International Journal of Advance Engineering and Research Development (IJAERD)
Volume 1,Issue 6,June 2014, e-ISSN: 2348 - 4470 , print-ISSN:2348-6406

Performance Evaluation of Annotation Based Innovative Parser Generator

Ms.Yogita S. Alone[1], Prof.Ms. V. M. Deshmukh[2]

[1]*M.E II[nd] Year (CSE),Prof. Ram Meghe Institute of Technology & Research,Badnera, Amravati University, India*
[2]*Head Of Department, Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, Amravati University, India*

Abstract: Innovative parser construction method and parser generator prototype generates a computer language parser from a set of annotated classes in contrast to classic parser generators which specify concrete syntax of a computer language using BNF notation.The process of parser implementation ispresented on selected concrete computer language.By using computer languages, define the structure of a system and its behavior. Today's common industry practice is to create a software system as a composition of software artifacts written in more than one computer language. Besides the general purpose programming languages (e. g. Java, C#) the domain-specific languages (DSL) Nowadays, DSLs have their stable position in the development of software systems in many different forms. Concerning abstraction level, it is possible to program closer to a domain.

Keywords: Parsing, Debugging, Annotation , Expression Tree.

I. INTRODUCTION

A parser checks a program written in a source language for syntactic correctness andarranges for further processing by other means using some host language. A parsergenerator usually accepts an annotated grammar of the source language and, as aminimum, produces the recognition part of the parser. Because an annotated grammaris just one more source language, a parser generator is a specific case of a parser andcan be used to bootstrap its own implementation**.**Program analysis is a key component of tasks such as program comprehension, slicing, visualization and metrication, and acts as a foundation for more comprehensive tools Furthermore DSLs enables explicit separation of knowledge in the system in natural structured form of domain. The growth of their popularity is probably interconnected with the growth of XML technology and using of standardized industry XML document parsers as a preferable option to a construction of a language specific parsers. A developer with minimal knowledge about language parsing is able to create a DSL with XML compliant concrete syntax using tools like JAXB [8]. Even though XML documents are suitable for document exchange between different platforms they are too verbose to be created and read by humans. On the other side, XML languages are easily extensible with new language elements according to their nature and processors so they are perfectly suited for constantly evolving domains.

In particular, it can distinguish between *static analysis*, concerning information gleaned from the program code, and *dynamic analysis*, concerning information collected from running the program. At the level of static analysis, we can identify four main levels of information, associated with four phases of compilation:

1. Preprocessing involves dealing with conditional compilation and textual inclusion, and is mainly an issue in C and C++, although C# also has a limited form of preprocessor
2. Lexical analysis collects characters into words, and eliminates comments and whitespace. Tools working at the lexical level can provide crude metrics by analyzing keywords, and can often be constructed using relatively simple tools such as lex, grep or awk.

3. Parsing-level analysis concerns the hierarchical categorization of program constructs into syntactical categories such as declarations, expressions, statements etc.
4. Semantic analysis deals with issues such as definition use pairs, program slicing and identifier analyses.

While information from each level is needed to build a full view of a program, in many ways the parser is central to this process. Typically, it is the parser that drives the lexical analysis phase by requesting and organizing tokens. A parser also acts as a foundation for the semantic analysis phase either directly, through events triggered on recognition of various constructs, or indirectly through the generation of some form of intermediate representation. The innovative approach to the definition of a concrete syntax for a computer language with textual notation.Contrary to traditional methods of parser generation (e. g. YACC, JavaCC), we focus on the definition of abstract syntax rather than giving an excessive concentration on concrete syntax (see Fig 1).In our approach the abstract syntax of a language is formally defined usingstandard classes well known from object-oriented programming.

Parser generator-traditional approach

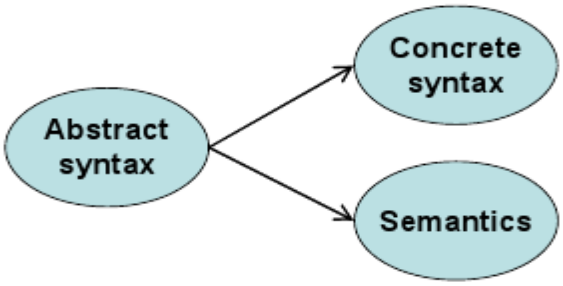

Fig 1: Comparing traditional and innovative approach

RELATED WORK

A. Program annotation in XML

In 2006" James F. Power & Brian A. Malloy" explained outlined of a general algorithm for the modification of the bison parser generator, so that it can produce a parse tree in XML format. It explore also an immediate application of this technique, a portable modification of the gcc compiler, that then allows for XML output for C, Objective C, C++ and Java programs. By modifying bison rather than gcc directly, It produced a tool that is applicable in any domain that uses the bison parser generator and, in particular, is directly applicable to multiple versions of gcc. While this approach does not have the same semantic richness as other approaches, it does have the advantage of being language independent and thus re-usable in a number of different domains. It do not envisage it as a stand-alone product, but believe that it will be useful as a starting point for more language-specific tools.Most of the constructs of languages such as Pascal andAda are context-free, and it is a straightforward matter togenerate a parser for these languages, particularly using aparser generator. An exception to this easy-parse rule canbe found in the language C, where a context-sensitive ambiguityexists between a declaration and an expression. This*declaration/expression ambiguityis* not withstanding, parserfront-ends for the C language have not been difficult to construct.

B. Annotation Parser Generator:

The language implementation begins with the concept formalization in the form of abstract syntax. Language concepts are defined as classes and relationships between them. Upon such defined abstract syntax a developer defines both the concrete syntax through a set of source code annotations and the language semantics through the object methods. Annotations (called also attributes) are structured way of additional knowledge incorporated directly into the source code. During thephase of concrete syntax definition the parser generator assists a developer with suggestions for making the concrete syntax unambiguous. fig6 shows the whole process of parser implementation using the described approach. If the concrete syntax is unambiguously defined then parser generator automatically generates the parser from annotated class. The specification of concrete syntax requires some additional information about textual representation of the language artifacts. In SAL it is:

- How to represent the number (notation),
- Which symbols are used for the operations of addition, multiplication and negation,
- The form of the notation of operation and the priority and associativity of all operations.

The operations will be expressed in postfix form using standard symbols + and *.The concrete syntax of a language is either textual or graphical. The graphical concrete syntax is often defined by the structure of the abstract syntax and a set of graphical representations for classes and associations in the abstract syntax Parsing only verifies that the program consists of tokens arranged in a syntactically valid combination. *semantic analysis*, where we even deeper to check whether they form a sensible set of instructions in the programming language.

II. SYSTEM DESIGN

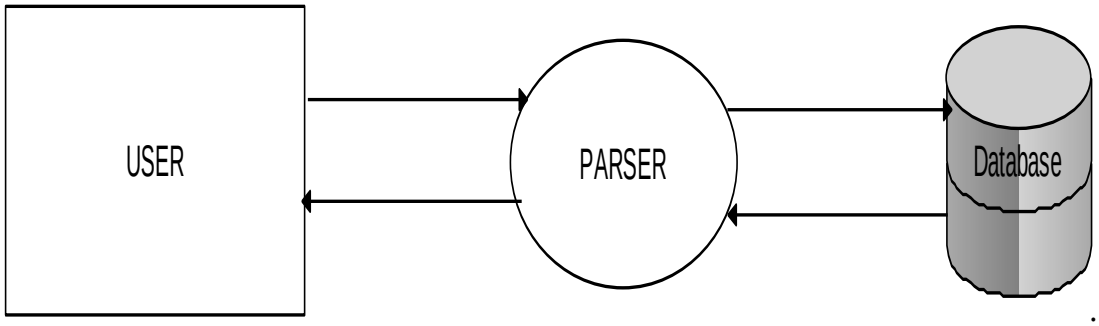

Fig 3: System design

Data flow diagrams (DFDs) reveal relationships amongand between the various components in a program orsystem. DFDs are an important technique for modeling asystem's high-level detail by showing how input data istransformed to output results through a sequence offunctional transformations. DFDs consist of four majorcomponents: entities, processes, data stores, and dataflows. The symbols used to depict how these componentsinteract in a system are simple and easy to understandhowever, there are several DFD models to work from,each having its own semiology.In this sections result analysis is discuss in detail. The parser basically use for syntax checking and generating syntax treeAt the time of parsing parser counts tokens, and many things.

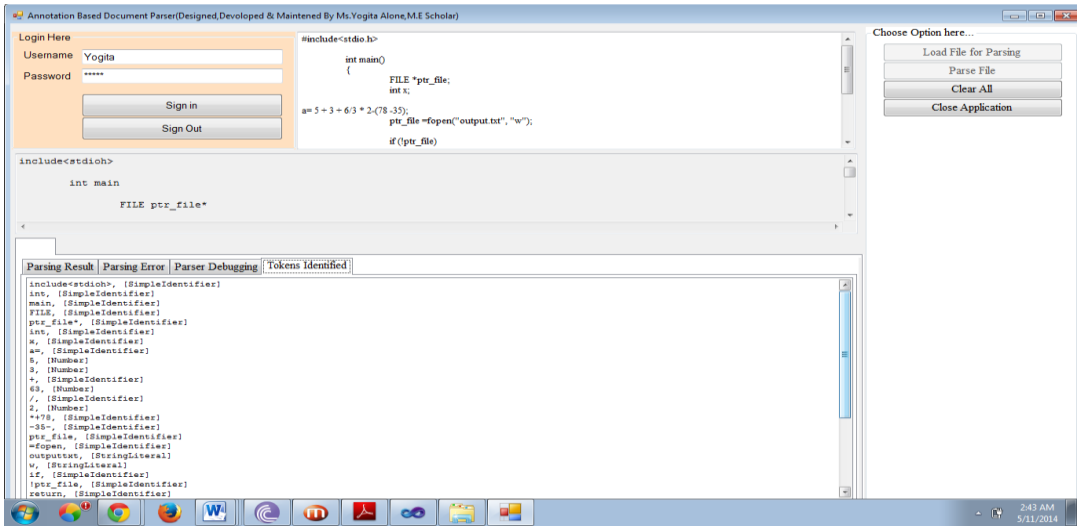

Figure-3.2 parsing C file which consist of SAL Expression

Name of Input file	Number of lines in file	Number of tokens	Initialization time(ms)	Execution time(ms)
vehical.xml	2087	7901	100	6864
File1.sgml	9	20	80	90
Education.xml	72	79	94	47
Sal.txt	2	6	93	16
Sil.txt	21	35	94	31

Figure 3.3 Comparison between different Files

III. CONCLUSION

The language itself is specified by a set of annotated classes. Annotation extend with additional information require for specification of concrete syntax, for example keywords and operator notation .The definition of abstract syntax and continue with creation of language in incremental way, it compare traditional approach of syntax. The language parser generated by using annotation. The generation of parser language counts the tokens, special symbol, and parenthesis.Innovative parser construction method and parser generator prototype which generates acomputer language parser from a set of annotated classes in contrast to classic parser generators which specify concrete syntax of a computer language using BNF notation. In the presented approach a language with textual concrete syntax is defined upon the abstract syntax definition extended with source code annotations. The process of parser implementation is presented on selected concrete computer language.

REFERANCES

1. JaroslavPorubän, Michal Forgáč, and Jaroslav.Poruban, Michal.Forgac, Miroslav.Sabo Slovak Republic "Annotation Based Parser Generator"2007.
2. V. Cepa,VDMVerlag Dr. Muller e.K "Attribute Enabled Software Development",., 2007. 216 p. ISBN 3836410168.
3. S. Cook, G. Jones, S. Kent, and A. C. Wills, "*Domain-Specific Development with Visual Studio DSL Tools*," Addison-WesleyProfessional, 576 pp. (2007).
4. C. Donnelly, R. Stallman, "Bison: The Yacc-compatible ParserGenerator", 2006, http://www.gnu.org/software/bison/manual/pdf/bison.pdf .
5. M. Fowler, "Language Workbenches: The Killer-App for Domain Specific Languages?" 2005. http://www.martinfowler.com/articles/ languageWorkbench.html
6. J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi, SoftwareFactories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, 2004. 500 p. ISBN 0471202843.
7. P. R. Henriques, M. J. Varando Pereira, M. Mernik, M. Lenič,J. G. Gray, H. Wu, "Automatic generation of language-based tools usingthe LISA system", In IEE proc., Softw. April. 2005, vol. 152, no. 2, pp.54-69.
8. "Java Compiler Compiler–The Java Parser Generator", https://javacc. dev.java.net .
9. "Java Architecture for XML Binding (JAXB)", https://jaxb.dev.java.net.
10. S. C. Johnson, YACC: "Yet Another Compiler-Compiler", Unix Programmer 1979de/Lehre/WS200304/Compilerbau/Uebungen/yacc.pdf
11. "JSR 175: A Metadata Facility for the Java Programming Language",http://jcp.org/en/jsr/detail?id=175 .
12. A. G. Kleppe.: A Language Description is More than a Metamodel.In: Fourth International Workshop on Software Language Engineering,1 Oct 2007, Nashville USA.

AUTHOR'S PROFILE

1) **Ms.Y.S.Alone** received the B.E.degrees in Computer Engineering from Bapurao Deshmukh College of Engineering, Sewagram wardha, in 2010.Now I am pursing ME(CSE) from Prof. Ram Meghe Institute Of Technology & Research ,Badnera, A mravati.

2) **Prof. Ms. V.M.Deshmukh** completed her B.E (CSE) from College of Engineering, Badnera in 1990 and M.E (CSE) from College of Engineering, Badnera in 2007. Pursuing Ph.D in Computer Science & Engineering (Design and development of XML parser).

