Scientific Journal of Impact Factor(SJIF): 3.134

e-ISSN(O): 2348-4470
p-ISSN(P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 2, Issue 6, June -2015

TOKEN BASED CREATIVE PARSER GENERATOR

Ms.Yogita S.Alone[1], Ms.Ruchita A. Kale[2], Mr.Gaurav. J. Sawale[3]

[1,2,3]*Department of computer science & engineering ,PRMIT&R ,Badnera,Amravati, Maharashtra ,India*

Abstract: *The parser generator is a context-free grammar. This context-free grammar is usually expressed in Backus -Naur form, BNF, or extended Backus-Naur form, EBNF, themselves metalanguages for context-free grammars. EBNF differs from BNF, allowing shortcuts resulting in fewer but more complicated productions.In this paper we praposed annotation based innovative Parser generator, and also find the solution for generating parsers for textual languages. Thepaper presents innovative parser construction method and parser generator prototype which generates a computer language parser from a set of annotated classes.In the presentedapproach a language with textual concrete syntax is definedupon the abstract syntax definition extended with sourcecode annotations. The process of parser implementation ispresented on selected concrete computer language – the SimpleArithmetic Language.*

Keyword: *Parsing, debugging, annotation,XML, Expression Tree, Lexical constructs.*

I. INTRODUCTION:

In this paper we present the innovative approach to the definition of concrete syntax for a computer Language with textual notation. Computer Languages are crucial tools in the development of software system. By using computer languages it define the structure of system and its behavior [1]Developers use different languages and paradigms throughout the development of a software system according to a nature of concrete sub problem and their preferences .Beside the general purpose programming languages(eg. C#) the domain specific languages(DSL) have become popular.DSLs have their stable position in the development of software system in many differentForms. Program analysis tools are the keystone of good software reverse engineering applications

In particular, Itcan distinguish between static analysis, concerning information gleaned from theprogram 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 intowords, band eliminates comments and whitespace. Tools working at the lexical level can provide crude lex, metrics by analyzing keywords, and can often be constructed using relatively simple tools such as 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.Concerning abstraction level,

It is possible to program closer to a domain. 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[10]. 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 withnew language elements according to theirnature and processors so they are perfectly suited for constantly evolving domains. It focus on the definition of abstract syntax rather than giving an excessive concentration on concrete syntax.

In this approach the abstract syntax of a language is formally defined using standard classes well known from object-oriented Programming. In our approach the language implementation begins with the concept formalization in the form of abstract syntax. Language concepts are defined as classes and relationships between them. Parser generator-traditional approach Fig 1:comparing traditional and innovative approach 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 the phase of concrete syntax definition the parser generator assists a developer. with suggestions for making the concrete syntax Unambiguous. Fig 2 shows the whole process ofparser implementation using the described approach. If the concrete syntax is Unambiguously defined then parser generator automatically generates the parser from annotated classes.

The paper presented on Annotation based innovative parser generator is organized as follow introduces us Annotation based innovative parser generator. II:Discusses Literature review of parser generator , as well as abstract and concrete syntax, defines System analysis and design, System architecture, several well-known widely used annotation , and algorithms are presented, also, System requirements are defined. techniques. presents Conclusion on this annotation based innovative parser generator

II. LITERATURE REVIEW

In "Program annotation in XML: a parser-based approach" outlined a general algorithm for the modification of the bison parser generator, so that it can produce a parse tree in XML format. We have also discussed 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, we have 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 our 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 as a stand-alone product, but believe that it will be useful as a starting point for more language[2]

"Introduction to JavaCC"A particularly common case is where the output of the parser is a tree that closely conforms to the tree of method calls made by the generated parser. In this case, there are additional tools that can be used in conjuction with JavaCC to automate the augmentation of the grammar. These tools are JJTree and JTB and are the subject of Chapter [TBD]. It is important to note that the lexical analyser works quite independently of the parser. In "Automatic Generation of Language-based Tools using the LISA system", Many applications today are written in well-understood domains.Onetrend in programming is to provide software tools designed specifically to handle the development of domain-specific applications in order to greatly simplify their construction. These tools take a high-level description of the specific task and generate a complete application[3].

In "Static Analysis for Event-Based XML Processing", the challenges that must be tackled in order to provide static analysis of event- based XML processing applications that use general purpose programming languages. Concretely, this paper has focused on SAX. The challenges include reasoning about sequences of SAX events both as input and as output, flow sensitive string computations, attribute maps, and field variables in Java. In addition to discussing the challenges, we have outlined a strategy for a particular program analysis that may serve as a starting point[4].

In "adaptive table-driven XML parsing and validation technique" that can be used to develop extensible high-performance Web services. The adaptive TDX encodes XML parsing states in compact tabular forms by support of permutation phrase grammar. As a result it ensures a memory space efficiency. This adaptive approach uses interpretive scanning at run time by leveraging these tabular forms to improve scanning performance [7].

In "A Language description is more than a metamodel" ,the act of language design is one in which a carefull balance must be upload between the three main elements of language description: abstract syntax ,concrete syntax and semantics .A software should be built iteratively starting with parts of the abstract syntax, then adding concrete syntax and semantics to these parts.Design Patterns in Parsing it discussed oops3as an example for the consequent use of design patterns in parsing and parser generation and it pointed out significant benefits of the architecture. The central concept is to represent source programs as trees and to implement tree manipulation using the Visitor pattern. Tree classes usually are specific to the source grammar and provide natural boundaries for divide-and-conquer in all algorithms. The Visitor pattern combines the class-specific pieces of an algorithm in a central class. Recognition is implemented as a visitor with template methods. It is

sub classed to Provide different ways to observe the recognition process. One particular Observer pattern instance connects recognition to a tree factory to represent a source program; the tree factory can be generated from annotations in the source grammar[22].

The Unix utility YACC parser a stream of token typically generated by lex,according to user specified grammar definition section includes include three things Ccode,definition& associativity rules.It explore all the rule which are mention above and extend the concept with the help of program examples.Conceptuallylex parses a file of characters and output a stream of tokens;yacc accepts a stream of tokens and parses it,performing action as appropriate.[21]

The development of superfast compilers such as are found in Borland's Turbo Pascal and Delphi systems.The fact that the use of high-level languages distances the programmer from the low-level nuances of a machine may lead to other difficulties and misconceptions [20].

A formal language description, ANTLR generates a program that determines whether sentences conform to that language. By adding code snippets to the grammar, the recognizer becomes a translator. The code snippets compute output phrases based upon computations on input phrases. ANTLR is suitable for the simplest and the most complicated language recognition and translation problems[10].

These exposed the challenges that must be tackled in order to provide static analysis of event-based XML processing applications that use general purpose programming languages. Concretely, this paper has focused on SAX. The challenges include reasoning about sequences of SAX events both as input and as output, flow sensitive string computations, attribute maps, and field variables in Java. In addition to discussing the challenges, it proposed a strategy for a particular program analysis that may serve as a starting point. To summarize, the key ideas suggested here are the following, which build on top of the existing program analysis technique for Java Servlets and XACT[22]

In 2002 ,SrikanthKarre et. al explains that the parsing of XML documents can be done using two approaches, Event Based Parsing and Tree BasedParsing. In Event Based Parsing, the XML data is parsed sequentially, one component at a time, and the parsing of events such as the start of a document, or the end of a document are reported directly to the application. SAX (Simple API for XML) is the standard API for event-driven parsing. In Tree Based Parsing, the XML document is compiled into an internal tree structure and stored in main memory [1]. Applications can then use this tree structure for navigation and data extraction. For example, the Document Object Model (DOM) uses tree based parsing, providing a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing manipulating them.

In 2004, Robert A et. al. describes a validating XML parsing method based on deterministic finite state automata (DFA). XML parsing and validation is performed by a schema-specific XML parser that encodes the admissible parsing states as a DFA. This DFA is automatically constructed from the XML schemas of XML messages using a code generator. A two level DFA architecture is used to increase efficiency and to reduce the generated code size. The lower-level DFA efficiently parses syntactically well-formed XML messages [4]. The higher-level DFA validates the messages and produces application events associated with transitions in the DFA. Two example case studies are presented and performance results are given to demonstrate that the approach supports the implementation of high-performance Web services.

In 2007, Su Cheng Haw et. al. -A Comparative Study and Benchmarking on XML Parsers, they studied different XML parsers and determine the strength and weaknesses of the product. Various studies have been conducted which compare on conformance to standards, speed, memory usage and so on [6]

III. PROPOSED SYSTEM

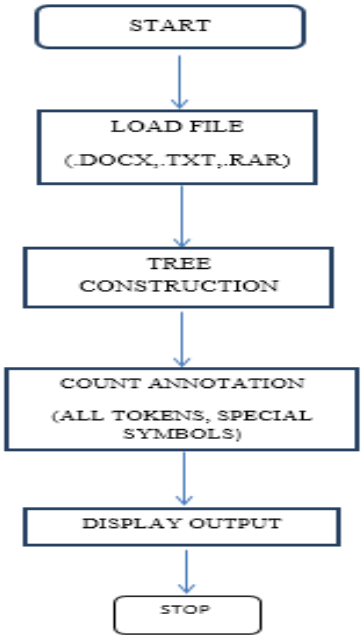

Fig 1:system flow

In Annotation based innovative Parser generator Annotation extend with additional information require for specification of concrete syntax, for example keywords and operator notation.The fig 4:show the data flow diagram of system. Stepwise description of proposed system design:

1. Initially load the file of any extension in annotation based innovative parser generator , there is no restriction on type of file ,it may be(.doc,.txt,.rar,etc).
2. After loading the parser converts the loaded file into text file with help of object notation, it then find the hash value for content of text file.
3. Form those hash values parser generates the binary tree structure form.
4. From generated tree structure it count all annotations, parenthesis and also find the errors.

IV. SYSTEM ARCHITECTURE

It is quite common to have multiple notations for one language. RELAX NG is an example of such a language with two different notations.XML syntax and compact syntax. By using this approach different notations of the same language can share both abstract syntax and Semantics. This means that other notations of the same language are not affected by this type of language evolution. For instance, Fig 3 presents. the language with 4 different notations sharing the same abstract syntax and semantics.

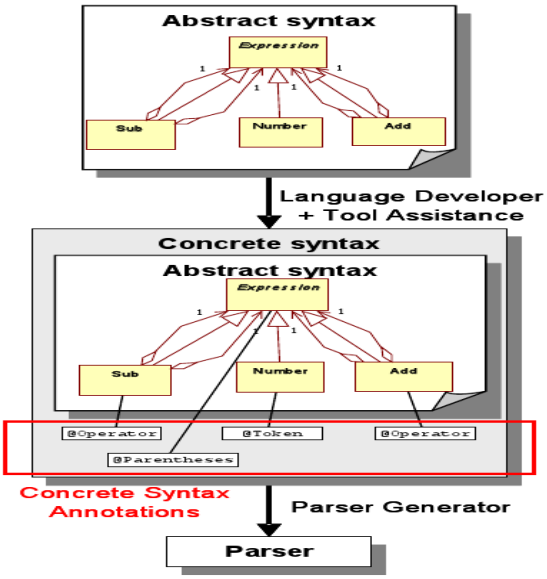

Fig 2:Generating Language Parser

.

Lexical constructs: Terminal symbols refering to complex strings defined by a regular expression (e.g., identifier). Lexical constructs are specified as strings that contain letters and underscores ( ) only. In defining a category, a developer may use the different EBNF. Using this notation, a category <expression> specifying the syntax for prefix arithmetic expressions could have the following syntax:

<expression> ::= integer j ['+' j '-' j '*' j '/'] <expression><expression> ;

From this, we see that an expression is either an integer value (lexical construct integer) or an arithmetic operator (either '+', '-', '*', or '/'), followed by two expressions. When building a parser, it performs a number of verifications. First, it identify parasite categories. These are categories that never lead to reserved words or lexical constructs. In other words, parasites are infinitely expanding categories. Second, it listed symbols that are never used, also called inaccessible symbols. These are reserved words or lexical constructs which may never be reached from the main category of the language. Parasites and inaccessible symbols normally generate warnings, telling the developer that some detailed analysis of the language must be performed to ensure that there is no error.The semantics of SAL is formally defined using Eval function which maps a value from syntactic domain Expression to a value from semantic domain Z (integers) and Value function which maps a value from syntactic domain Number to a value from semantic domain Z.

Eval : Expression ® Z

In this approach 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 the phase of concrete syntax definition the parser generator assists a developer with suggestions for making the concrete syntax unambiguous. Fig 3 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 classes.Grammars are defined using EBNF (Extended Backus-Naur Form) syntax. In this syntax, we distinguish between three concepts:

Categories: Non-terminal symbols which are defined using an EBNF description.Categorynames are specified as strings written between < and > (e.g., <grammar>).

Reserved words: Terminal symbols refering to a specific string (e.g., while). Reserved words are specified as strings between single quotes (') (e.g., 'while').Lexical constructs: Terminal symbols refering to complex strings defined by a regular expression (e.g., identifier). Lexical constructs are specified as strings that contain letters and underscores ( ) only. In defining a category, a developer may use the different EBNF. Using this notation, a category <expression> specifying the syntax for prefix arithmetic expressions could have the following syntax:

<expression> ::= integer j ['+' j '-' j '*' j '/'] <expression><expression> ;

From this, we see that an expression is either an integer value (lexical construct integer) or an arithmetic operator (either '+', '-', '*', or '/'), followed by two expressions. When building a parser, performs a number of verifications.

First, it identify parasite categories. These are categories that never lead to reserved words or lexical constructs. In other words, parasites are infinitely expanding categories. Second ,it listed symbols that are never used, also called inaccessible symbols. These are reserved words or lexical constructs which may never be reached from the main category of the language. Parasites and inaccessible symbols normally generate warnings, telling the developer that some detailed analysis of the language must be performed to ensure that there is no error.The semantics of SAL is formally defined using Evalfunction which maps a value from syntactic domain Expression to a value from semantic domain Z (integers) and Value function which maps a value from syntactic domain Number to a value from semantic domain Z.

Eval : Expression ® Z
Value : Number ® Z

The semantic function Eval is defined by the following equations.
Eval [| Number n |] = Value [| n |]
Eval [| UnaryMinus e |] = – Eval [| e |]
Eval [| Add e1 e2 |] = Eval [| e1 |] + Eval [| e2 |]
Eval [| Mul e1 e2 |] = Eval [| e1 |] * Eval [| e2 |]
Certainly we can find many different notations for SAL. For example, we can write down a sentence from SAL in the following notation using standard symbols and the operator infix form.

1 + 2 * 7

Above fig:3 abstract syntax tree of the sentence above is depicted.

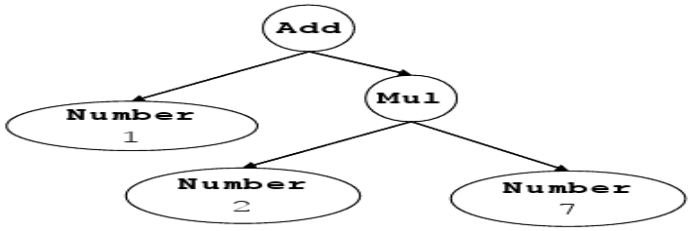

Fig 3: abstract syntax tree

V. 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 . it describes the definition of abstract syntax and continue with creation of language in incremental way, it compare traditional approach of syntax. The languageparser generated by using annotation. The generation of parser language counts the tokens, special symbol parenthesis.Innovative parser construction method and parser generator prototype which 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 .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.These exposed the challenges that must be tackled in order to provide static analysis of event-based XML processing applications that use general purpose programming languages. Concretely, this paper has focused on SAX. The challenges include reasoning about sequences of SAX events both as input and as output, flow sensitive string computations, attribute maps, and field variables in Java. In addition to discussing the challenges, it proposed a strategy for a particular program analysis that may serve as a starting point..In relations can be classified with good performance and two advantages of our method actually improved the performance of relation classification. The immediately extension of our work is to improve the performance of relation classification by enriching the dataset, For compatibility, the types from the DTD specification have been translated mechanically. A careful analysis of the original XML document might lead to a more precise specification of the types of elements and attributes.

REFERENCES:

[1]. JaroslavPorubän, Michal Forgáč, and JaroslavPoruban,Michal.Forgac,Miroslav.Sabo:for the Java Programming Language",
[2]. A.G. Kleppe.: *A Language Description is More than a Metamodel*. In: Fourth International Workshop on Software Language Engineering, 1 Oct 2007, Nashville, USA.
[3]. M. Mernik, J. Heering, A. M. Sloane, "When and How to Develop Domain- Specific Languages", ACM Computing Surveys, Vol. 37, No. 4, December 2005, p. 316–344.
[4]. P. A. Muller, F. Fondement, F. Fleurey, M. Hassenforder, R. Schneckenburger, S. Gerard, J.M. Jezequel, :*Model-Driven Analysis and Synthesis of Textual Concrete Syntax*, Journal on Software and SystemsModeling (SoSyM), Volume 7 (4), Springer, 2008, p. 423 - 441.
[5]. T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*, Pragmatic Bookshelf, 376 pp. (2007).
[6]. M. Mernik, J. Heering, A. M. Sloane, "When and How to Develop Domain-Specific Languages", ACM Computing Surveys, Vol. 37, No. 4,December 2005, p. 316–344.
[7]. P. A. Muller, F. Fondement, F. Fleurey, M. Hassenforder, R. Schneckenburger,S. Gérard, J.M. Jézéquel, : Model-Driven Analysis andSynthesis of Textual Concrete Syntax, Journal on Software and SystemsModeling (SoSyM), Volume 7 (4), Springer, 2008, p. 423 - 441.
[8]. T. Parr, The Definitive ANTLR Reference: Building Domain-SpecificLanguages, Pragmatic Bookshelf, 376 pp. (2007).
[9]. "RELAX NG Specification", http://relaxng.org/ , 2003.vb.
[10]. T. Stahl, M. Voelter, Model-Driven Software Development:Technology, Engineering, Management, Wiley, 2006. 444 p. ISBN0470025700.
[11]. P. D. Terry, Compiling with C# and Java, Addison Wesley, 2004, 624p. ISBN 032126360X.
[12]. A.V. Aho, R. Sethi, and J.D. Ullman, Compilers – Principles, Techniques, and Tools,Addison-Wesley, Reading, Massachusetts, 1986. [Demonstrates how grammars are usedin the construction of compilers for programming languages.]
[13]. S.C. Johnson, Yacc – Yet Another Compiler-Compiler, Computer Science Tech. Rept.No. 32, Bell Laboratories, Murray Hill, New Jersey, July 1975.
[14]. TWEAST: A Simple and Effective Technique to ImplementConcreteSyntaxAST Rewriting Using Partial ParsingAkimDemaille Roland LevillainBenoˆıtSigoureEPITA Research and Development Laboratory (LRDE)14/16, rue Voltaire, F94276,Le KremlinBicˆ etre, France.
[15]. Induction, Grammars, and Parsing
[16]. COMPTOOLS: A CompilerGenerator for C and JavaPar : Gilbert Babin Cahier de recherche no 04-0910 novembre 2004.
[17]. Formalizing adequacy:a case study for higher-order abstract syntaxJames Cheney Michael Norrish Rene Vestergaard.
[18]. Textual Notation Syntax DefinitionOn the Design of Application Protocols
[19]. Maptool-Mapping between concrete And Abstract Syntaxes"BasimM.Kadhim,WilliamM.Waite 1995
[20]. Integrated Denition of Abstract and ConcreteSyntax for Textual LanguagesHolgerKrahn, Bernhard Rumpe, and Steven Volkel2007

