Advanced Science and Technology Letters
Vol.106 (Information Technology and Computer Science 2015), pp.52-55
http://dx.doi.org/10.14257/astl.2015.106.12

A Study on the JavaScript Compiler for Extension of the Smart Cross Platform

Yunsik Son[1], Jaehyun Kim[2], Yangsun Lee[2*]

[1]Dept. of Computer Engineering, Dongguk University
26 3-Ga Phil-Dong, Jung-Gu, Seoul 100-715, KOREA
sonbug@dongguk.edu
[2]Dept. of Computer Engineering, Seokyeong University
16-1 Jungneung-Dong, Sungbuk-Ku, Seoul 136-704, KOREA
{statsr, yslee}@skuniv.ac.kr
[*]Corresponding Author

Abstract. In this paper, we will introduce the JavaScript compiler for the SCP (Smart Cross Platform) to expand the SCP's coverage. The SCP is the virtual machine based solution that supports various programming languages and platforms, and its aims are to support programming languages like ISO/IEC C++, Java and Objective C and smartphone platforms such as Android and iOS. By adding the new compiler to SCP's compiler collection, SCP can be support the new type of the programs such as web applications written by JavaScript.

Keywords: Smart Cross Platform, Virtual Machine, ECMA Script, JavaScript Compiler, Compiler Construction

1 Introduction

The existing developmental environments for smart devices contents are needed to generate specific target code depending on target devices or platforms, and each platform has its own developing programming language and methodologies. These characteristics in smart contents developments make the contents development process very inefficient. Therefore, even if the same contents are to be used, it must be redeveloped depending on the target machine and a compiler for that specific machine is needed, making

SCP (Smart Cross Platform) is a virtual machine based solution which aims to resolve such problems, and it support programming languages like C/C++, Java, and Objective C [1, 2, 3]. In this study, a compiler for use in a program designed in the JavaScript programming language [4] to be used on a SCP is designed. In order to effectively implement the compiler, it was designed to seven modules; lexical analysis, syntax analysis with SDT (Syntax Directed Translation), error recovery, symbol collector, semantic analyzer, code generator, and symbol table.

ISSN: 2287-1233 ASTL

2 Relative Studies

2.1 Smart Cross Platform

The Smart Cross Platform was developed by our research team as a platform based on virtual machines for smart devices. It is composed of three parts; compiler, assembler and virtual machine. It supports C/C++, Objective C and Java programming languages for contents development [1, 2, 5]. Contents made of each language are converted to an intermediate code by the compiler. SIL is used as intermediate code which can accommodate procedural language and object orientated language. Intermediate code is converted to execution format by the assembler and executed in the SVM (Smart Virtual Machine).

Smart cross platform compiles an application program and is composed of three parts. A compiler creating SAF format file composed of SIL (Smart Intermediate Language), an assembler converting SAF (Smart Assembly Format) file to a SEF (Smart Executable Format), and a virtual machine executing SEF format file entered [2, 3]. The SVM system is designed as a hierarchical model minimizing burden created during the process of retargeting other devices and operating environment.

2.2 The JavaScript Programming Language

JavaScript is one of the object-based programming language that is used on fields for web applications. In HTML5 technology, JavaScript is a core component as a programming language to develop not only client-side HTML 5 contents and web applications, but also server-side network applications as Node.js.

JavaScript was developed by Brendan Eich in Netscape Communications, and standardized as EMCA Script. It has C-style syntactic structure and the JavaScript's syntax intentionally resembles Java syntax, but provides many various primitive types like Undefined, Null, Boolean, Number, String, Symbol, and Object. Also, variables can change type dynamically. It is a major issue to design and implement the JavaScript compiler.

In execution aspect, JavaScript program is interpretively executed by JavaScript Engine embedded in web browsers, not compiled. This is another major issue and differ characteristic of our SCP's JavaScript from general JavaScript. Our proposed JavaScript developmental environments are based on compilation – execution method.

3 JavaScript to SIL Compiler

The JavaScript compiler supports the contents written in JavaScript language on SVM which generates platform independently stack-based SIL code as target code. In this study, the JavaScript to SIL compiler was designed by 7 parts as can be seen in Fig. 1.

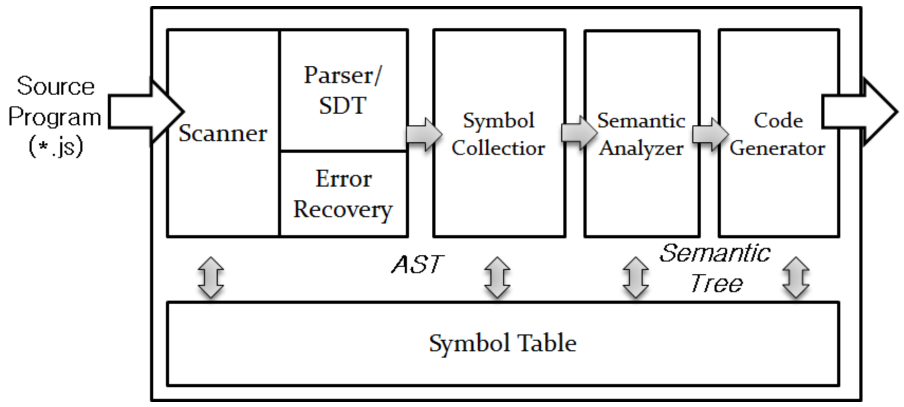

Fig. 1. JavaScript to SIL Compiler Model

The JavaScript to SIL compiler implements the characteristics of the JavaScript language and therefore was designed with seven different parts; Scanner (lexical analysis), Parser (syntax analysis) with SDT (Syntax Directed Translation), Error recovery, Symbol collector, Semantic analyzer, Code generator, and Symbol table. The detailed information for each part is as follows.

Scanner, Parser with SDT, and Error recovery modules are easily can be group as a processor to analyze the input JavaScript programs and generate analyzed AST (Abstract Syntax Tree) for input program

The symbol collector carries out the job of saving information into the symbol table which is obtained by traversing and analyzing the given AST. The routines consist of the functions, dynamic types, declarations and others.

Semantic analyzer is composed by two parts; the declarations semantic analysis module and the statements semantic analysis module. The declarations semantic analysis module checks the process of collecting symbol information on the AST level, to verify cases which are grammatically correct but semantically incorrect. The statements semantic analysis module uses the AST and symbol table to carry out semantic analysis of statements and creates a semantic tree as a result. A semantic tree is a data structure which has semantic information added to it from an AST. It is responsible for all that has not been taken care of during the syntax analysis process and then it is used to generate codes as it has been designed to generate codes easily.

The code generation part receives the semantic tree as an input after all analysis is complete and it generates a SIL code which is semantically equal to the input program (*.js). For this, the SIL code is expressed as symbols so it is convenient to generate and handle them. For type conversion code lists, the same data structure is kept so that the code generation process can take place efficiently. Type conversion code lists are data structures that pre-computed the process of converting a semantic code into a SIL code when generating a code. A code generator visits each nodes of the semantic tree to convert them into SIL codes.

Finally, symbol table is used to manage the symbols and its attributes in the given programs. The symbol table is designed for easily managing the symbol information and reflected the semantic features of the JavaScript.

4 Conclusions and Further Researches

In this paper, we have designed a new compiler to support the JavaScript programs as the one of the compiler in compiler collection of Smart Cross Platform. We defined seven modules to construct the proposed compiler and to generate a SIL code for use on a SVM which is independent of platforms. We hope to the compiler will be implemented that is expand the coverage of Smart Cross Platform and reduce the cost of retargeting the exist contents.

In the future, there is need for implementation phase of a JavaScript-SIL compiler so that JavaScript base contents can be run on a SVM. Further research on optimizers and assemblers for SIL code programs are also needed so that SIL codes that have been generated can run effectively on SVMs.

Acknowledgments. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning (No.2013R1A2A2A01067205).

References

1. Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Executing Virtual Machine Codes on Smart Platforms", International Journal of Smart Home, SERSC, Vol. 6, No. 4, 2012, Australia, pp. 93-105.
2. Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Smart Devices", Information -an International Interdisciplinary Journal, Vol. 16, No. 2, International Information Institute, 2013, Japan, pp.1465-1472.
3. Y. Son, Y.S. Lee, "Design and Implementation of an Objective-C Compiler for the Virtual Machine on Smart Phone", CCIS, Springer, Vol.262, 2011, Heidelberg, pp.52-59.
4. ECMAScript 2015 Language Specification, 6th Edition of ECMA-262, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
5. Y.S. Lee, S.M. Oh, Y.S. Son, "Design and Implementation of HTML5 based SVM for Integrating Runtime of Smart Devices and Web Environments", International Journal of Smart Home, SERSC, Vol.8, No.3, 2014, Australia, pp.223-234.