©2010 International Journal of Computer Applications (0975 - 8887)
Volume 1 – No. 20

# Programming Language Inter-conversion

Dony George
FCRIT, Vashi, Navi Mumbai

Priyanka Girase
FCRIT, Vashi, Navi Mumbai

Mahesh Gupta
FCRIT, Vashi, Navi Mumbai

Prachi Gupta
FCRIT, Vashi, Navi Mumbai

Aakanksha Sharma
FCRIT, Vashi Navi Mumbai

## ABSTRACT
In this paper, we have presented a new approach of programming languages inter-conversion which can be applied to all types of programming languages. The idea is about implementation of the intermediate language for inter-conversion. This language can be used to store the logic of the program in an algorithmic format without disturbing the structure of the original program. This paper also discusses the major advantages and challenges associated with this conversion approach. We have also performed a theoretical case study on the conversion of code written in the C++ programming language to Java.

## General Terms
Legacy System, Assembly Language, Operator Overloading, Object Oriented Programming, Procedural Programming, Extensive Markup Language (XML), Pointers

## Keywords
Intermediate Language, XML Tags, Turing Machine, C++.

## 1.BACKGROUND
In the early stages of programming history, assembly languages (a family of low level languages) were used for programming computer, microprocessors & microcontrollers. However due to their complexity, it was difficult to work with them.

Hence higher level programming languages came into existence. To convert the program written in high level language into machine language, a software known as a compiler is used. Higher level languages were better than lower level languages because they were much easier to work with. They had better program readability and hence easy program management.

Over time, various higher level programming languages came into existence starting from BASIC and COBOL, based on *procedural programming* approach, up to C++ and Java, based on *object oriented programming* approach.

All the modern programming languages are Turing Complete in terms of their primary features (expressive power & computation)[8]. The point where they differ is the environment for which they are designed to work. Based on the environment, for which a language has been designed, their secondary features (Platform specific features) differ from each other. Each language has its own specific features and hence a corresponding payoff in the form of disadvantages associated with them.

For example, C++ allows direct access to processors' internal registers. However, it lacks some important features that can lead to serious errors such as system security not being maintained. Though the keyword "goto" adds extra expressiveness to any C++ program, it also leads to serious problems related to memory management and automatic de-allocation of objects.

Java on the other hand provides high level of system security. It also supports various types of features through its language library such as accessing network, GUI design, parallel processing etc. However it is unable to provide access to machine registers. Hence based on various factors such as user requirement and system security, a specific programming language is chosen for system implementation.

The difficulty arises when an existing code needs to be implemented again using a different programming language. With time, new programming languages are developed and it may be necessary to switch our programs to them. The difficulties may arise in the rewriting and testing of standards written in one language in another language.

Hence it becomes necessary to have some tools perform these tasks for us.

## 2.INTRODUCTION
Programming Languages' Conversion has been a challenging issue since almost a decade. Conversion of a code written in one language to another does not just mean the inter-conversion of syntax between these languages. It is the operation of performing transformation while maintaining the structure as far as possible. At the panel of 'International Conference on Software Maintenance', Language conversion has been formulated as one of the ten most challenging problems of next Century [2].

Various research papers have been written to express the ideas related to this field. Early ideas were based on designing a simple

syntax replacing system. Most of modern ideas & research papers concentrated on designing a compiler for the same purpose [3].

Replacement algorithms work for simple codes but are unable to maintain the structure of complex codes. The modern approach only converts the program but does not add the language specific features of the destination programming language. Also, operation of most of the compilers is based on converting a specific source code into another specific source code. However, there are many programming languages and designing a compiler for inter-conversion between each of these languages would be quite expensive.

The migration of a legacy code into another programming language can be done at different levels which are increasingly more ambitious and difficult to implement. Hence, as we go higher up the levels, the need for manual conversion also increases. At lower levels, migration takes the form of transforming the code from one language to another. At higher levels, the system structure may be changed as well, for example conversion of a code written in a purely procedural language into a purely object oriented language. At still higher levels, the global architecture may also have to be changed.

# 3.DIFFICULTIES

The different languages provide different features based on the platform which they are being implemented. This leads to difficulties in designing an intermediate language for them as the same feature may or may not be supported in the other language. Hence code conversion becomes even more challenging because the features of the source language need to be somehow simulated into the destination language. Hence, this imposes a limitation on code conversion.

Another problem which we have to deal with while building a converter is the conversion of data types. Although we do not always realize it, different programming languages usually have different data type conventions. Consider the example of C++ & java. The concept of pointer & pointer related operations are supported only in C++. Although an equivalent feature is also present in Java in the form of References (also called as an Internal Pointer), both of them are not the same because java does not allow arithmetic operations to be performed on references, while arithmetic operations are allowed to be performed on pointers in C++. Moreover, the data type sizes also vary from platform to platform for C++, whereas they are fixed in Java.

With new technologies emerging every day, it becomes even more difficult to maintain uniformity between different languages. Also with advancement of programming languages, new features get introduced. This demands consequent changes in all the inter-conversion compilers associated with that language.

For example, ANSI C++ gets revised every 3-5 years. The revised language may have new keywords. For example ANSI C++0x has added `constexpr` as the new keyword [9]. Hence, all the identifiers with the same name in early programs need to be modified otherwise they will be interpreted incorrectly.

# 4.EXISTING TECHNOLOGIES

Some of the existing computer language converters are:

1. JLCA [7]:

Java Language Conversion Assistant is a tool that automatically converts existing Java-language code into Visual C# code. It provides developers using Visual Studio .NET 2003 a quick, low-cost method of converting Java-language applications to Visual C# and the .NET Framework. These applications can then be extended to utilize XML Web services and the complete .NET developer platform, including ASP.NET, ADO.NET, and Microsoft Windows® Forms.

The drawback of using JLCA is that it converts the Web/JSP apps to ASP.NET 1.1 apps, which is compatible with Visual Studio 2003. Visual Studio 2005 can be used only with the ASP.NET 2.0 apps. Thus the ASP.NET apps migrated by JLCA are not fully compatible with Visual Studio 2005. Also, there is a limitation with using Windows Forms Designer for the upgraded form in Visual Studio because Windows Forms Designer cannot design forms derived from *'com.ms.wfc.ui.Form'*. So, any change in the upgraded forms' user interface layout must be done manually.

2. BCX [6]:

BCX is a small command line tool that inputs a BCX BASIC source code file and outputs a 'C' source code file which can be compiled by any C or C++ compilers. Using BCX and a C compiler, we can produce powerful 32-bit native code Windows console mode programs, windows GUI applications, and Dynamic Link Libraries (DLL's) without having to incur the costs of an expensive commercial BASIC compiler.

The only drawback is that Hardware Voices Controls are disabled.

3. PERTHON [5][10]:

Perthon converts Python source code to human- readable Perl 5.x source code. It makes use of Damian Conway's Parse::Rec Descent for parsing, and aims to re-implement the Python language as specified in the Python Reference Manual and BNF grammar. Perthon is similar to Jython, which re-implements Python on the JVM, except that Perthon works at the source code (not byte code) level. Perthon does not yet support 'use', 'BEGIN', 'END', etc. This is due to how Perl handles these expressions: they get executed while parsing.

It also does not handle 'bless', 'packages', etc. The references may or may not be resolved correctly. The prefix/postfix operators are not resolved as well.

# 5.OUR PROPOSITION

## 5.1 The Conversion Process

The conversion process is an iterative process, i.e., like a compiler there are several processes involved in this. For a better conversion process, the system should not be just a simple syntax replacement system. It should also preserve the structure or should modify it to make it even better by removing redundant codes. Hence, it will be better if a compiler is designed for this purpose. And the purpose of compilation is to convert the given program into its corresponding Intermediate-language. This

Intermediate-language can be converted into any programming language using another compiler.

## 5.2 Intermediate Language

The Intermediate-language grammar should be designed with special care such that it should be able to represent all possible programs i.e. it should able to represent machine level, functional programs, procedural programs, as well as object oriented programs. Nowadays various language conversion tools work using a specific type of language only. [1]

The idea of having an intermediate language is that since all the programming languages have some common features such as logical and arithmetic operators, looping and conditional statements, built-in data types, user defined data types, comments etc. So, based on these facts a new language can be defined having all these features to represent the characteristics of the programming language.

While converting into destination language from intermediate language, various other issues are also present. One major issue is how to simulate the feature of source language into destination language if it is not present. If a feature is directly not available in the language then it can be modelled and implemented using a separate library. Thus a library can be used to simulate the features of source language and also to add the language specific feature of the destination language while performing conversion.

There are various features which can only be found in native language but modern languages do not support them. Such features in any case cannot be represented in the destination language. Various features have been deprecated from higher programming language for security and readability purpose.

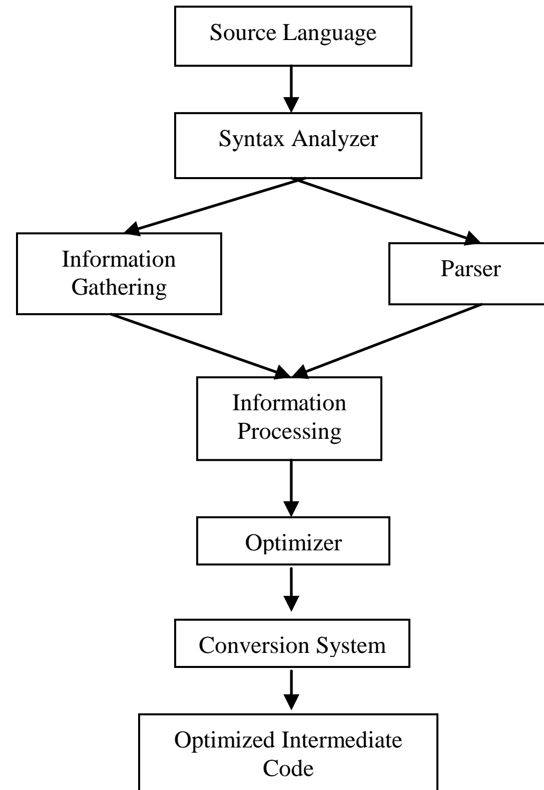
For example, machine languages can be written in C++ / C but this feature is not available in Java. Also the keyword ‘goto’ in C++ allows flow of control to jump from a point in the code to any other point. However, this jumping can lead to serious problems related to memory management and security issues. Hence, it is not recommended to have such a feature in a larger system and so it has not been implemented in Java.

## 5.3 Process

### *5.3.1 Intermediate-language Conversion Process*

**Step 1**

The Language is first processed through the compiler. The compiler checks for any syntax errors and displays the errors if present. The Process continues if and only if the program is syntactically correct. This acts as a first layer in the compilation process.




**Schematic Representation of Language Conversion Process**

**Step 2**

The program is parsed to determine its structure. This structure helps in creating a parsing tree which also helps in the resolution of any ambiguity present in the program.

**Step 3**

The program is scanned again to determine the additional data regarding the program such as variable names, function/ method names, function overloading or overriding if found and other important information.

**Step 4**

Now, based on the data collected in steps 2 & 3, the conversion software converts the code into an intermediate language file.

**Step 5**

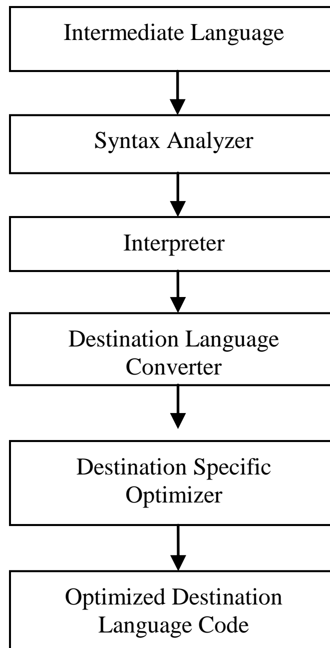
Further this converted code is passed through a special ‘Language Optimizer Tool’ which optimizes the code; so that any unnecessary codes are removed and certain codes are replaced by their optimal equivalent codes, if possible.

Larger Programs are generally written in modules. In such cases a separate Intermediate-language file can be prepared for each module. Later, individual files can be combined while converting program into destination language.

The output contains the language which is an intermediate representation of the source code. To convert this code into

destination language, the output needs to be passed through the de-conversion process of destination compiler.

### *5.3.2 Destination Language Conversion Process*




**Schematic Representation of Destination Language Conversion**

**Step 1**

This steps checks if the syntax of the Intermediate-language is correct or not. This is necessary to ensure that the destination language is perfectly valid.

**Step 2**

Now, separate interpreter software can be used to convert the intermediate language code into its destination language equivalent.

**Step 3**

For further optimization, a specific pattern in a code can be recognized which can be replaced by its equivalent optimum code. This task can also be done manually. The major advantage of separately implementing this optimizer is that, if a change has to be made in the optimizer later, it can be simply made by changing the optimizer without modifying the actual code.

**Step 4**

The output of the optimizer is the source code of the language.

The major advantage of this structure is that we have a separate compiler for each language. Thus, changes or updates made in one of the programming language do not affect processing of other compilers.

Also, the advantage of having separate software for de-compilation of the converted code into programs is that for each individual language we can have a separate optimization function to perform language specific optimization.

The intermediate language should be selected in such a way that it should be able to represent all the programming languages. Thus, a standardization committee can be chosen which can decide the representation standard.

For Example: An XML language can be chosen for the intermediate language. Consider, the java statement

int a = 10 ;

This statement can be represented into intermediate language as

"<start> <data_type>Integer <var>a := 10 <end> "

Here, each line starts with a <start> tag & ends with an <end> tag.

Similarly, <Block> & </Block> can be used to represent a block of codes.

The major advantage of using XML representation is that it can represent all types of statement written in any programming language.

# 6.APPLICATIONS

Nowadays, newer and better programming languages are being developed frequently. Thus, with time programming language needs to be changed. But various industry related standards and other mathematical standards are written in older programming language. Toolkits can play an important role in converting these standards to modern language while maintaining the accuracy and precision of the operation. Thus, this leads to a cost effective solution.

Today businesses struggle with the problem of how to maintain the investment made in their software applications that were developed using "legacy" programming languages. Newer programming languages can offer businesses access to new technologies, growing developer populations, and lower maintenance/development costs. Unfortunately, transitioning from one programming language to another is not easy. Reengineering and manual rewrites can take years, can consume large amounts of manpower, and can result in lost/broken functionality. Automated language conversion services solve these problems.

# 7. CASE STUDY (C++ to Java Converter)

To demonstrate our idea, we have performed a theoretical study on the feasibility of converting a code written in C++ to a Java code.

## 7.1 Background

### *7.1.1 C++*

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as a middle-level language, as it comprises of a combination of both high-level and low-level language features.

The prominent application domains of C++ are system software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.

### *7.1.2 Java*

Java derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities.

The prominent features of Java are its platform independence, automatic garbage collection, vast availability of ready-made APIs and so on. Java is the foundation for Web and networked services, applications, platform-independent desktops, robotics, and other embedded devices.

## 7.2 Comparison

Both C++ and Java have their unique advantages and disadvantages making it difficult to make a statement as to which is a better programming language.

### *7.2.1 Design Goals*

C++, which extends the C language, was designed mainly for system programming. C is a procedural programming language designed for efficient execution. C++ has added support for statically-typed object-oriented programming, exception handling, scoped resource management, and generic programming, in particular. It also added a standard library that includes generic containers and algorithms.

Java was created initially to support network computing. It relies on a virtual machine for security and high portability. It is bundled with an extensive library designed to provide a complete abstraction of the underlying platform. Java is a statically typed object-oriented language that uses a syntax similar to C, but is not compatible with it. It was designed with the goal of being easy to use and accessible to a wider audience.

#### *7.2.1.1 Speed*

C++ is traditionally known to be as much as 3 times faster than Java. However, Java programs' execution speed improved significantly with the introduction of Just-in-time compilation in 1997/1998 for Java 1.1, the addition of language features supporting better code analysis and optimizations in the Java Virtual Machine itself, such as Hotspot becoming the default for Sun's JVM in 2000.

#### *7.2.1.2 Performance*

Early versions of Java were significantly outperformed by statically compiled languages such as C++. This is because the program statements of these two closely related languages may compile to a few machine instructions with C++, while compiling into several byte codes involving several machine instructions each when interpreted by a JVM.

In addition to executing the compiled code, computers running Java applications also need to run the Java Virtual Machine (JVM), whereas compiled C++ programs can be run without external applications.

## 7.3 Conversion using Intermediate-language

### *7.3.1 Intermediate-language Syntax*

Here, we will be discussing about how the intermediate language should represent the C++ & Java programs for their inter-convertibility. We are considering the language implementation to be only for C++ to Java and vice versa. The intermediate language for the conversion should be chosen such that it should be able to represent each and every feature of ANSI C++ (We are considering ANSI because it is the standard version of C++) and Java.

It is said that Java has evolved from C++. There are various features which are common to C++ & Java for e.g.: looping, conditional statements, data types, classes, etc. Thus, these features can be represented by tags, similar the ones used in XML, i.e., each feature can be represented by a separate category tag.

For e.g.

Sample 1:

```
if (x == 5)
{
        x = 7;
}
```

The above statement can be represented by::

```
<IF> x <EQUALS> 5
<START>
        <begin>   x = 7 <end>
<END>
```

Sample 2:

```
class T{
        int i;
        public T(int ii){
                i = ii;
        }
        public void display(){
                System.out.println("I    is:    "+i);

        }
}
```

The above class definition can be implemented as:

```
<User_Type name = "class"  identifier = "T" access = "public">
<Start>
        <begin> <data_type> integer <identifier> i <end>
        <method access = "public" type = "constructor">
        <Start>
                <param type = "integer"
                identifier = "ii">
                <begin> i = ii <end>
        <End>

        <method access = "public"
```

*type = "void"*

*identifier = "display">*

*<Start>*

*<begin> <Display> <string>*

*"I is: "<value> I </Display>*

*<end>*

*<End>*

*<End>*

Each line starts with a <begin> tag and terminates with an <end > tag.

The common tags such as IF, FOR, WHILE, SWITCH can be used to represent key words. The scope can start from the <START> tag and end with the <END> tag. Complex Systems such as pointers, unsigned int are difficult to represent. Hence they can be represented by predefined special tags. The difference between the previous tags and this predefined special tag is that the previous tags are handled differently during decoding of the code.

Unsigned int can be represented by a separate class. So, we construct the tags as <Lib_Unsigned> where the Lib prefix means that the source library needs to be referred while converting the code into the destination language. Apart from this, overloading is not allowed.

This is how the language can be prepared.

Designing a converter includes designing the intermediate language and designing the conversion library.

In C++ and Java, various features have been implemented differently. For example, primitive data types such as unsigned int and Boolean. To handle this, we can implement a class for unsigned int and Boolean values. Similarly for taking input in C++ we have the same format for all data types. For example *"cin>> x;"* works for all the possible data types of x. The same is not true for Java. For each type of data type we have a separate function to take user input. Thus it will be better to have a class that can help us to take user level input. While taking the user level input we need to only call the method. In this way the library can be designed to handle these kinds of issues.

Thus, as explained in the main idea, we can have the language converter where language converter will keep all the keywords as tags similar to XML tags. These tags are also used to define the block. Comments can also be represented using these tags.

### *7.3.2 Issues*

The issues that may arise during the conversion process are:

a. Pointers:

Pointers in C++ cannot be completely represented in Java but can be approximated by using references. As long as the reference lies within the boundary of the program, both have equal powers. Thus pointers can be implemented.

b. Pre-processor:

Pre-processor mainly includes "# define" & "# include" directives. The "# define" pre-processor directive is C++ can be replaced by public final & static variable in java. For example-

*# define MAX 10*

can be converted as

*public final static MAX = 10;*

c. Destructors:

Destructors are not present in java. Instead, we have the *finalize* method associated with each class. Generally, *finalize* is called just before an objects' memory is about to be set free. It is similar to destructor but not exactly equivalent to it. Using the *finalize* method to replace destructors may affect the performance in some cases.

d. Friend Function:

They can be implemented by using composition. Create a new class having the instance of each of the desired class. And write a function that can access both the members. Even though this is not a perfect solution, it will work in most cases.

e. Operator Overloading:

This can be implemented by adding a special name to the operator and then replacing all the operator calls by its function call. Care should be taken that the name of function should not match that of any previously defined function.

f. Multiple Inheritances:

This is impossible to achieve. The only option is to use interfaces.

Thus, these are some of the issues that may crop up during inter-conversion and how the library should manage these issues for the conversion.

## 8.FUTURE DEVELOPMENT

In the future, it may even be possible to convert code between two completely different platforms at the click of a button. This may depend on advancements in the field of Artificial Intelligence because a library alone can not do such a conversion perfectly and recognize the replaceable pattern in the language. For e.g., converting a desktop application to a web application and vice versa. Similarly it may be possible to create mobile applications from the logic used to build a similar desktop or web application.

## 9. CONCLUSION

Achieving the maximum efficiency of conversion without compromising the quality of converted system is the programmers' dream. Even though language conversion might seem to be easy, it is actually a Herculean task with many different complications. It has been rightly placed among the top 10 challenges before the programming world. A lot of progress needs to be made even before a reliable semi-automatic convertor becomes available.

As a solution to this problem, we have proposed the creation of an intermediate language which not only stores the code in an algorithmic manner, but also maintains the program structure. For this to be practical, we may have to implement a set of specific standards for each major programming language so as to enhance its convertibility. Finally separate convertors to and from the intermediate language have to be created for each language.

This is not an easy task, but if correctly implemented, it would greatly change the future of software development. It would simplify the process of developing programs, maintaining them and hence bring down costs tremendously.

## 10.REFERENCES

[1] Guarded commands, non-determinacy and formal derivation of programs (White Paper) by Edsger K. Dijkstra

[2] The Realities of Language Conversions by A. A. Terekhov and C. Verhoef (St. Petersburg University)

[3] Code Migration through Transformation: An Experience Report by K. Martin and J. Wong

[4] Three Tools for Language Processing: BNF converter, Functional Morphology and Extract by Markus Forsberg , Markus Forsberg , C Markus Forsberg , Ny Serie Nr

[5] Perthon available at: http://freshmeat.net/projects/perthon

[6] BCX was originally started by Kevin Diggins. This project is a direct continuation of his great efforts. It is now completely open source and developed by a group. Available at: http://bcx-basic.sourceforge.net

[7] The Microsoft Java Language Conversion Assistant (JLCA) is a tool that automatically converts existing Java-language source code to C# for developers who want to move their existing applications to the Microsoft .NET Framework. Available at: http://support.microsoft.com/kb/819018

[8] Programming language available at: http://en.wikipedia.org/wiki/Programming_language

[9] C++0x (pronounced "see plus plus oh ex") available at: http://en.wikipedia.org/wiki/C++0x

[10] What 'bridgekeeper' does available at: http://www.crazy-compilers.com/bridgekeeper