JPT : A SIMPLE JAVA-PYTHON TRANSLATOR

Eman J. Coco, Hadeel A. Osman and Niemah I. Osman

College of Computer Science and Information Technology,
Sudan University of Science and Technology, Sudan

ABSTRACT

Java and Python are two of the most popular and powerful programming languages of present time. Both of them are Object-Oriented programming languages with unique advantages for developers and end users. Given the features of Python and how it is related to emerging fields in computer science such as Internet of Things, Python is considered a strong candidate of becoming the main programming language for academia and industry in the near future. In this paper, we develop JPT, which is a translator that converts Java code into Python. Our desktop application takes Java code as an input and translates it to Python code using XML as an intermediate language. The translator enables this conversion instead of having to rewrite the whole Python program from start. We address a number of cases where the translation process is challenging and highlight cases where manual inspection is recommended.

KEYWORDS

Compiler, Interpreter, Document Object Model, Translator

1. INTRODUCTION

Java [1] and Python [2] have recently emerged in the programming world, however, they both earned their place being among the most popular programming languages today. They both have many powerful features desired by programmers. Compared to Java, Python is an easier language for novice programmers to learn. One can progress faster if learning programming in Python as a first language, because Java is restrictive and more complex compared to Python. Python is more user-friendly, robust, easier to read and understand, has a more intuitive coding style and is easier to debug. It is also more productive than Java because it is a dynamically typed programming language whereas Java is statically typed. Python is stable and used in giant organizations including Philips, Google, NASA, US Navy and Disney [3].

There is no compelling evidence that Python will definitely replace Java in the near future. However, given the features of Python above, and how it is related to emerging fields in computer science such as Internet of Things (IoT), it is a strong candidate to dominate in both academia and the software market. A shift from one programming language to another is not an overnight process, and is considered a tedious job for all. If programmers want to translate their software programs from Java to Python to gain its features, they will have to rewrite the whole program from start which consumes time and increases cost. Therefore, a mechanism that translates programs from Java to Python automatically is necessary. Program conversion process has been placed among the top 10 challenges in the programming world [4]. Achieving the maximum efficiency of the conversion without compromising the quality of the translated program is the programmer's target.

The work in [5] presents an approach for programming language inter-conversion which can be applied to all types of programming languages. They implement an intermediate language for

DOI: 10.5121/caij.2018.5201

inter-conversion that can be used to store the logic of the program in an algorithmic format without disturbing the structure of the original program. Separate translators to and from the intermediate language however have to be created for each language. There is a number of programming language translators available online [6]-[11], some are web-based such as [7] while others are desktop applications as [8] and [11]. In addition, several are free, for instance [6],[9] and [11], but only a few are open source including [6] and [10]. These translators convert between a number of programming languages including Visual Basic, C, C++, C#, Java, Ruby, Python, Perl and Boo. Nevertheless, only two of these translators convert Java to Python, these are [6] which is compatible with older versions of Python, and [7] which is free only to a limited number of characters.

In this paper, we develop a simple Java-Python translator that takes a Java file code as input and translates it to Python file code as output. The objective of this work is to analyze the conversion process considering the similarities and differences between the two languages. Providing an open source translator which discloses conversion steps from source to intermediate to target language enables academics and professionals to gain more insight on how to best modify code such that it is error free after conversion. In addition, this work will contribute in the possible switch from Java to Python by helping reduce the software evolution cost as well as help Java programmers to learn Python.

The Simple Java-Python translator covers the basic principles of programming languages. We consider class and method declaration, comments, declaring and initializing primitive, floating point and boolean variables and all selection and iteration statements. The translator reads Java statements from the Java program, converts them to eXtensible Markup Language (XML) tags as an intermediate code and writes them in a .xml file. It then reads XML tags, converts them to Python statements and writes them in a Python file.

The rest of this paper is organized as follows. Section 2 describes Java and Python syntax and explains language processors (compiler and interpreter). The system design is illustrated in Section 3. Section 4 demonstrates the implementation of the translator and provides execution examples. Finally, Section 5 is conclusions.

2. JAVA AND PYTHON SYNTAX AND LANGUAGE PROCESSING

Java is a direct descendant of C, from which it inherits many Object-Oriented features. The primary motivation to develop Java was the need for a platform-independent language that is used to create software that can run on various machines. Java gained popularity with the emergence of the World Wide Web, as the Web required portable programs. Java is robust, multithreaded and distributed. In addition, it is portable, and was designed such that one can write code once, run it anywhere, anytime, forever [1].

Python programming language was created in the late 1980s and is a higher-level programming language. It is considered to be a higher-level language than C, C++, Java, and C#. Python is considered an easy language for beginning programming because it has clear syntax and it is easy to write programs using it [2].

Following, we briefly display Java and Python basic syntax that is considered in this work. We also highlight language processors and their relevance to Java and Python.

2.1 Java and Python Basic Syntax

Here we compare Java and Python in terms of syntax as to gain basic understanding of what the translator is expected to do. Table 1 compares Java and Python in terms of comments, variables, data types and statements.

Table 1 Java vs. Python Syntax

	Java	**Python**
Print statement	System.out.println("Welcome to java");	print("Welcome to python")
Comments	// line comment /* paragraph comment */ /** javadoc comment */	#comment
Declaring Variables	Datatype variable_Name = value;	variable_Name = value
Primitive Data Types	byte, short, int, long, float, double, char, boolean.	int, long, float, complex, boolean.

If statement	if (condition) {
	Statement;
	}
		If condition:
		Statements

If else statement	if (condition)
	{ Statement;
	} else {
	Statement;
	}
		If condition:
		Statements
		else:
		Statements

Nested if statement	if (condition)
	{ Statement;
	}//end outer if
	else
	{
	if{
	Statement;
	} //end inner if
	else{
	Statement;
	}//end inner else
	}//end outer else
		If condition:
		Statements
		else:
		if condition:
		Statements
		else:
		Statements

Switch statement	switch (expression)
	{
	case value1: statements;
	break;
	case value2: statements;
	break;
	default: statements;
	}//end switch
		Doesn't have a switch statement

While statement	while (condition)
	{
	// body of loop
	}
		while condition:
		# *block*

for Statement	for (initialization; condition; iteration)
	{
	// body
	}
		for n in range (begin, end, step):
		# block

2.2 Compilers and Interpreters

Language processing is achieved by one of two approaches (or both): compiler and interpreter.

2.2.1 Compilers

A compiler is a program that can read a program in one language (the source language) and translate it into an equivalent program in another language (the target language). The compiler consists of two parts: analysis and synthesis. The Analysis divides the source code into pieces, applies the grammatical structure to them and generates an intermediate representation of the source code. If the syntax of the source code is ill, it generates an informative message to the user. It also collects information about the source code and stores them on the symbol table. The synthesis part uses a symbol table and intermediate representation to generate the target program [12].

2.2.2 Interpreters

An interpreter is another common kind of language processors that directly executes the source program on user inputs. The task of an interpreter is more or less the same as of a compiler but the interpreter works in a different fashion. The interpreter takes a single line of code as input at a time and executes that line. It will terminate the execution of the code as soon as it finds an error. Memory requirement is less because no object code is created.

The machine language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement [12].

2.3 Java and Python Languages Processors

2.3.1 Java

Java is both a compiled and interpreted language. When a Java program is written, the *javac* compiler converts the program into bytecode. Bytecode compiled by *javac*, is entered into the Java Virtual Machine (JVM) memory where it is interpreted by another program called *java*. This *java* program interprets bytecode line-by-line and converts it into machine code to be run by the JVM [13].

2.3.2 Python

There are four steps that python takes when the return key is pressed: lexing, parsing, compiling, and interpreting. Lexing is breaking the line of code into tokens. The parser takes those tokens and generates an Abstract Syntax Tree (AST) which is a structure that shows the relationship of tokens. The compiler then takes the AST and turns it into one (or more) code objects. Finally, the interpreter takes each code object and executes the code it represents.

3. DESIGNING THE JAVA-PYTHON TRANSLATOR

In computing, a translator (or converter) is a computer program that takes a file written in a specific language or format and transforms it into another format without losing the functional or logical structure of the original file [12]. Following, we explain the design of our translator and the translation process.

3.1 System Description

The Simple Java Python Translator reads Java statements from the Java file, converts them to XML tags and writes them in a scripting file (.xml). Then it reads XML tags, converts them to Python statements and writes them in a Python file as shown in Figure 1.

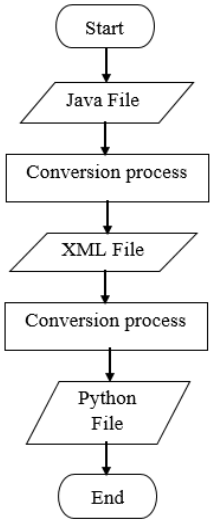

Figure. 1. Translation Stages

3.1.1 Reasons for using an intermediate language

The main reason of using an intermediate language is to facilitate the process of conversion by extracting the basic components of each statement on which programming languages depend to build their own statements. Also it can be used to facilitate the process of translation to other programming language besides Python without the need to start from the beginning and repeat the translation process from Java. This can be achieved after some modifications associated with the structure of the language to which the code is translated.

3.1.2 Selecting XML as an intermediate language

XML was designed to carry data and to be both human- and machine-readable. The reasons why we select XML as an intermediate language are:

- One of the most time-consuming challenges for developers is to exchange data between incompatible applications. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

- XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

- With XML, data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc.).

3.2 Translation Process

The translation process goes through two phases. In the first phase the Java file is converted to an XML file and in the second phase the XML file is converted to a Python file.

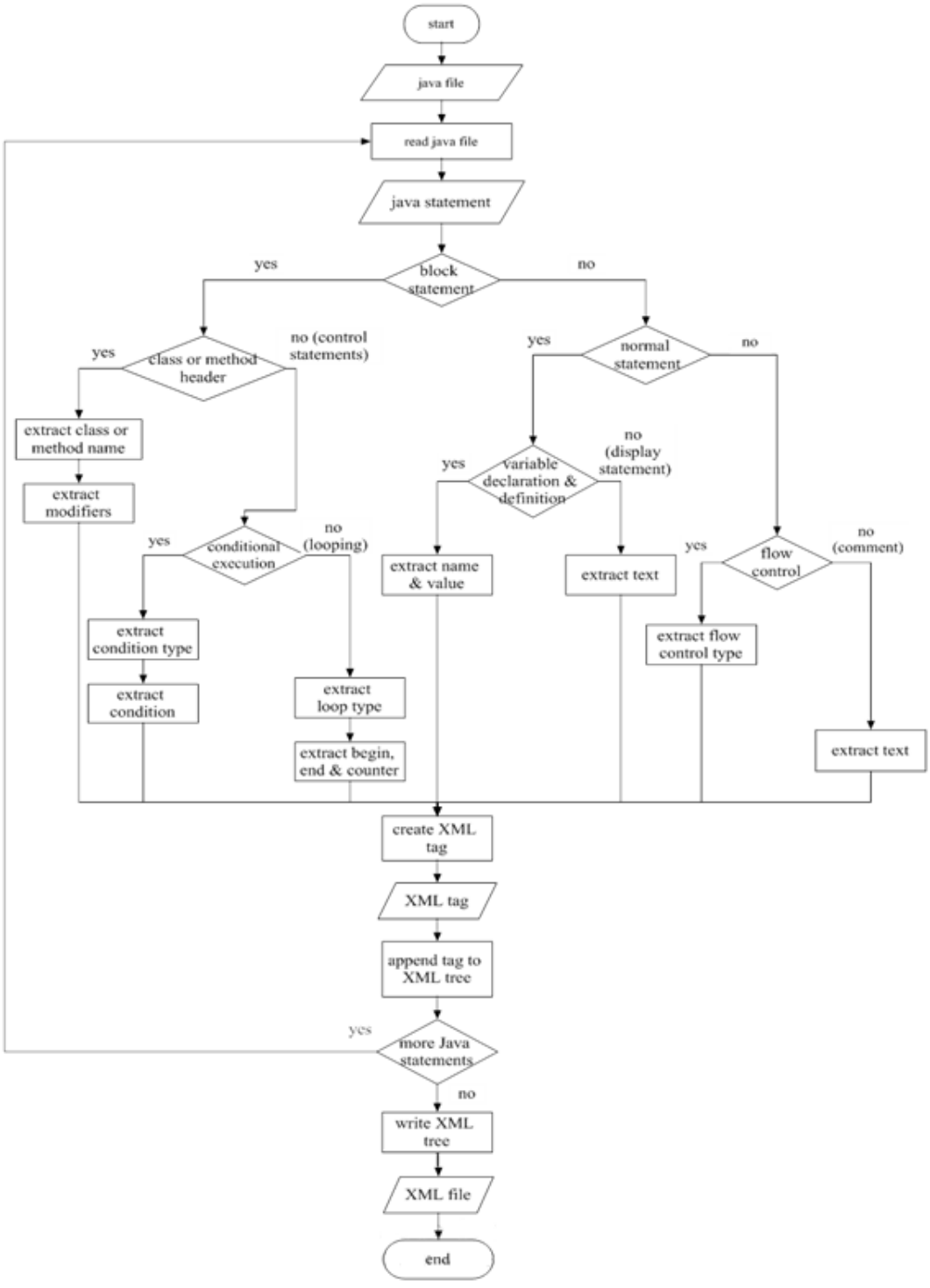

Figure. 2 Java to XML Phase Flowchart

3.2.1 The Java to XML Phase

The translator reads the Java file character by character to extract the type and components of the Java statements. Each Java statement is converted to an appropriate XML tag; the tag name is

determined based on the type of Java statement. The components of the Java statement are stored in the tag attributes.

When an XML tag is created the translator appends it to the XML Document Object Model (DOM) tree. The conversion process will continue until the end of the Java file. Then the translator transforms the DOM tree to an XML file as shown in Figure. 2.

Table. 2 describes XML tags names, attributes assigned to each one and if the tag has a value or body. The tag name determines the type of the Java statement and the tag attributes determine the components of the Java statement.

3.2.2 The XML to Python Phase

The translator reads the XML file to extract the DOM tree and then it reads the tree nodes which are XML tags. Each XML tag is converted to an appropriate Python statement; the type of Python statement is determined based on the tag name. The components of the Python statement are extracted from the tag attributes.

The translator writes the Python statement in a Python file. The conversion process will continue until the end of the DOM tree. The steps of this phase are shown in Figure. 3Figure.

Table. 2 XML Tags Components

Tag Name	Tag Attributes								Value	Body
	access	identifier	static	type	condition	begin	end	counter		
Class	✓	✓								✓
Comment				✓					✓	
Method (method header)	✓	✓	✓	✓						✓
param (method parameter)		✓		✓						
call (method call)									✓	
return (method return)									✓	
If					✓					✓
Else										✓
For						✓	✓	✓		✓
while (also do…while)					✓					✓
Continue	Doesn't have tag attributes, value or body									
Break	Doesn't have tag attributes, value or body									
var (variables declaration and definition)	✓	✓	✓	✓					✓	
Display									✓	

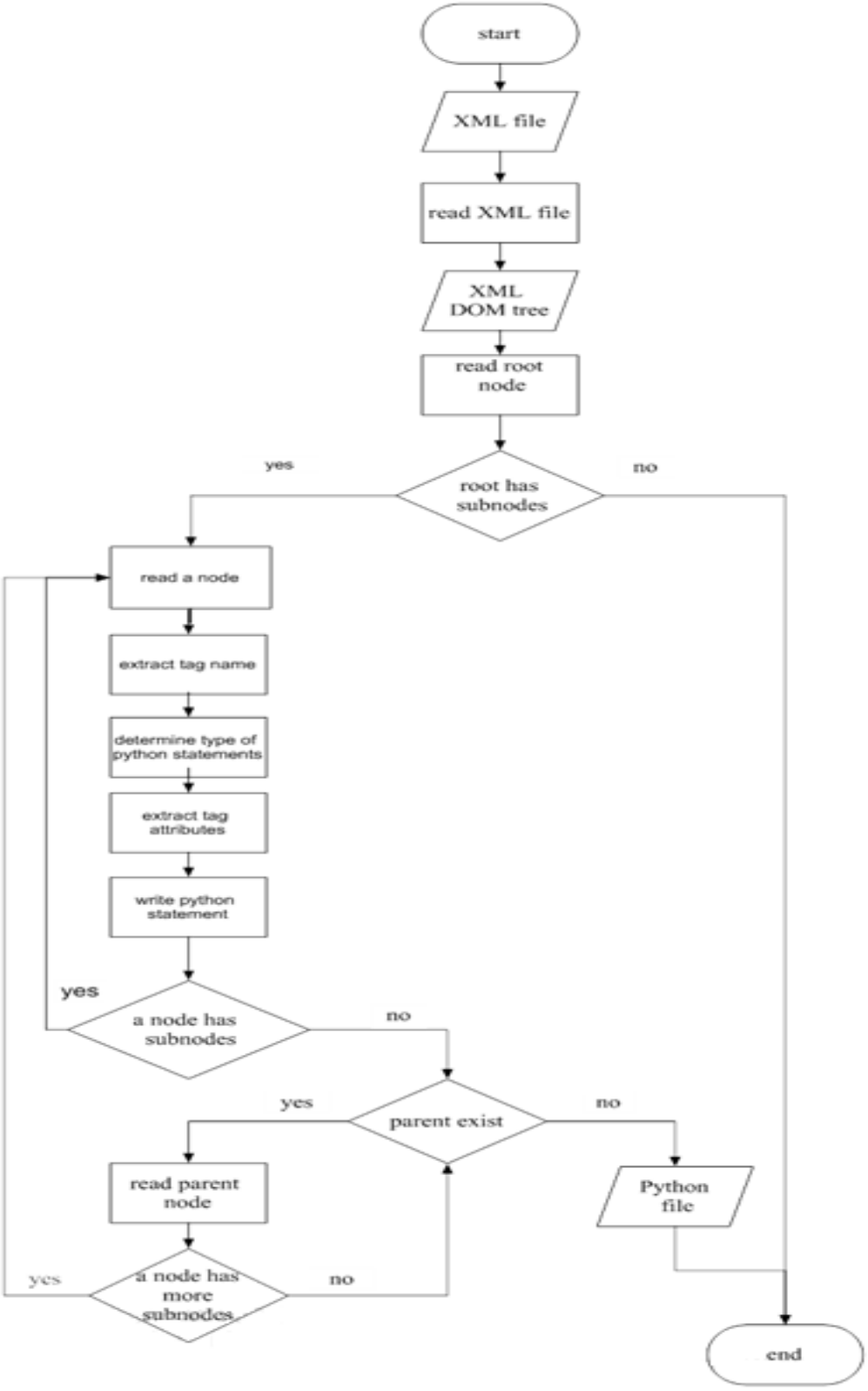

Figure. 3 XML to Python Phase Flowchart

3.3 Translation Example

Figure. 4 (a) is an input Java code to the translator, Figure. 4 (b) illustrates intermediate XML code and Figure. 4 (c) is the output Python code from the translator.

```java
public class Demo{
    //converter
    public static void main(String args[]){
        int x = 10;
        while( x > 0){
            System.out.println(x);
            if( x == 5)
                break;
            else
                x--;
        }
    }
}
```

(a)

```python
import sys
# converter
def main(args):
        x = 10
        while x>0:
                print(x)
                if x==5:
                        break
                else:
                        x = x-1
main("args")
```

(c)

```
- <root>
  - <class access="public" identifier="Demo">
      <comment type="line"> converter</comment>
    - <method access="public" identifier="main" static="true" type="void">
        <param identifier="args" type="String[]"/>
        <var access="" identifier="x" static="false" type="int">10</var>
      - <while _if="1" condition="x>0">
          <display nl="true">x</display>
        - <if _if="2" condition="x==5">
            <break/>
          </if>
        - <else _if="2">
            <var access="" identifier="x" static="false" type="">x-1</var>
          </else>
        </while>
      </method>
    </class>
  </root>
```

(b)

Figure. 4 Sample Input & Output From the Translator: (a) the Java Input, (b) the Intermediate XML Code and (c) the Python Output

4. RUNNING THE TRANSLATOR

The Java Python Translator is a desktop application that is developed using Java. It has been tested on Windows and Linux Operating Systems. This section demonstrates examples of input and output of the translator. It also discusses the challenges facing the translator to attain accurate output.

4.1 Simple Program

Figure. 5 (a) is an input Java code to the translator and Figure. 5 (b) is the output Python code from the translator.

This code covers comments, variables declaration and definition, type casting and displays on the screen. We make the following notes:

- A Javadoc comment is converted to a multiple line comment because Python doesn't have doc comments.
- Reserved words in Python should not be used as variable names in the Java code because this causes a problem with the Python code.
- Python does not allow the declaration of variables without assigning values to them. So if a variable is declared in Java with no initial value, the translator assigns zero to the variable in Python.
- In the case of declaration or definition of more than one variable in the same statement in Java, the translator processes each variable separately and each variable has its own line in Python because it is not possible to define more than one variable in the same statement.
- When concatenating a string with another data type, by default Java considers it as a string, but Python produces an error, so the translator casts the other data type to string.
- Casting String to integer or string to float is out of the scope of the translator because Java uses specific classes for casting: Integer.parseInt("string") or Double.parseDouble("string").
- When casting to short or long the translator converts to integer and also converts double to float; because Python does not contain the data types short, long and double. A more complex solution would be to store a larger variable in two smaller ones as not to lose the value of the large variable.
- Python print() statement prints a new line by default at the end of the statement, so when System.out.print() is used in the Java code, the converter adds(end="") to the end of the print() Python statement.

4.2 Methods

Figure. 6 shows an example of method conversion where Figure. 6 (a) is an input Java code to the translator and Figure. 6 (b) is the output Python code from the translator.

By default, the execution of the program in Java starts from the main method. Therefore, the translator invokes the main method in the Python code if found in the Java code, because Python does not require the existence of the main method for execution.

Unlike Java, Python doesn't accept an empty block. So in case of an empty method body the converter writes print(end="") to act as the method body.

```java
/**
This is javadoc comment
*/
public class Demo{
    public static void main(String args[]){
        /*this is paragraph comment
        */
        /*Don't use python reserved word
        as variables name
        */
        int x;
        short y, z;
        double d = 3.5;
        x = (int) d;
        y = (short)(d + 3);
        String s ="x = " + x;
        System.out.println(s);
        char ch = (char) 65;
        System.out.println("ch = "+ch);
        float f = (float) x;
        d = (double) f+y;
        System.out.print(f);

    }
}// this is line comment
```

(a)

```python
''' *
        This is javadoc comment
        '''
import sys
def main(args):
        ''' this is paragraph comment
                        '''
        ''' Don't use python reserved word
                        as variables name
                        '''
        x = 0
        y = 0
        z = 0
        d = 3.5
        x = int(d)
        y = int(d+3)
        s = str("x = ")+str(x)
        print(s)
        ch = chr(65)
        print(str("ch = ")+str(ch))
        f = float(x)
        d = float(f)+y
        print(f,end="")
main("args")
#  this is line comment
```

(b)

Figure. 5 Simple Example: (a) Java Input Code and (b) Python Output Code

```java
public class Demo{
        static void method1(){
        }
        public static void method2(int x, int y){
            System.out.println("x = "+x);
            System.out.println("y = "+y);
        }
        protected static double method3(){
            return 5.0;
        }
        public static String method4(String name){
            return name;
        }
        public static void main(String [] args){
            method1();
            double x = method3();
            System.out.println(method4("Converter"));
        }
}
```

(a)

```python
import sys
def method1():
        print(end="")
def method2(x,y):
        print(str("x = ")+str(x))
        print(str("y = ")+str(y))
def method3():
        return 5.0
def method4(name):
        return name
def main(args):
        method1()
        x = method3()
        print(method4("Converter"))
main("args")
```

(b)

Figure. 6 Method Example: (a) Java Input Code and (b) Python Output Code

4.3 Selection statements

Selection statements include the if statement, the if..else statement and the nested if.

4.3.1 The if statement

Figure. 7 is an example of if statement where Figure. 7 (a) is an input Java code to the translator and Figure.7 (b) is the output Python code from the translator.

When the translator finds if or else statements without a body it writes print(end="") in Python to act as the body.

```java
public class Demo{
    public static void main(String args[]){
        double d = -3.5;
        if( d > 0 ){
            System.out.println("positive");
        }
        if( d < 0 )
            System.out.println("negitive");
        if( d == 0 );
    }
}
```

(a)

```python
import sys
def main(args):
        d = -3.5
        if d>0:
                print("positive")
        if d<0:
                print("negitive")
        if d==0:
                print(end="")
main("args")
```

(b)

Figure. 7 if Example: (a) Java Input Code and (b) Python Output Code

4.3.2 The if...else statement

Figure. 8 shows the if...else statement where Figure. 8 (a) is an input Java code to the translator and Figure. 8 (b) is the output Python code from the translator.

```java
public class Demo{
    static int x;
    public static void main(String args[]){
        x = 5;
        if(x%2==0){
            System.out.println(x+" Is An Even Number");
        }
        else{
            System.out.println(x+" Is An Odd Number");
        }
    }//end of main method
}//end of class
```

(a)

```python
import sys
x = 0
def main(args):
        x = 5
        if x%2==0:
                print(str(x)+str(" Is An Even Number"))
        else:
                print(str(x)+str(" Is An Odd Number"))
main("args")
# end of main method
# end of class
```

(b)

Figure. 8 if...else Example: (a) Java Input Code and (b) Python Output Code

4.3.3 Nested if statement

Figure 9 (a) is an input Java code to the translator and Figure. 9 (b) is the output Python code from the translator. This code illustrates nested if statements in Java and Python.

```java
public class Demo{
    public static void main(String args[]){
        int degree = 76;
        char grade;
        if (degree >= 90)
            grade = 'A';
        else
            if (degree >= 80)
                grade = 'B';
             else
                if (degree >= 70)
                    grade = 'C';
                 else
                    if (degree >= 60)
                        grade = 'D';
                    else
                        grade = 'F';
        System.out.println("Grade = " + grade);
    }//end of main method
}//end of class
```

(a)

```python
import sys
def main(args):
        degree = 76
        grade = 0
        if degree>=90:
                grade = 'A'
        else:
                if degree>=80:
                        grade = 'B'
                else:
                        if degree>=70:
                                grade = 'C'
                        else:
                                if degree>=60:
                                        grade = 'D'
                                else:
                                        grade = 'F'
        print(str("Grade = ")+str(grade))
main("args")
# end of main method
# end of class
```

(b)

Figure. 9 Nested if Example: (a) Java Input Code and (b) Python Output Code

```java
public class Demo{
    public static void main(String args[]){
        char ch = 'b';
        char ch2 = 'h';
        switch(ch){
            case 'a':
               System.out.println("case 'a'");
               break;
            case 'b':
                switch(ch2){
                    case 'x':
                       System.out.println("case 'x'");
                       break;
                    default:
                       System.out.println("inner default");
                       break;
                }
               break;
            default:
               System.out.println("outer default");
        }
        System.out.println("outer");
    }
}// end of class
```

(a)

```python
import sys
def main(args):
        ch = 'b'
        ch2 = 'h'
        if ch=='a':
                print("case 'a'")
        else:
                if ch=='b':
                        if ch2=='x':
                                print("case 'x'")
                        else:
                                print("inner default")
                else:
                        print("outer default")
        print("outer")
main("args")
#  end of class
```

(b)

Figure. 10 Switch Example: (a) Java Input Code and (b) Python Output Code

4.4 The switch statement

It is known that Python does not contain the switch statement in its structure as Java. Even though there are many ways to represent the Java switch statement in Python code, it does not cover all cases. The translator converts the Java switch statement to nested…if statements in Python, because it is clear, simple and easy to understand. It is worth noting that even with no break statement found in the switch statement the conversion process is done assuming its existence.

Figure. 10 (a) is an input Java code to the translator illustrating the switch statement in Java and (b) is the corresponding nested..if statements Python code.

4.5 Iteration statements

The iteration statements are the while, the do..while and the for loop.

4.5.1 The while statement

When the converter finds a while statements without a body it writes a continue statement to act as a body. This is shown in Figure. 11 (a) and (b).

```
public class Demo{
    public static void main(String args[]){
        int base = 3, power = 2;
        while( power > 1){
            base *= base;
            power--;
        }
        System.out.println("result = "+base);

        int x = 0;
        while( x < 10 )
            x++;
        System.out.println("x = "+x);

        while( x == 0);
    }//end of main method
}//end of class
```

(a)

```
import sys
def main(args):
        base = 3
        power = 2
        while power>1:
                base = base*(base)
                power = power-1
        print(str("result = ")+str(base))
        x = 0
        while x<10:
                x = x+1
        print(str("x = ")+str(x))
        while x==0:
                continue
main("args")
```

(b)

Figure. 11 while Example: (a) Java Input Code and (b) Python Output Code

4.5.2 The do…while statement

Python does not have a do…while loop, therefore the converter converts it to a while loop with the following steps:

- The translator sets the condition of the while loop to True.
- It then appends an if…else statement to the body of the do…while loop with a continue statement in the if body and a break statement in the else body. If condition represents a do…while condition.

The do..while Java input and Python output are shown in Figure. 12 (a) and Figure. 12 (b), respectively.

```java
public class Demo{
    public static void main(String args[]){
        int prevPrevVal = 0;
        int prevVal = 1;
        int currVal;
        //Fibonacci
        do{
            System.out.print("   ");
            currVal = prevVal + prevPrevVal;
            System.out.print(currVal);

            prevPrevVal = prevVal;
            prevVal = currVal;
        }while(prevVal <= 10);
        System.out.println();
    }//end of main method
}//end of class
```

(a)

```python
import sys
def main(args):
        prevPrevVal = 0
        prevVal = 1
        currVal = 0
        # Fibonacci
        while True:
                print("   ",end="")
                currVal = prevVal+prevPrevVal
                print(currVal,end="")
                prevPrevVal = prevVal
                prevVal = currVal
                if prevVal<=10:
                        continue
                else:
                        break
        print()
main("args")
# end of main method
# end of class
```

(b)

Figure. 12 The do...while Example: (a) Java Input Code and (b) Python Output Code

4.5.3 The for statement

There are many differences in syntax and logic between Java and Python with respect to the for statement. Therefore, the translator makes important changes to eliminate these differences. These differences include:

- Python does not allow defining variables in the header of the for statement as Java.
- The index of the for statement in Java holds the start value, while in Python a variable separate from the index variable holds the start value.
- Start and end values should be explicitly mentioned in the Python for statement.
- The for loop in Java repeats until the end value but in Python until the end value minus one. For example: if a for loop starts at 0 and ends at 5, in Java this for loop repeats 6 times (0,1,2,3,4,5) while in Python it repeats 5 times (0,1,2,3,4). Therefore, when the start value is assigned to the index of the Java for statement in the header, the translator defines it before the Python for statement.
- The translator adds a public variable to reserve the index space and uses the variable which holds the start value as an index. It increases (or decreases) the index at the end of the for statement body and adds an if…else statement to break the loop if the condition is false. The body of the for statement becomes the body of the if statement.

The code shown in Figure. 13 covers the for statement, where Figure. 13 (a) is the Java input code and Figure. 13 (b) is the Python output code.

```java
public class Demo{
    public static void main(String args[]){
        for( int i = 0; i < 10; i+=2){
            System.out.print(i+" ");
        }
        System.out.println("\n");
        int j = 0, x = 5;
        for(; j <= 10;){
            System.out.print(j+" * "+x);
            System.out.println(" = "+(j*x));
            j++;
        }
    }
}
```

(a)

```python
import sys
def main(args):
    i = 0
    for public in range(i,10,1):
        if i<10:
            print(str(i)+str(" "),end="")
        else:
            break
        i = i+(2)
    print("\n")
    j = 0
    x = 5
    for public in range(j,10+1,1):
        if j<=10:
            print(str(j)+str(" * ")+str(x),end="")
            print(str(" = ")+str((j*x)))
            j = j+1
        else:
            break
main("args")
```

(b)

Figure. 13 The for loop Example: (a) Java Input Code and (b) Python Output Code

- When the repetition condition is not clarified in the Java for statement, the translator considers the condition to be True and the end value of the Python for loop is set to the maximum value of the integer type to prevent errors. Figure. 14 (a) and Figure. 14 (b) show this conversion.
- When the header of the Java for loop is empty the translator represents it as a while True statement. This is also shown in Figure. 14.

```java
public class Demo{
    public static void main(String args[]){
        int i;
        for(i = 0; ; i+=2){
            System.out.println(i);
            break;
        }
        for(;;){
        }
    }
}
```

(a)

```python
import sys
def main(args):
        i = 0
        i = 0
        for public in range(i,sys.maxsize,1):
                if True:
                        print(i)
                        break
                else:
                        break
                i = i+(2)
        while True:
                continue
main("args")
```

(b)

Figure. 14 Empty for Example: (a) Java Input Code and (b) Python Output Code

5. CONCLUSIONS

This paper has presented a simple Java Python translator that converts Java code to Python code using XML as an intermediate language. The translator covers class and method declaration,

comments, declaring and initializing primitive, floating points and boolean variables, selection statements, switch statement and iteration Statements.

After designing, implementing and testing the translator, we found that it successfully converts the syntax of Java programs to Python without having to rewrite Python programs from start. We recommend checking the Python program to make sure that the logic is correct. As future work, we would like to extend the translator to include Object Oriented Programming (OOP) principles, arrays and data structures.

REFERENCES

[1] Naughton, P., & Schildt, H., (1996) "Java: the complete reference". Osborne, McGraw-Hill.

[2] Halterman, R. L., (2011) "Learning to program with python", Richard Halterman.

[3] Fangohr, H., (2004) "A Comparison of C, MATLAB and Python as Teaching Languages in Engineering", Lecture Notes on Computational Science, pp. 1210-1217, 2004.

[4] Terekhov, A. A., & Verhoef, C., (2000) "The realities of language conversions". IEEE Software, vol. 17, no. 6, pp. 111-124.

[5] George, D., Girase, P., Gupta, M., Gupta, P., & Sharma, A., (2010) "Programming Language Inter-conversion". International Journal of Computer Applications, vol. 1, no. 20.

[6] java2python, [online], available at:https://github.com/natural/java2python, date accessed: 15/4/2017.

[7] Varycode, [online], available at: http://www.thetaranights.com/varycode/, date accessed: 3/10/2017.

[8] Tangible Software Solutions, [online], available at: https://www.tangiblesoftwaresolutions.com/index.html, date accessed: 6/10/2017.

[9] JLCA, [online], available at:http://support.microsoft.com/kb/819018, date accessed: 9/10/2017.

[10] BCX, [online], available at:http://bcx-basic.sourceforge.net/, date accessed: 3/10/2017.

[11] Perthon, [online], available at: http://freshmeat.net/projects/perthon, date accessed: 5/10/2017.

[12] Aho, A. V., Lam, M., Sethi, R., & Ullman, J. D., (2006) "Compilers: Principles, Techniques, and Tools", Addison-Wesley, Second Edition.

[13] Brown, P. J, (1979) "Writing interactive compilers and interpreters", Wiley Series in Computing, Chichester: Wiley.

AUTHORS

Eman Jammaa Coco received the B.Sc. degree (first class honours) in Computer Systems and Networks from Sudan University of Science and Technology, Khartoum, Sudan, in 2017. She is currently a Computer Programmer at Nile Technical Research Center. Her current interests include ERP using Odoo and Linux administration.

Hadeel Ali Osman received the B.Sc. degree (first class honours) in Computer Systems and Networks from Sudan University of Science and Technology, Khartoum, Sudan, in 2017. She is currently a Teaching Assistant in Sudan University of Science and Technology. Her current interests include Odoo and design of Web applications.

Niemah Izzeldin Osman received the B.Sc. degree (first class honours) in Computer Science from Sudan University of Science and Technology, Khartoum, Sudan, in 2002, the M.Sc. degree (with distinction) in Mobile Computing from the University of Bradford, U.K., in 2006 and the Ph.D. degree in Communication Networks from the University of Leeds, U.K in 2015. She is currently an Assistant Professor at the Department of Computer Systems and Networks, Sudan University of Science and Technology, Sudan. Her current research interests include performance evaluation of 4G LTE networks, Internet of Things and QoE of mobile video.