Comparision of Different Types of Parser and Parsing Techniques

Mr.Amitesh Saxena, Mrs. Snehlata Kothari

***Abstract*— Today most recursive descent parsers are generated by providing grammars and generating parsers according to these grammars. An alternative approach to constructing parsers consists of parser combinators, which do not need a separate step to generate the parser, and furthermore claim to be clear and simple in use. Despite these claimed advantages, parser combinators have not been widely adopted and are rarely actually compared to parser generators. Presently there are a lot of XML parsers, and many of them evolve, improve and become complicated. Though all parsers provide the same purpose, they differ in conditions of specification, performance, reliability and also conformance to standards. If a appropriate choice has been not made, it is very much possible to leads to the trouble of unnecessary hardware requirement, which will result in productivity degradation.**

***Index Terms*—XML parsers, parsers, parsers combinators.**

I. Introduction

A tool that supports us in getting an overview of a software system must somehow translate that system into a model. This translation is a challenging point. Someone must write a parser that can translate that software system into the model he wants to support. So, the maintainers of such tools must provide a parser for every programming language they want to support. But it is not only the number of languages that is a problem. A language itself also evolves. A parser that works with a specific version and/or dialect could not work with the next version anymore.

A parser does two things while processing its input:

1. Split the input into tokens.
2. Find the hierarchical structure of the input.

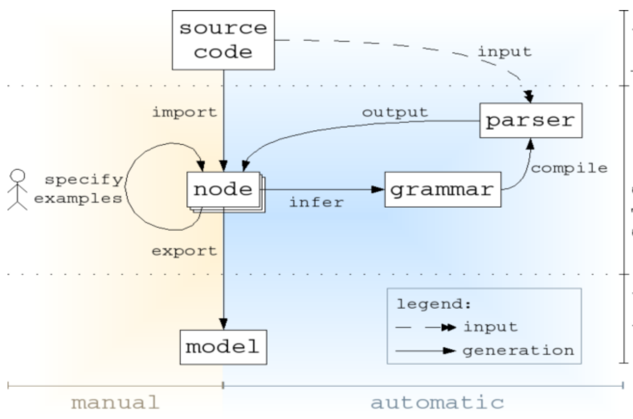


Figure 1: The way from source code to a model.

Manuscript received February 20, 2015.

Mr.Amitesh Saxena, Ph.D. Scholar, Pacific University, Udaipur

Mrs. Snehlata Kothari, H.O.D. IT Department, Pacific University, Udaipur

A common activity within computer science is the analyzing of tokens, called parsing. Contemporary parsers are constructed by means of parser generators and to a lesser extent by parser combinators. Although parser generators are the standard approach to constructing parsers, they are not always the easiest method. Parser combinators have long claimed to be more intuitive and easier in use than their generating counterpart.

II. Different Parser and Parsing technique

There are different levels of parsing. They go from 'Lexical Analysis' to 'Precise Parsing'. Between these two barriers there is 'Fuzzy Parsing', 'Island Grammars', 'Skeleton Grammars' and 'Error Repair.

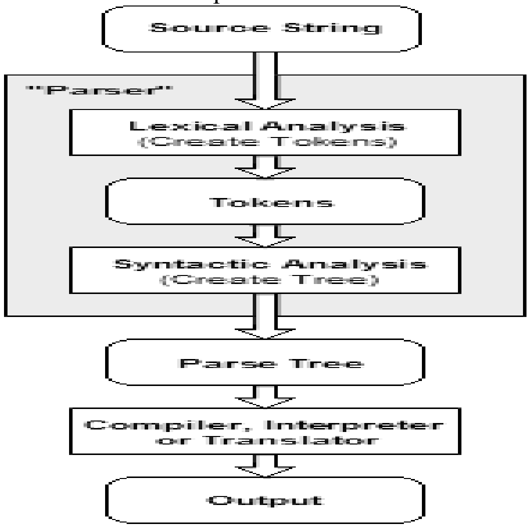

Source String
"Parser"
Lexical Analysis (Create Tokens)
Tokens
Syntactic Analysis (Create Tree)
Parse Tree
Compiler, Interpreter or Translator
Output

Figure 2: overview of parsing process

a) Fuzzy Parsing

Most reengineering frameworks use a form of fuzzy parsing in order to support more programming languages or more dialects of the same programming language. The goal of a fuzzy parser is the extraction of a partial source code model based on a syntactical analysis. The key idea of fuzzy parsing is that there are some anchor terminals. The parser skips all input until an anchor terminal is found and then context-free analysis is attempted using a production starting with the found anchor terminal

b) Island Grammars

With island grammars we get tolerant parsers. An island grammar is a grammar that consists of detailed productions describing certain constructs of interest (the islands) and liberal productions that catch the remainder (the water). By varying the amount and details in productions for the constructs of interest, we can trade off accuracy, completeness and development speed. There are some different versions of island grammars known besides the one that we just defined [MOON 01]. Leon Moonen speaks of the following:

• Lake grammar: When we start with a complete grammar of a language and extend it with a number of liberal productions (water) we get a lake grammar. Such a grammar is useful when we want to allow arbitrary embedded code in the program we want to process.
• Islands with lakes: This is a mix of productions for islands and water. We can specify nested constructs as islands with lakes.
• Lakes with islands: This is another mix of productions for islands and water.

III. Markup Languages used in parsing

A. *Standard Generalized Markup Language (SGML)*

It deals with the structural markup of electronic documents. The basic SGML document consists of a DTD or Document Type Declaration, one of several top level elements (otherwise known as tags or markups), paragraphs and text. The top level element should be a <book>, <chapter>, <article>, or <sect1>, depending on the type of document you are writing. We will be using <article> for our documents. Here is an example of a simple SGML document.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
  <sect1 id="introduction"><title>Hello world introduction</title>
      <para>
      Hello world!
      </para>
  </sect1>
</article>
```

Notice on the document that how we commented out the license using the <!-- and the -->. This is important; if you forget this you will get all kinds of errors when you run the file through the SGML parser. This information will not be viewable once you build it. The reason it is not viewable is the parser thinks it's just a comment (and it is!) so it just drops it out of the final parsed document.

B. *Hyper Text Markup Language (HTML)*

HTML is not a programming language, but rather a markup language. If you already know XML, HTML will be a snap for you to learn. We urge you not to attempt to blow through this tutorial in one sitting. Instead, we recommend that you spend 15 minutes to an hour a day practicing HTML and then take a break to let the information settle in. We aren't going anywhere! .HTML hasn't been around for many years. HTML is a **markup** language for **describing** web documents (web pages).

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- A markup language is a set of **markup tags**
- HTML documents are described by **HTML tags**
- Each HTML tag **describes** different document content

C. *Extensible Markup Language (XML)*

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all free open standards. The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Many application programming interfaces (APIs) have been developed to aid software developers with processing XML data, and several schema systems exist to aid in the definition of XML-based languages.

XML declaration

XML documents may begin by declaring some information about themselves, as in the following example:

<? xml version="1.0" encoding="UTF-8"**?>**

XML is used for structuring the data. The Structured data includes things like spreadsheets, address books, configuration parameters, financial transactions, and technical drawings. XML is a set of rules (you may also think of them as guidelines or conventions) for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids common pitfalls in language design: it is extensible, platform-independent, and it supports internationalization and localization. XML is fully Unicode-compliant.

XML has come into common use for the interchange of data over the Internet. IETF RFC 7303 gives rules for the construction of Internet Media Types for use when sending XML. It also defines the media type's application/xml and text/xml, which say only that the data are in XML, and nothing about its semantics. The use of text/xml has been criticized as a potential source of encoding problems and it has been suggested that it should be deprecated. RFC 7303 also recommends that XML-based languages be given media types ending in +xml; for example image/svg+xml for SVG. Further guidelines for the use of XML in a networked context may be found in RFC 3470, also known as IETF BCP 70 a document which covers many aspects of designing and deploying an XML-based language.

XML parsers

Oracle provides XML parsers for Java, C, C++, and PL/SQL. This chapter discusses the parser for Java only. Each of these parsers is a standalone XML component that parses an XML document (and possibly also a standalone document type definition (DTD) or XML Schema) so that they can be processed by your application. In this chapter, the application examples presented are written in Java

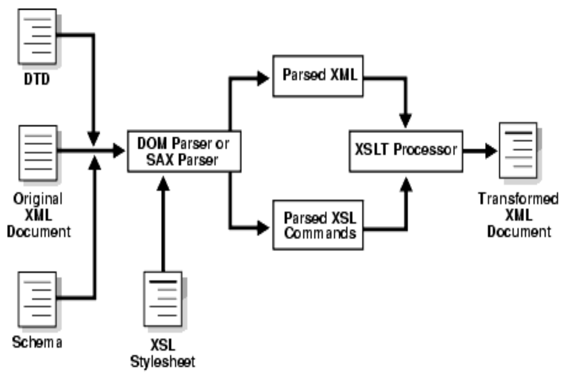


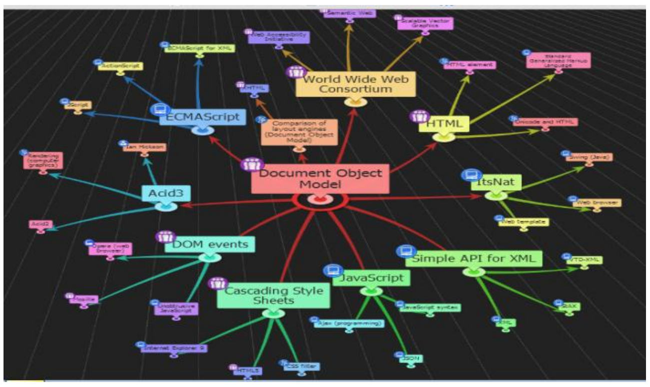
Figure 3 XML Parser for Java

IV. DOCUMENT OBJECT MODEL [DOM]

The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API). The history of the Document Object Model is intertwined with the history of the "browser

Figure 4 DOM Model

wars" of the late 1990s between Netscape Navigator and Microsoft Internet Explorer, as well as with that of JavaScript and JScript, the first scripting languages to be widely implemented in the layout engines of web browsers. The XML DOM is:



- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

Parsing

Parsing can also be used as a linguistic term, for instance when discussing how phrases are divided up in garden path sentences. Parsing is also an earlier term for the diagramming of sentences of natural languages, and is still used for the diagramming of inflected languages, such as the Romance languages or Latin. Parsing is a common term used in psycholinguistics when describing language comprehension. In this context, parsing refers to the way that human beings, rather than computers, analyze a sentence or phrase (in spoken language or text) "in terms of grammatical constituents, identifying the parts of speech, syntactic relations, etc." This term is especially common when discussing what linguistic cues help speakers to parse garden-path sentences. In computing, a parser is one of the components in an interpreter or compiler that checks for correct syntax and builds a data structure (often some kind of parse tree, abstract syntax tree or other hierarchical structure) implicit in the input tokens.

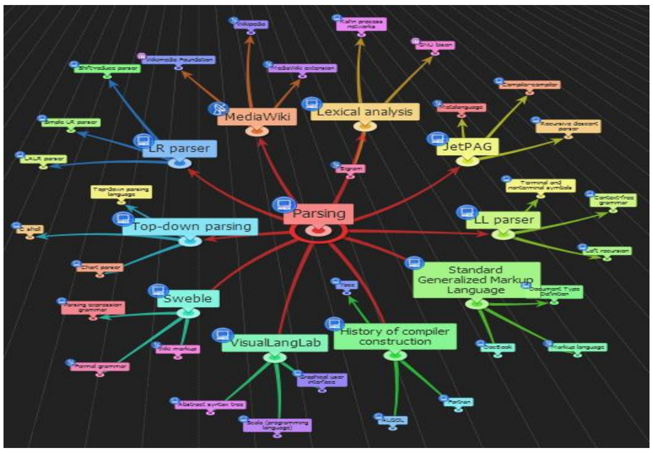


Figure 5:- parsing modal

Proposed work of research

I will try to find out the difference between processing performance of XML DOM parsing by using three operating systems

Node	Time pt1	Time pt2	Time pt3
Node 1	pt1-t1	pt2-t1	pt3
Node 2	pt1-t2	pt2-t2	pt3
Node 3	pt1-t3	pt2-t3	pt3
Node 4	pt1-t4	pt2-t4	pt3

Proposed modal for analysis

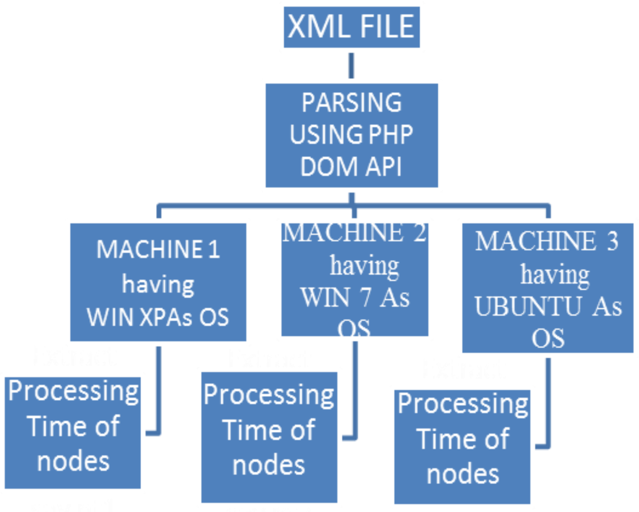

XML FILE
PARSING USING PHP DOM API
MACHINE 1 having WIN XPAs OS
MACHINE 2 having WIN 7 As OS
MACHINE 3 having UBUNTU As OS
Processing Time of nodes
Processing Time of nodes
Processing Time of nodes

Figure 6 Proposed Modal

V. Conclusion

I am working on the parsing technique to find the best parsing technique for different operating system. I have displayed the working modal of my research. We use Descriptive statistics along with 1x3 factorial ANOVA Technique and for the comparison mean, SD, z-test, t- test have been performed for data analysis.

References

[1] Extensible Markup Language, http://www.w3.org/TR/REC-xml.

[2] OASIS, http://www.oasis-open.org.

[3] Juancarlo Anez, "Java XML Parsers-A Comparative Evaluation of 7 Free Tools," Java Report Online, February 1999.

[4] Michael Claben, XML Parser Comparison, http://www.webreference.com/xml/column22/index.html. Feb 1999.

[5] Clark Cooper, Summary of XML Parser Performance Testing http://www.xml.com/lpt/a/Benchmark/exec.html. May 05, 1999.

[6] Mohseni, P., "Choose Your Java XML Parser", 2001, http://www.devx.com/xml/Article/16921.

[7] Karre, S. and Elbaum, S., "An Empirical Assessment of XML Parsers", *6th* Workshop on Web Engineering, 2002, pp. 39-46.

[8] Noga, M., Schott, S., L¨owe, W. (2002): Lazy XML Processing. In ACM DocEng, ACM Press, New York, 2002.

[9] Y. Oren, "SAX Parser Benchmarks", *http://piccolo. sourceforge.net/bench.html*, 2002.

[10] Nicola M. and John J. (2003): XML parsing: a threat to database performance. CIKM 2003: 175-178.

[11] Elliotte, R.H., "SAX Conformance Testing", XML Europe, 2004.

[12] Van Engelen, R. (2004): Constructing finite state automata for high performance XML web services. In Proceedings of the International Symposium on Web Services (ISWS), 2004.

[13] Takase T., Miyashita H., Suzumura T., and Tatsubori M. (2005): An adaptive, fast, and safe XML parser based on byte sequences memorization. WWW 2005: 692-701.

[14] Sosnoski, D.M., "XMLBench", 2005 http://www.sosnoski.com/opensrc/xmlbench.

[15] XMLJ News Desk, "Journal Readers choice Award", 2004 http://xml.sys-con.com/read/44008.htm.

[16] E. Perkins, M. Kostoulas, A. Heifets, M. Matsa, and N. Mendelsohn, "Performance Analysis of XML APIs", in *XML 2005 Conference proceeding*, 2005.

Name:- **Mr. Amitesh Saxena**
Qualifiaction:- M.tech,Phd(Pursuing)
Phd Scholar , Pacific University, Udaipur

Mrs. Snehlata Kothari
Qualification :- Phd
HOD(IT) at Pacific University Udaipur