International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 02 | Feb 2019 www.irjet.net p-ISSN: 2395-0072

Divide and Conquer Algorithm for Multilingual Scripting Programming Language (MLProScript)

Osée Muhindo MASIVI

Computer and Management Information System, School of Business, Rusangu University, Zambia

---

Abstract - *"Divide and Conquer" has been proved to be one of the most suitable strategies for complex computations algorithm design. This paper presents "Divide and Conquer" application to design a simple, extensible and easy to implement parser algorithm. Results prove an improved, easy to implement model for multilingual programming language. The proposed algorithm splits the source code into smaller code sets easy to analyze and translate. Then the conquer algorithm part execute each code set using a simple interpreter. The last module combines code sets results and avails then to the target user program. A high level of multilingual programming language prototype for both programming language developers and end user programmers was built and tested.*

Key Words: Divide and Conquer, Parser, multilingual, scripting language.

1. INTRODUCTION

Two major techniques have been used to build parsing algorithms. The "bottom-up" based algorithms and he "top-down" based algorithms [1] [2]. Depending on the technique used by the parsing algorithm, two tools are intensively used to automatically generate the parser: YACC is used for LALR [2] parsers while the ANTLR [3] is a widely used tool to automatically generate parsers based on the LL(*) grammar [4] [5] [6]. The ANTLR has the particularity of being capable to generate the Recursive Descent Parser.

Beside the good time-complexity of the aforementioned algorithms, their implementation is not straightforward especially in scripting and end-user Programming languages [7]. These new trends in programming language development has introduced the concept of language extendibility which requires the possibility for the end user to add and use his own keywords and functionalities to the programming language. It is in this view that this article proposes an application of the "Divide and conquer" algorithm design strategy [7] to design a new parsing algorithm for a multilingual scripting language.

The proposed algorithm is intended to allow end-user to add his own non-English keywords and insert scripts anywhere in his code. The aim is to create a library which combines the lexicon creator, parser and the interpreter using Divide and Conquer famous algorithmic design pattern.

2. METHODOLOGY

To come with Multilingual Scripting Programming Language (MLProScript) integrated library, three major groups of classes are needed [8]: lexicon classes, parser classes and interpreter classes. Lexicon related classes allow the user to define, retrieve and analyze language keywords. Finite Automata techniques were used to create lexicon elements patterns.

Parser related classes are uses the generalized divide-and-conquer in an object oriented three steps [9] [10] [11] [12]. The first step is "Divide". This step divides the program source into program subsets (expressions or functions). The second step is "Conquer". This step recursively solves each sub problem using the first step and the associated trivial registered functions used as divide threshold. If the program subsets is simple action or a registered function, the problem solved by straightforward method and return the solution so obtained. Otherwise, divide the input data into two or more disjoint subsets. The last step is "Combine". This step takes the solutions to the sub problems and merge them into a solution to the original problem.

Compile related classes' main role is to initialize lexicon and parser classes, prepare and load end user code for lexical and parsing analysis compilation. After compilation, scripts results are given to the caller program.

3. RESULTS

3.1. Lexemes Creator and Analyzer

MLProScript is built on six major programming language concepts as suggested by [13] and [14]: character set, numbers, strings, identifiers, reserved words (operators, keywords and delimiters), expressions and statement. Keywords and Regular expression automata are stored in xml files (Listing 1.1).

```
<?xml version="1.0" standalone="yes"?>

<Grammar>

 <MoCle>

  <Name>Number</Name>

  <Pattern>^[0-9]+|^[0-9]+.[0-9]+</Pattern>
```

```
 </MoCle>
 <MoCle>
  <Name>String</Name>
  <Pattern>^\"[^\r\n\"]*\"</Pattern>
 </MoCle>
 <MoCle>
  <Name>Identifiant</Name>
  <Pattern>^[_A-Za-z][A-Za-z0-9]*</Pattern>
 </MoCle>
 <MoCle>
</Grammar>
<?xml version="1.0" standalone="yes"?>
<Lexicon>
 <Francais>
  <Type>test</Type>
  <Cle>if</Cle>
  <Mot>si</Mot>
 </Francais>
…
</Lexicon>
```

Listing1.1: Grammar and Lexicon xml files sample

The end user is allowed to create (Listing 1.1) and modify his own keyword in his own language.

```
public void setElement(string lang, string file, string type, string mot, string cle){
try{
DataSet dict = new DataSet();
dict.ReadXml(file);
DataTableCollection dtb = this.getAllElemt("dictKeywords.xml");
bool exist = false;
for (int i = 0; i < dtb.Count;i++ ){
        if (dtb[i].TableName == lang){
                exist = true;
                break;
        }
}
if (exist){
   DataRow dr = dict.Tables[lang].NewRow();
   dr["Type"] = type;
        dr["Mot"] = mot;
        dr["Cle"] = cle;
        dict.Tables[lang].Rows.Add(dr);
        dict.WriteXml(file);
}
else{
        DataTable newLang = new DataTable(lang);
        newLang.Columns.Add("Type", typeof(System.String));
        newLang.Columns.Add("Mot", typeof(System.String));
        newLang.Columns.Add("Cle", typeof(System.String));
        dict.Tables.Add(newLang);
        DataRow dra = dict.Tables[lang].NewRow();
        dra["Type"] = type;
        dra["Mot"] = mot;
        dra["Cle"] = cle;
        dict.Tables[lang].Rows.Add(dra);
        dict.WriteXml(file);
        }
}
catch (Exception ex)
{
```

```
Console.WriteLine("Error" + ex.Message);
}
}
```

Listing1.2: XML file reading

The xml files are used to feed the scripting language (Listing 1.3) with grammar (numbers, strings, identifiers), reserved words (operators, keywords and delimiters) including their translations in local languages.

```
public class Lexeure
  {
    ...
    static private Dictionary<string, string> MyAutomata;
    static private Dictionary<string, string> MyKeyWords;
  …
    public static void initial()
    {//Feed dictionary
      MyAutomata = new Dictionary<string, string>();
      MyKeyWords = new Dictionary<string, string>();
      try{
        LexiconElements lexEl = new LexiconElements();
        DataTable table = lexEl.getElemt("Grammar.xml");
        for (int i = 0; i < table.Rows.Count; i++)
        {
         MyAutomata.Add(table.Rows[i]["Name"].ToString()
        ,table.Rows[i]["Pattern"].ToString());
        }
        MyKeyWords = mots.charSpec;
      }
      catch (Exception ex) {
        throw new DataException(ex.Message);
      }
    }
```

Listing1.3: Lexicon feeding

Once source code is provided, the lexical analyzer reads the source code as a stream of characters and converts it into meaningful lexemes. Listing 1.3 shows part of MLProScript lexicon analysis.

```
private bool estNombre() { // Test Number
      motifNombre = MyAutomata["Number"];
       Match correspond =
      Regex.Match(source.Substring(index),
patternNombre);
       if (!correspond.Success){
         return false;
         int longueur = correspond.Length;
        string nombre = source.Substring(index, longueur);
        valToken = int.Parse(nombre);
         index += longueur;
        localise.accColonne += longueur;
         return true;
    }
    public string lexiconAnalyser()
    {    ...
      if (estOperateur(out tokenSuivant))
        return tokenSuivant;
      else if (estSeparateur(out tokenSuivant))
        return tokenSuivant;
      else if (estMotcleOuIdent(out tokenSuivant))
        return tokenSuivant;
      else if (estNombre())
        return "Nombre";
      else if (estTexte())
        return "Texte";
      valToken = source[index];
      index++;
      mondebut= index - 1;
```

```
        mataille = 1;
        monerreur = true;
        return "ERREUR";
    }
...
}
```

Listing1.4: Lexical analyzer

III.2. Divide and Conquer Parser

MLProscript Divide and Conquer algorithm is an adaptation of Split and merge technique in [15] using the algorithm 201 sketched bellow.

Algorithm 2.1: Div_Conquer

1. Input: sourceCode
2. Result=null
3. If (sourceCode is empty)
   Return null
4. ElseIf(notDividable(sourceCode))
   Return Result + Conquer(sourceCode)
5. Else
   Return Div_Conquer(sourceCode, start,end)

In the divide step (line 5), the source code is analyzed and into a set of small independent and executable codes from the start index to the end index. These sets can be single expressions, expressions in parentheses and known functions.

In the conquer step (line 4 last part) each independent and executable code is evaluated using target language straightforward expressions and statements or user defined and recorded functions. The combine step (line 4) takes place whenever a not divisible code set is reached. The conquered result is then added to the previous result.

Actions in the conquer combine use either target programming language operators (C# in our case).

III.3. MLProScript Interpreter

Interpreter classes are responsible for initializing defined keywords and their translations (from xml file), operators and functions. Listing 3.1 illustrates the loadNonEnglish function in the compiler.

```
Lexicon lex = new Lexicon();
DataTableCollection dtb
=lex.getAllElemt("dictKeywords.xml");
public void loadNonEnglish (DataTable nonEngl, string
engl){
    for (int i = 0; i < nonEngl.Rows.Count; i++){
        string newword;
        if (nonEngl.Rows[i]["Cle"].ToString() == engl){
            newword =
nonEngl.Rows[i]["Mot"].ToString();
            ….}
ParserFunction funct = ParserFunction.GetFunction(engl);
ParserFunction.AddGlobal(newword, funct);
…
}
}
```

Listint 1.1 Script interpretation

4. IMPLEMENTATION AND TESTS

4.1. Prototype Implementation

MLProScript was implemented using MS.NET 2012 (C#, Framework 4.5) resulting to a dynamic link library “MLProScript.dll”. In addition, a set of two xml files were created to store respectively multilingual keywords and operators (dictKeywords.xml) and lexemes automaton regular expression patterns (Grammar.xml).

4.2. Result Tests using the prototype

4.2.1. Lexemes definition

Fig. 1 shows the prototype of a how the user creates a lexeme in his local language. Once the programming language is created and the keywords are stored in the xml file which can be used to share the programming language with other programmers.

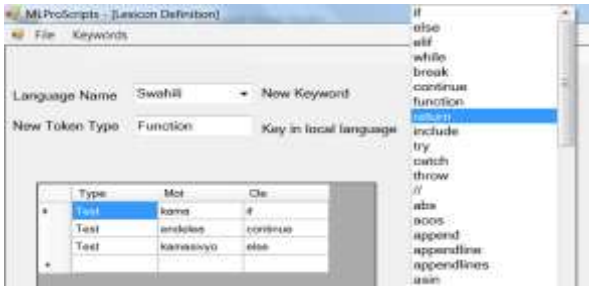

Fig. 1. Reserved word Definition prototype

4.2.2. Source code creation and execution

This paper has provided two ways of creating and running the scripts. In the first way, source code can be directly executed in the MLProScript Running Environment (MRE). The source code can be either directly typed in the environment or typed in a separate text file then loaded and executed. Fig.2. Shows a code directly typed and executed in MRE.

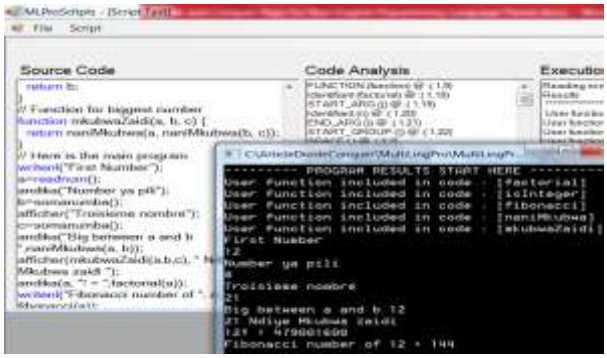

Fig. 2. MLProScript Running Environment (MRE).

Fig.3. shows a sample code type in two separated files. The user defined functions file (funct.mlpro) and the main program (_tests.mlpro) in which the instruction “include” (translated here by “chukua” in Swahili) allows to call and include user defined functions.

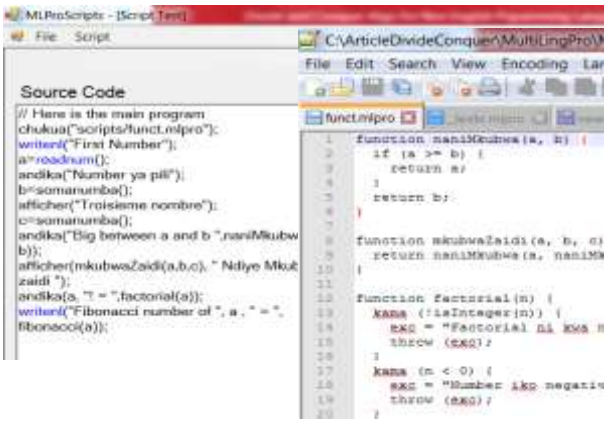

Fig. 3. Source code running from separate files

For the second way, MLProScript source code can be executed as script in another .Net program. The following few steps are needed to test the MLProScript library in a .Net (C# or Visual Basic) program:

1. Create a .Net project and save it;
2. Copy the two files (.dll and .xml) in the project bin/debug folder;
3. Add the dll as reference to the project;
4. Import (using) the dll in the source code file where the MLProScrip is to be called;
5. Instanciate the class MLScript either as a global attribute or private variable. If you plan to call the script in a windows form application, call the method MyConsole.AllocConsole() after the instanciation.
6. Run your script. Two kinds of scripts are handled by MLProScript : script located in a text file and free text script.
   6.1. To run a text file script, call the .run(file_location) method of the MLScript object.
   6.2. To run a free text script, assign the script text to the attribute .mySource or MLScript object, then call .run() method from the same.

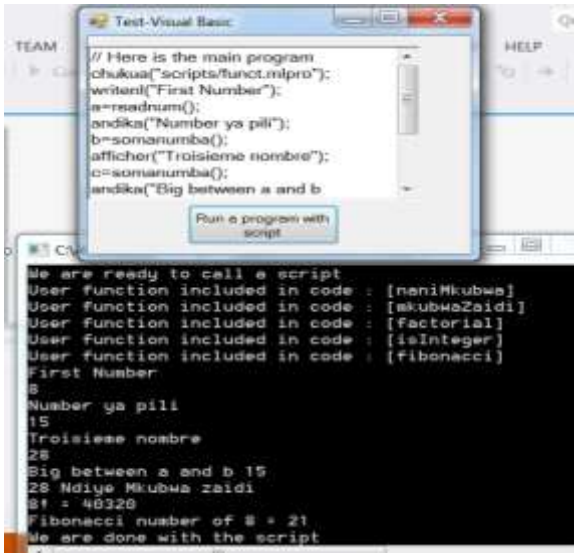

Fig. 4: MLProScript Scripting test in Visual Basic

Listing 4.1 shows how the source contained in the aforementioned source files are called and executed as scripts in a Visual Basic program. Results can be seen in Fig.4.

```
Dim prog As New MLProScript

MyConsole.AllocConsole()

'User program

Console.WriteLine("We are ready to call a script")
```

```
' Call the srcipt

prog.myMain("scripts/_tests.mlpro")

' Put the code in a TextBox

txtSource.Text = prog.mySource

' Continue user program

Console.WriteLine("We are done with the script")
```

Listing 4.1 MLProScript Scripting in Visual Basic

5. CONCLUSION AND DISCUSSION

The executable programming language MLProScript presented above proves that divide and conquer is a reliable alternative to design and implement interpreted multilingual programming languages. I allowed the researcher in this paper to create friendly parser for a scripting language using C#. Using the implemented prototype, junior developers and teachers can create programming languages in a clear, familiar, systematic and rapid manner using their local languages lexemes. Their programming languages can then be easily used by non-English end users and students without any language barrier and frustration [16] [17] [18] [19]. On the long run, results from this article can improve local language promotion, programming productivity, quality, and time-to-market among young programmers.

Furthermore, MLProScript is a real multilingual programming language platform. It allows the language developer to produce a language using lexemes from any natural spoken language [20]. Another improvement is that the end user can mix two or many languages in his scripts.

In order to make MLProScript a real scripting programming language, further research should focus on introducing more object oriented features as well as windows user interface components. Finally, since the MLProScript needs to be internationalized in order to be used by non-English programming language developers.

REFERENCES

[1] E. W. Dijkstra, "Stiching Mathematicak Centrum," Algol Buletin Supplement nr 10, 1961.

[2] F. DeRemer and T. Pennello, "Efficient Computation of LALR(1) Look-Ahead Sets," Transactions on Programming Languages (ACM), vol. 4, no. 4, p. 615–649, 2015.

[3] T. Parr and R. Qwong, "ANTLR: A Predicated-LL(k) Parser Generator," SOFTWARE-PRACTICE AND EXPERIENCE, vol. 25, no. 7, p. 789–81, 1995.

[4] T. Parr and K. Fisher, "LL(*): The foundation of the ANTLR parser generator.," Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) , pp. 425-436, 2011.

[5] A. Wöß, M. Löberbauer and H. Mössenböck, "LL(1) Conflict Resolution in a Recursive Descent Compiler Generator. .," in Modular Programming Languages, Joint Modular Languages Conference, JMLC 2003, Klagenfurt, Austria, 2003.

[6] B. Kuhl and A.-T. Schreiner, "An Object-oriented LL(1) parser generator" (2000). , Vol. 35 (No.," ACM SIGPLAN Notices, vol. 35, no. 12, 2000.

[7] H. Prähofer, D. Hurnaus and H. Mössenböck, "Building End-User Programming Systems Based on a Domain-Specific Language.," 2019.

[8] G. P. Arya, P. R. Neha Sohail, P. Kumari and S. Khatoon, "Design and Implementation of a Customized Compiler," (IJCSIT) International Journal of Computer Science and Information Technologies, vol. 8, no. 3, pp. 342-346, 2017.

[9] R. D. Necaise, Data Structures and Algorithms Using Algorithms Using, Hoboken: Wiley, 2011.

[10] B. Miller and D. Ranum, Problem Solving with Algorithms and Data Structures, 2013.

[11] C. Chow, T. Y. Chen and T. Tse, "The ART of Divide and Conquer," in The Symposium on Engineering Test Harness (TSETH '13), Proceedings of the 13th International Conference on Quality Software (QSIC '13), IEEE Computer Society, Los Alamitos, CA (2013), 2013.

[12] T. Kozsik and I. B. Z. H. Melinda Toth, "Static analysis for divide-and-conquer pattern discovery," Computing and Informatics, vol. 35, pp. 764-791, 2016.

[13] E. A. Albacea, Concepts in Programming Languages, 2nd ed., Quezon City: JPVA Publishing House, 2003.

[14] O. M. Masivi, "Extensible and Executable Lexicon Metamodel for non-English-like Programming Languages," International Journal of Scientific Research and Development (IJSRD), vol. 6, no. 11, pp. 858-863, 2019.

[15] V. Kaplan, "A New Algorithm to Parse a Mathematical Expression and its Application to Create a Customizable Programming Language," ICSEA 2016 : The Eleventh International Conference on Software Engineering

Advances, pp. 272-277, 2016.

[16] A. Kamal, M. N. Monsur, S. T. Jishan and N. Ahmed, "ChaScript: Breaking language barrier using a bengali programming system," 8th International Conference on Electrical and Computer Engineering, 29 January 2015.

[17] Y. A. Bassil and A. M. Barbar, "MyProLang – My Programming Language A Template-Driven Automatic Natural Programming Language," in Proceedings of the World Congress on Engineering and Computer Science 2008, SanFrancisco, 2008.

[18] P. J. Guo, "Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities," in CHI '18 Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, Montreal QC, Canada, 2018.

[19] A. K. Veerasamy and A. Shillabeer, "Teaching English Based Programming Courses to English Language Learners/Non-Native Speakers of English," International Proceedings of Economics Development and Research (IPEDR), vol. 70, no. 4, pp. 17-22, 2014.

[20] A. Riker, "Natural Language in Programming An English Syntax-based Approach for Reducing the Difficulty of First Programming Language Acquisition," Brandeis University, Waltham, Massachusetts, 2010.

[21] P. L. Shiuan and C. T. H. Ann, "A Divide-and-Conquer Strategy for Parsing," in Proceedings of the ACL/SIGPARSE 5th International Workshop on Parsing Technologies, Santa Cruz, USA, pp. 57 – 66.

BIOGRAPHY

Dr MASIVI is a Computer Science PhD holder from University of the Philippines Los Banos. He is currently Associate Professor lecturing IT and Computer Science courses at Rusangu University (Zambia) and ISP/Muhangi (RDC).