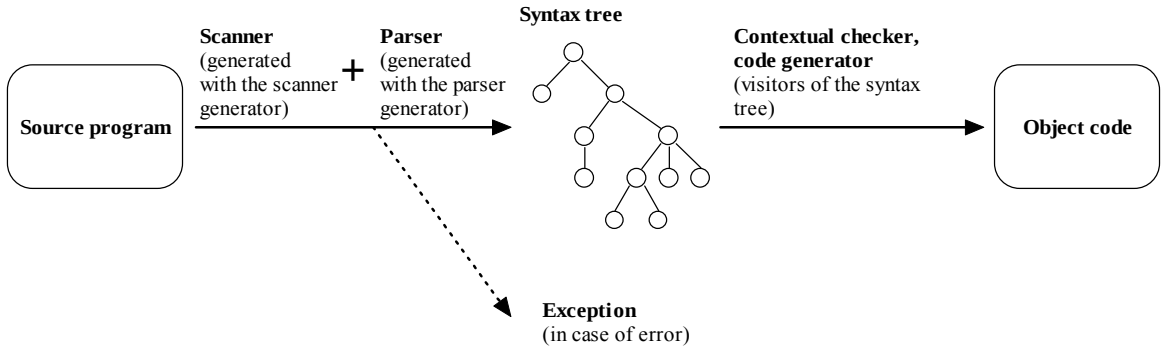

Fig. 2 The phases of a compiler developed using the generators

We have chosen recursive descent parsing because it is best suited for an object oriented design. The syntax tree is complete because for each type of non-terminal in the grammar it is generated a new class of nodes.

Each of the functions that are produced by the generator returns a certain type of node. The children of the node created inside a function are constructed through recursive calls to other functions of the parser.

The parser generator produces C++ code that can be used together with the C++ code produced by the scanner generator.

The user of these tools will be working with the complete syntax tree constructed by the automatically generated parser to realize the other phases of a compiler. Fig. 2 shows the phases of a compiler developed using the two generators.

The generators were used at the developing of the Object Pascal compiler, but also at the developing of an assembler for the virtual object code.

IV. The Design of the Compiler

The development of the compiler is based on a subset of *Object Pascal grammar* which is implemented by the Borland Company in its product Delphi [7].

The main facilities offered by the Object Pascal subset are:

- a type system
- the conditional statement *if*
- the cyclic statements, *for* and *while*
- simple statements like assigns and procedure/function calls

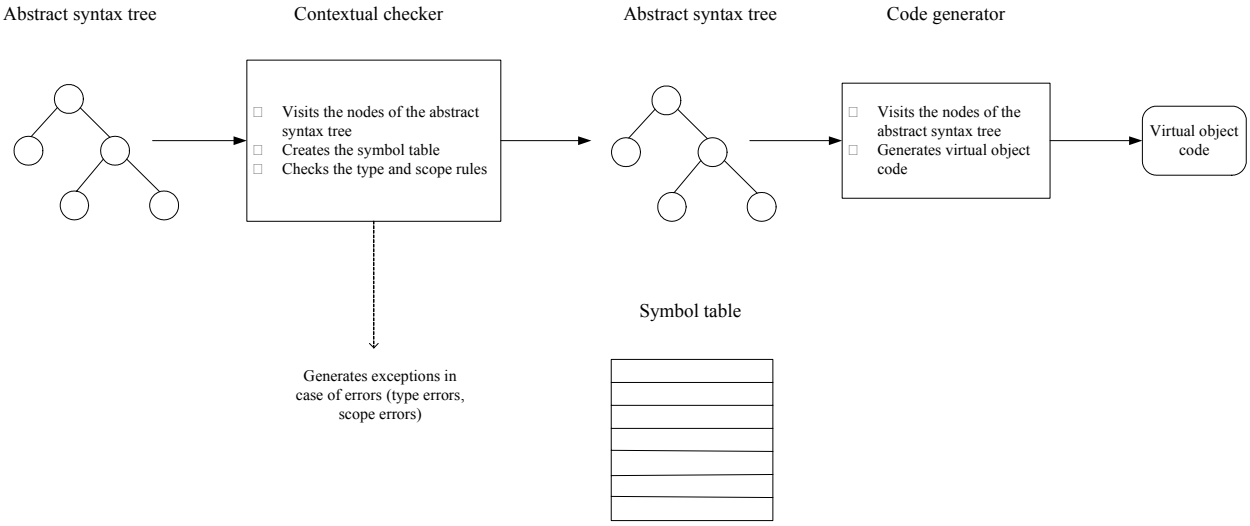

Fig. 3 The structure of the compiler

- procedures and functions
- object oriented programming
- exceptions.

The type system includes:

- the four simple types from Pascal: integer, real, boolean and char
- the Pascal string type
- structured types like arrays and files
- the procedure type.

The type system also allows the definition of new types.

The facilities of object oriented programming offered by this Object Pascal subset are:

- inheritance
- polymorphism
- constructors
- information hiding.

There has been added a set of predefined functions and procedures which are directly translated in the corresponding instructions of the virtual machine. The main feature of this functions and procedures is their fast execution.

In fig. 3 we present the structure of the compiler.

The *abstract syntax tree* corresponding to a Object Pascal program includes two types of nodes: internal nodes, corresponding to the non-terminal symbols of the grammar; and leaf nodes, corresponding to the terminal symbols [8].

Both *contextual analysis* and *code generation* are realized through sequential traversals of the abstract syntax tree. The two complex operations are represented through the *visitor* design pattern. The two visitors are the contextual checker and the code generator. This pattern makes the abstract syntax tree structure independent of its operations.

The *symbols table* is the main structure of the compiler. It is implemented like a hash table. The table contains objects of different classes; each object corresponds to each kind of identifier (variables, constants, etc.). The symbols table is constructed during the contextual analysis [9]. Both contextual analysis and code generation use the information about identifiers contained by this table.

The *errors* reported by all phases of the compilation are treated through the exception mechanism from C++. The way of treating errors doesn't allow the recovery in case of errors, meaning that the compiler stops its execution when the first error is detected.

V. The Design of the Virtual Machine

The largest part of the source code of the virtual machine is platform independent. All the function calls that are platform dependent are grouped within a single interface.

The classes that extend this interface are the only one that contains platform dependent code.

The kernel of the virtual machine has the role of executing instructions. The main components of the kernel are the stack, the data memory and the code memory. Besides the kernel, the virtual machine includes modules for console type interface, graphic interface, communication and file management.

The structure of the virtual machine is shown in the fig.4.

The main feature of the virtual machine is the organization of the data memory. This is a collection of the following types: integer, real, character, logic, string, array and class. The first five store values of simple types, while those of array or class type contain a list of references to other objects of simple type or array/class type.

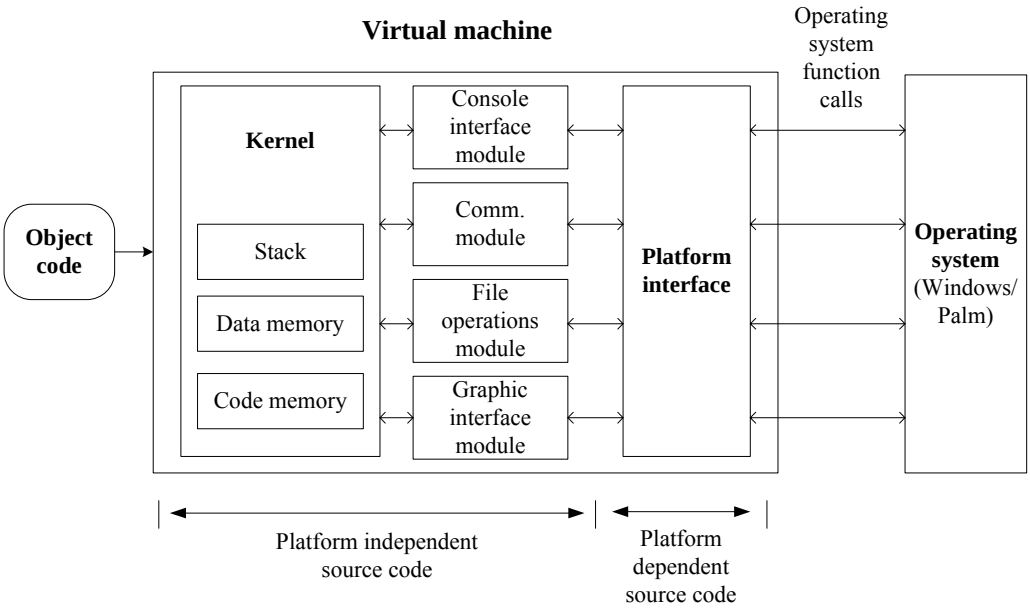

Fig. 4 The architecture of the virtual machine

A reference counter is associated with each object in the data memory, which is used for a mechanism of automatic deallocating.

The instructions of the virtual machine offer direct support for operations specific to object oriented programming, like the dynamic linking of functions (polymorphism). For this purpose the virtual machine accesses a table of classes. Also, there are special instructions for working with exceptions.

There have also been implemented instructions corresponding to common operations (printing to console, file operations, conversions, etc.). If these operations had been implemented like function calls then they would have been executed slower. For other operations an instruction for system calls is used. System calls with specific codes are

generated for working with the graphic interface, for operations with the communication module, etc.

The module for graphic interface offers functions for drawing on the screen, for creation of user interface elements (buttons, edit fields, etc.) or for access to various operating system functions.

The communication module allows communication through infrared (IrDA protocol), or through radio (Bluetooth protocol)[10].

VI. Conclusions

Because modern computers, PCs as well as handhelds, offer good enough computational performances, we focused primarily on the quality of the object oriented design. This way we ensured a clear, modular code, that can be easily extended. This decision proved to be correct as the execution of code is fast enough under Windows as well as under Palm OS.

The project could be developed further through:

- more language facilities (for example pointers, definition of interfaces, modular development of programs, etc.)
- optimization of the processes of compilation and execution
- more execution facilities (more graphic interface components, access to more functions of the operating system)
- development of virtual machines for other platforms.

The system of programs can be easily modified to allow execution of programs on other platforms, as most of the code of the virtual machines is portable, platform independent.

References

[1] Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen, *Modern Compiler Design,* John Wiley & Sons, 2003.
[2] David A. Watt, Deryck F. Brown , *Programming Language Processors in Java – Compilers and Interpreters*, Prentice Hall, 2000.
[3] Lonnon R. Foster, *Palm OS Programming Bible*, Wiley Publishing, 2002.
[4] *** Palm OS Programmer's API Reference, PalmSource Inc.,2002.
[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns – Şabloane de proiectare*, Editura Teora, 2002.
[6] Kris Jamsa, Lars Klander , *Totul despre C şi C++*, Editura Teora, 2001.
[7] Steve Teixeira, Xavier Pacheco, *Delphi 5*, Editura Teora, 2002.
[8] Spinllis D., *Global Analysis and Transformations in Preprocessed Languages*, IEEE Transactions on Software Engineering, Vol. 29, No. 11, November 2003.
[9] Horia Ciocarlie , *A Mechanism of Visibility Control*, Preceedings of the International Conference on Signal Processing, ICSP Istanbul, 2004.
[10] *** *Palm OS Programmer's Companion*, PalmSource Inc., 2002.