ISSN: 2321-7782 (Online)
Volume 2, Issue 1, January 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

Annotation based innovative Parser Generator

Y. S. Alone[1]
M.E.
Department of Computer Science & Engineering
PRMIT&R
Amravati - India

V. M. Deshmukh[2]
Prof.
Department of Computer Science & Engineering
PRMIT&R
Amravati - India

Abstract: ***In this paper we proposed annotation based innovative Parser generator, and also find the solution for generating parsers for textual languages. The paper presents innovative parser construction method and parser generator prototype which generates a computer language parser from a set of annotated classes. The paper summarizes result of the studies of implemented parser generator and describes its role. XML is emphasized for its language neutrality and application independency, and thus it has been adopted as the data exchange standard in cloud computing environments.***

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 we define the structure of system and its behavior 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 different forms. Program analysis tools are the keystone of good software reverse engineering applications. In particular, we 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 lex, metrics by analysing 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. 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. 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 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 of parser implementation using the described approach. If the concrete syntax is unambiguously defined then parser generator automatically generates the parser from annotated classes.

Parser generator-traditional approach

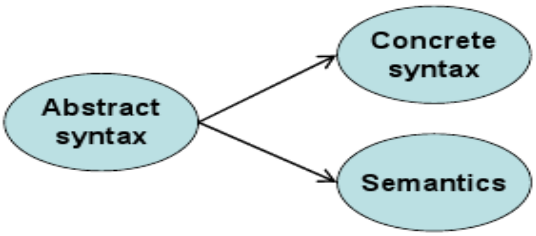

Fig 1:comparing traditional and innovative approach

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 our approach. By using our 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 2 presents. The language with 4 different notations sharing the same abstract syntax and semantics.

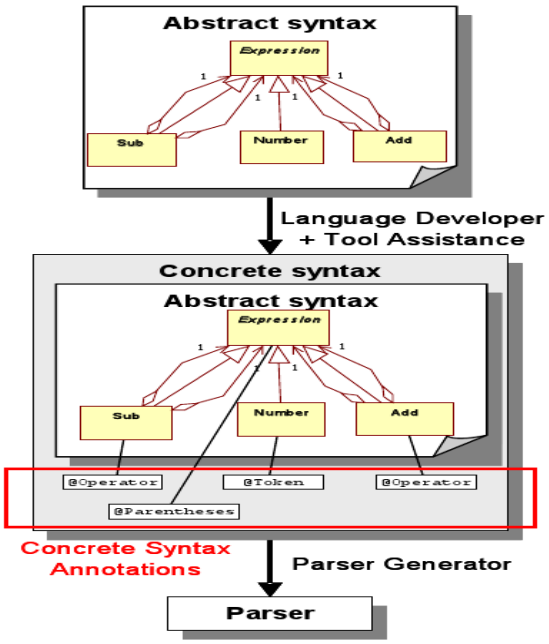

Fig 2: Generating Language Parser

The Seminar on Annotation based innovative parser generator is organized as follow introduces us Annotation based innovative parser generator. 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" we study 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. We do not envisage it as a stand-alone product, but believe that it will be useful as a starting point for more language [1] "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.One trend 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. In "Static Analysis for Event-Based XML Processing", we study 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. 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 memory space efficiency. This adaptive approach uses interpretive scanning at run time by leveraging these tabular forms to improve scanning performance. In "A Language description is more than a metamodel", the act of language design is one in which a careful 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 "paper discussed oops3 as 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.

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 paper start with 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

parenthesi.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. 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. 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. Jaroslav Porubän, Michal Forgáč, and JaroslavPoruban,Michal.Forgac,Miroslav.Sabo:Annotation Based Parser Generator.
2. V. Cepa, “Attribute Enabled Software Development”, VDM Verlag Dr. Muller e.K., 2007. 216 p. ISBN 3836410168
3. C. Donnelly, R. Stallman, “Bison: The Yacc-compatible Parser Generator”, 2006, http://www.gnu.org/software/bison/manual/pdf/ bison.pdf .
4. M. Fowler, “Language Workbenches: The Killer-App for Domain Specific Languages?” 2005. http://www.martinfowler.com/articles/ languageWorkbench. html .
5. J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004. 500 p. ISBN 0471202843.
6. P. R. Henriques, M. J. Varando Pereira, M. Mernik, M. Lenič, J. G. Gray, H. Wu, “Automatic generation of language-based tools using the LISA system”, In IEE proc., Softw. April. 2005, vol. 152, no. 2, pp. 54-69.
7. “Java Compiler Compiler–The Java Parser Generator”, https://javacc. dev.java.net .
8. “Java Architecture for XML Binding (JAXB)”, https://jaxb.dev.java. net.
9. S. C. Johnson, YACC: “Yet Another Compiler-Compiler Unix Programmer's Manual Volume 2b, 1979, http://www2.informatik.unierlangen.de/Lehre/WS200304/Compilerbau/Uebungen/yacc.pdf . [10] ”JSR 175: A Metadata Facility for the Java Programming Language”, http://jcp.org/en/jsr/detail?id=175 .
10. A. G. Kleppe.: *A Language Description is More than a Metamodel.* In: Fourth International Workshop on Software Language Engineering, 1 Oct 2007, Nashville, USA.
11. 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.
12. 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 Systems Modeling (SoSyM), Volume 7 (4), Springer, 2008, p. 423 - 441.
13. T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*, Pragmatic Bookshelf, 376 pp. (2007).

