ISSN:0975-9646

Transpiler and it's Advantages

Rohit Kulkarni[1], Aditi Chavan[2], Abhinav Hardikar[3]

Department of Computer Engineering,
Modern College of Engineering, Shivaji Nagar, Pune, India

***Abstract*— This paper sheds some light on what a transpiler is and it's advantages when it is used with a normal compiler. It also introduces a concept called "Pluggable Target Compiler" and it's advantages.**

***Keywords*— Transpiler, Compiler, source-to-source compiler.**

I. INTRODUCTION

Transpiler and Compiler are one of the major parts of a larger domain called System Programming. A Compiler can be simply defined as a program that reads a program in one language, known as the source language and translates it into an equivalent program in another language, known as the target language.[1] In a simple compiler, source language is a high level language and the target language is the corresponding object program.[2]

A transpiler or a source-to-source compiler is a compiler where the source language as well as the target language is a high level language.

II. PHASES OF A TRANSPILER

A transpiler has phases very similar to that of a compiler. The phases are:

1. *Lexical Analysis:* In this phase each character is read and grouped into meaningful sequences called lexemes[1] or tokens. For example, consider the following sentence:

"IBatType baseBallBat = new BaseBallBat("my bat");"

After lexical analysis the following lexemes are generated:

a. IBatType
b. baseBallBat
c. =
d. new
e. BaseBallBat
f. (
g. "my bat"
h.)
i.;

2. *Syntactic Analysis:* This is the phase where the lexemes are parsed and major constructs of the language are recognised. These constructs are then represented in the form of a tree structure called a syntax tree. [1]
3. *Semantic Analysis:* Using the syntax tree the semantic analyser checks for semantic consistency. For example, to check whether all the variables in an equation are of the same type. [1]
4. *Intermediate Code Generation:* For every construct defined, there is a routine which is executed when the respective construct is recognised during the syntactic analysis.[2] These routines hold the intermediate code structure, which is used to generate the intermediate code. Intermediate code can be represented in a structured form like the 3-Address code[1] or in any structured language like XML or JSON.
5. *Machine Independent Optimisation:* The aim of this phase is to optimise the intermediate code so that the resultant target code that is generated will be better.[1] Some of the goals of code optimisation are:
   a. Discover program run-time behaviour at compile time.
   b. Speed up runtime execution of compiled code.
   c. Reduce the size of compiled code.
6. *Translation:* In this phase intermediate code is translated into the target cod with the help of translation schemes.

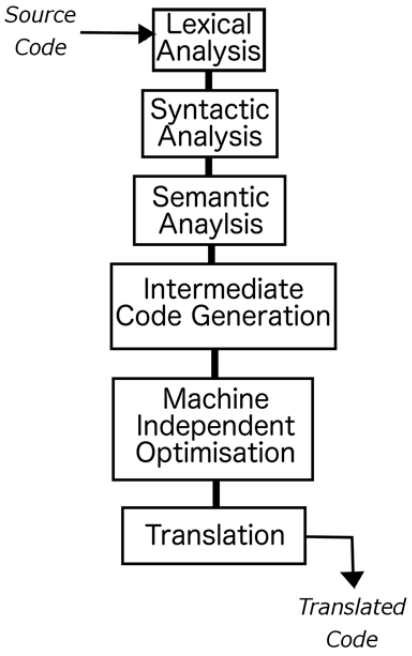

Fig 1: Phases of a Transpiler

III. ADVANTAGES OF A TRANSPILER

The following are the advantages of using a Transpiler:

1. A transpiler has the ability to translate it's language to the target language, which is then compiled using the target language's compiler to generate native or VM code. Therefore the transpiler automatically inherits the advantages of the target compiler.
2. Since the transpiler generates structured, human readable intermediate code, it is easy to convert this structured intermediate code to any other language using a simple translator.
3. It reduces the amount of time taken to convert a source code from one language to another language, as it need not be done manually.
4. Larger constructs of the target language can be shortened and implemented in the transpiler's language as smaller constructs. This would reduce the amount of typing needed to get the work done and also reduce its complexity.
5. Transpiler is a great medium to learn programming as they can provide simplified syntaxes and constructs.

IV. Pluggable Target Compiler

What is a Pluggable Target Compiler?

One of the most famous cross-platform programming languages is Java. It achieves the cross-platform ability by using a executor called the Java Virtual Machine or the JVM. JVM takes bytecode as input, converts it to native code and executes it immediately. JVM has been implemented in almost all major operating systems.

Before Java 7 was released to the public, Java was criticised for being slow in comparison to native programming languages like C [3]. Another criticism of Java that still persists in Java 8 is the aloofness of API (Application Programming Interface) from the native system. An example of the aloofness can be the Look and Feel (themes) of the applications developed in Java. Another example can be the difficulty of using native APIs of the client operating system.

To solve such issues a transpiler can be made in which the code generator is separate. This code generator can be plugged into the transpiler and can generate native code for the respective operating systems thus allowing easy access to the native APIs.

Another advantage of such a code generator is that it can be programmed in any programming language and it gives the developer the freedom of choosing the appropriate target compiler. A target compiler is a compiler which compiles the code generated by the code generator.

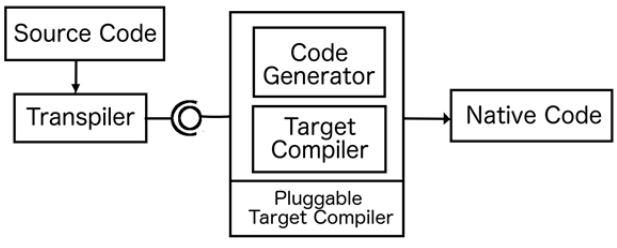

Fig 2: Block representation of a Pluggable Target Compiler.

This module of the code generator and the target compiler bundled together is called the Pluggable Target Compiler. Fig.2 shows a block representation of a Pluggable Target Compiler.

How is a Pluggable Target Compiler used?

Consider two systems, one with Windows 7 and the other with Apple Mac OS X. Windows 7 supports C++ for native code generation and Mac supports Objective-C.

Therefore it is needed to create two pluggable target compilers with:

1. Input: XML/JSON Intermediate Code
   Output: Objective-C source file
   Target Programming Language: Objective-C
   Written in: Objective-C

2. Input: XML/JSON Intermediate Code
   Output: C++ source file
   Target Programming Language: C++
   Written in: C++

In these target compilers, rules have to be written to convert the intermediate code into equivalent target language code.

To plug these target compilers, an interface is used. The developer has to just select the correct interface and compile the code.

Once the code has been translated into the respective target language, the target compiler is run and the native code is generated.

V. Advantages of Pluggable Target Compiler

The aim of the transpiler with a pluggable target compiler (TwPTC) is to generate runnable code on various platforms without writing code for each platform. The advantages of using a TwPTC are:

1) It generates native code which is generally faster than the programs compiled using a cross-platform compiler.
2) TwPTC generated code has easy access to the native API as it is directly compiled to native code.
3) It allows for a single front-end (source language) to have multiple back-ends (target languages) which is advantageous over a normal transpiler.
4) Since the translation schemes are stored in it, it is very easy to improve the efficiency of them and regenerate the new code.
5) Since the Transpiler and the PTC are separate, they can be installed on separate machines and using network protocols it can allow compiling over any network.
6)

VI. Conclusion

There are many transpilers available on the Internet like Cetus or ROSE, but most of them have fixed target languages. With the help of PTC this limitation can be removed and a single language can be used to generate native executable for many platforms.

REFERENCES

1. A. V. Aho, M. S. Lam, R. Sethi and J. D. Ulman, “Compilers: Principles, Techniques and Tools”, 2nd ed., Pearson Education Inc., 2013, ISBN-13: 978-81-317-2101-8.
2. J. J. Donovan, “Systems Programming”, Tata McGraw-Hill Edition, 1991, ISBN-13: 978-0-07-460482-3.
3. Computer Language Benchmarks Game, http://benchmarksgame.alioth.debian.org/u64q/java.html