International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 7, July - 2013

Optimized XML Assembler /Parser For XML Files

Hitesh Anand
Department of Computer Science
CT Group of Institutions,
Shahpur Campus, Jalandhar (Punjab)

ABSTRACT

In this paper, we describe the parsing of XML files via generated Optimized code and validation of the files in database computing field. In this paper, we present an elegant and effective framework for combining content and collaboration.So, in order to reduce the delay in parsing of generated XML files and its validation, we need an optimized code for that purpose so that XML parser converts an XML document into an XML DOM object rapidly, with high perform ability and less loading time will be taken in this process.In this paper, we introduced our XML parser that was implemented using XML classes. The main goal of this parser was to check XML documents for errors in a very easy and fast way, in order to help the programmer to determine whether the XML document is Well-formed or not.

Categories and Subject Descriptors

[**Programming Languages**]: Language: VB Script and XML

Platform: Any Operating System, Notepad.

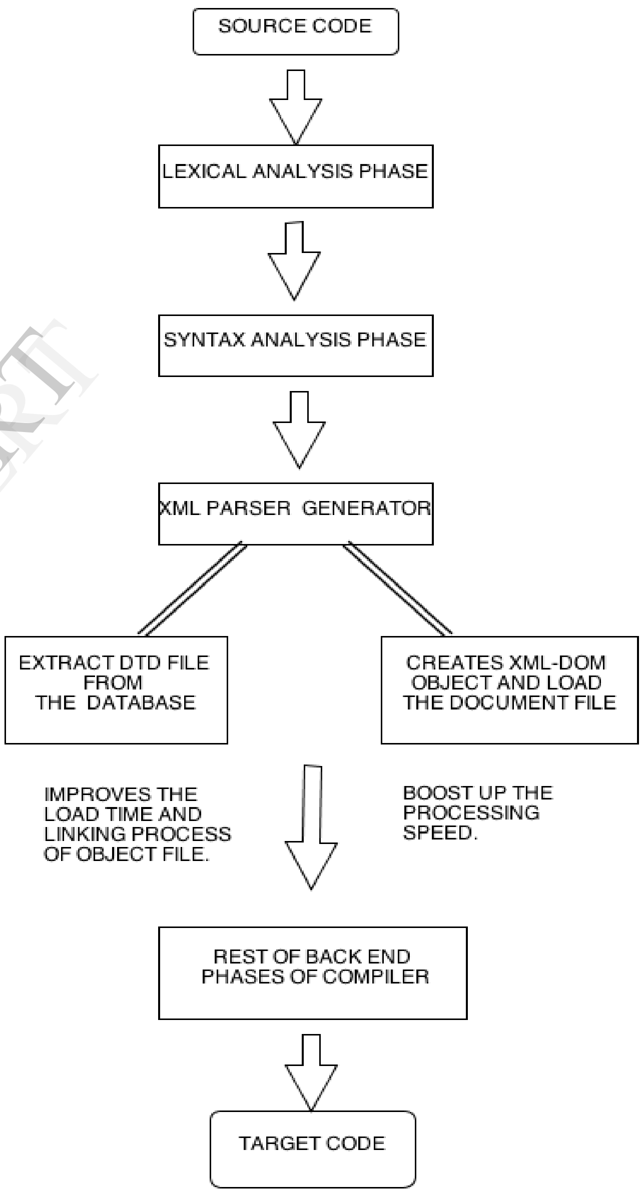
General Terms

VB, XML, Verification,parser generator, Doc-Type, DOM

INTRODUCTION

We know that XML parser converts an XML document into an XML DOM object. The XML DOM defines a standard way for accessing and manipulating XML documents. The paper just puts light on developing a tool which provides an elegant and effective framework for combining content and implementation. The parser integrates scanning, parsing, and validation into a single-pass without backtracking by utilizing compact tabular representations of schemas during run-time environment. So as to enhance the particular validation process of XML files, the tool helps in saving the load time and processing time via maintaining the reliability and bug free aspects of particular program, scripts written in form of markup languages.

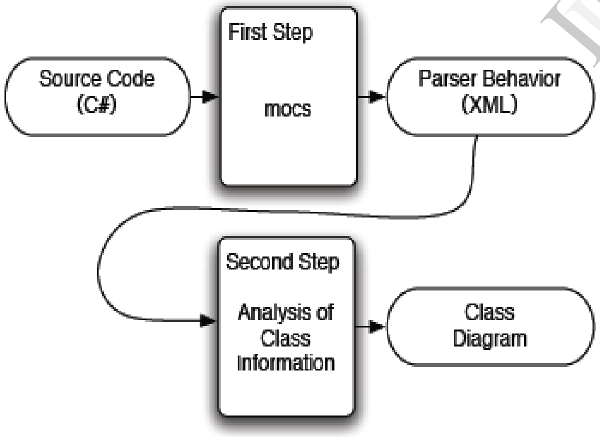
Structure of XML PARSER


SOURCE CODE
LEXICAL ANALYSIS PHASE
SYNTAX ANALYSIS PHASE
XML PARSER GENERATOR
EXTRACT DTD FILE FROM THE DATABASE
CREATES XML-DOM OBJECT AND LOAD THE DOCUMENT FILE
IMPROVES THE LOAD TIME AND LINKING PROCESS OF OBJECT FILE.
BOOST UP THE PROCESSING SPEED.
REST OF BACK END PHASES OF COMPILER
TARGET CODE

Firstly, when the XML Document file is being loaded by creating the object file with the help of "Microsoft XML DOM", Then the product module begin its threads which takes more time resulting in delaying the validation process of loaded XML document. So, in order to overcome this problem, the required tool is being developed which helps in enhancing the performance scale rate by decreasing the loading and linking time of the xml object file. Thus, it shows a performance penalty from the viewpoint of the generated XML document size.The procedures in the package can be used separately to tokenize or parse various pieces of XML documents. The framework supports XML Namespaces, character, internal and external parsed entities, and xml: space, attribute value normalization, processing instructions.

REVERSE ENGINEERING TOOLS AND COMPILER FRONT-ENDS

In many cases the system design documents are not updated after the source code is modified, Reverse Engineering tools have been developed as a solution for finding discrepancies between the source code and design documents. A basic function of the reverse engineering tools is to generate class diagrams from the source code.
In order to develop the reverse engineering tools, capability similar to a compiler front-end must be developed to analyze the source code. A typical compiler frontend reads the source code and executes lexical analysis, syntax analysis, and semantic analysis. It describes XML externalization built into compiler front-ends and its application to quick reverse engineering tool development.



DOMAIN DESCRIPTION

Today, the use of XML has spread across various fields of applications. Since we know that it is used for configuration of certain applications, storing data indatabases, retrieving data, exchanging data over the Internet and invoking remote methods.Actually when we write code in notepad or any professional XML editor, then after successful completion, we need to validate the xml files. So, the validation of those xml files will take enough time in the ongoing process.The tool shows a performance penalty from the viewpoint of the generated XML document size. For example in Dreamweaver software, there is a facility to validate the xml files. But firstly loading of product files takes place then the validation process starts.

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. Now we have to consider Token and its attributes, written in XML form. We have to save this code file as .DTD file. Say, “Token.DTD”.

<! ELEMENT LEX (Token)>

<! ELEMENT Token (Token_Id, Token_Proc, Token_Routine)>

<! ELEMENT Token_Id (#PC DATA)>

<! ELEMENT Token_Proc (#PC DATA)>

<! ELEMENT Token_Routine (#PC DATA)>

After writing the document type data file, we have to write the concerned XML document file. In it, we define the attributes concerned to .DTD file with the help of doctype system file and using standard version of XML file as shown below:

<? XML version =”1.0”?>

<! DOCTYPE LEX SYSTEM “Token.DTD”>

<LEX>

<Token>

<Token_Id> 20 </Token_Id>

<Token_Proc>Perform Tool Operations</Token_Proc>

<Token_Routine> the functions to be called for ongoing process </Token_Routine>

</Token>

</LEX>

XML DATABASES

XML-enabled databases are used to extract data from the database and transform it. The most widely used query languages for this purpose, *SQL/XML* and *XQuery*, which provides a set of extensions to SQL for creating XML documents and fragments from relational data.
Text-based native XML databases store XML in form of text which may be a file within the database itself, a file in the file system outside the database, or a proprietary text format which implies that relational databases storing XMLwhich possess an advantage of retrieving entire documents or document fragments, as all it takes is a single

index-lookup and deals with three types of indexes. The three possible types of indices are:
1. Value indices index text and attribute values.
2. Path indices index the location of elements and attributes.
3. Full-text indices index the individual tokens in text and attribute values.

The compiler is traditional basic software that is indispensable for developing software. The main purpose of a compiler is only the generation of efficient object code. However, there are rare cases where a compiler is used for different purposes from the code generation. The compiler includes excellent algorithms and valuable information based on the results of years of research. This paper describes XML externalization built into compiler front-ends by using a parser generator and its application to the quick development of a reverse engineering tool.

XML DOM Tree provides an interface to access and change the content and structure of an XML file.It is an object model of the document which resides in memory after parsing that makes manipulation easier. All elements can be accessed through the DOM tree. The elements, their text, and their attributes are all known as nodes. Theembedded path expression then loads the XML document describing the contents of the“LEX” file which includes ‘Token’ and further includes its Elements. Thus, we come to know that XML DOM provides Comprehensive functionality, maximal flexibility, Ease of development and conformance to standards.

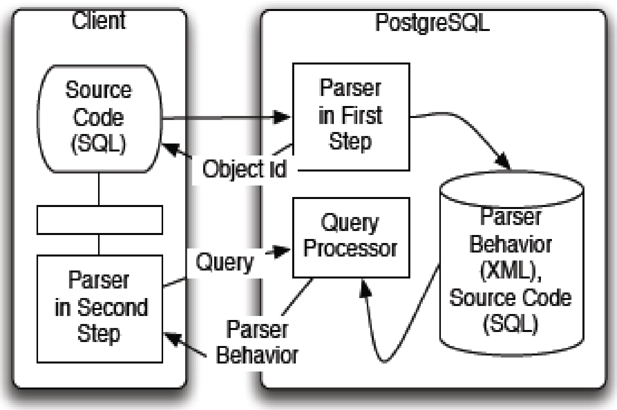
In all the above process of creating XML file and validating it, we face a problem of delay in system program loading and linking of object file. Therefore, a tool is required to overcome the above scenario of validation of XML files and dealing with XML databases. So, In order to boost up the processing speed and improving the linking process of object file, the following optimized code is effectively dealt with.

```
<Script Language=”VBScript”>
        Set dom_object=Create Object (“Microsoft.XMLDOM”)
        Path=Prompt (“Choose the XML Document”)
        dom_object.Load (Path)
If (dom_object.parseError <> 0) then
Msgbox dom_object.parseError.Reason & “        “
& dom_object.parseError.Line
          Else
 Msgbox dom_object.DocumentElement.XML
        End If
</Script>
```

In the above code snippet, we come to know the validation and verification process of xml document file. The XMLDOM object is loaded and verifies the xml language code. If on scanning of xml file, an error is found then the tool is responsible, to give the reason why that error has occurred and also responsible to highlight the line where that error has occurred in the corresponding XML document snippet code.

If on scanning of XML document file, XML parser did not find any error or bug, and thenDOM object is redirected to show the Document element of XML file and display the required result of corresponding XML document snippet. The parsing framework offers support for XML validation, and detects many validation errors. Content is validated given user-specified constraints, which the user can derive from a DTD, from an XML schema, or from other competing doc-type specification formats. When XML schemas are updated or extended, the tabular forms can be regenerated and populated to the generic engine without requirement of redeployment of the parser. This adaptive approach balances the need for performance against the requirements of redeployment of the Web services.

At run-time of the first step, the modified query processor in SQL analyzes the SQL source code, stores the parser behavior as large objects and other information, and it returns the object id.Theparser in the second step sends a query to the SQL in order to obtain the parser behavior generated via Query Processor.



Compiler frontend reads the source code and executes lexical analysis, syntax analysis, and semantic analysis. It describes XML externalization and produces output at the target code.This technique can be used for enhancing extensible high-performance Web services for large complex systems that typically require extensible schemas. The parser integrates scanning, parsing, and validation into a single-pass without backtracking by utilizing compact tabular representations of schemas and a push-down automaton (PDA) at runtime. The tabular forms are constructed from a set of schemas or WSDL descriptions through the use of permutation grammar. The engine is implemented as a PDA-based, table-driven driver; as a result, it is independent of XML schemas. When XML schemas are updated or extended, the tabular forms can be regenerated and populated to the generic engine without requirement of redeployment of the parser. This adaptive approach balances the need for performance against the requirements of reconstruction and redeployment of the Web services.

CONCLUSION:

In this paper, we introduced our XML parser that was implemented using XML classes. The main goal of this parser was to check XML documents for errors in a very easy and fast way, in order to help the programmer to determine whether the XML document is Well-formed or not.

Properties	XML Reader	Document Handler	XML:Parse()
Streaming	100%	No Streaming	100%
Productivity	No Productivity	100%	100%

A parser generator was developed to build XML externalization functionality into the compiler front-ends. After replacing an original parser generator, generating a parser using it, and modifying a few lines of source code in the compiler, we were able to obtain a special compiler that generates three kinds of XML data, namely, lexical information, parser behavior, and parse tree. The parser behavior was applied to quickly develop a reverse engineering tool for C#. During the tool development, a compiler front-end is separated into two-steps. In the first step, a special C# compiler reads C# source code, analyzes it, and writes the parser behavior in XML. In the second step, the parser behavior in XML is read and analyzed. The reverse engineering tool shows a performance penalty from the viewpoint of the generated XML document size.

Our experiments show the adaptive parser usually demonstrates performance of five times faster than traditional validating parsers. It makes the maintenance of an application element, which eliminates several classes of common bugs, and is capable of cutting, pasting, splitting and assembling XML documents with max efficiency. XML Parser minimizes the amount of application-specific state that has to be shared among user-supplied event handlers.

ACKNOWLEDGMENT

My sincere thanks to Mr.Manoj Kumar, (Center Head, Lally Infosys, and Jalandhar)for his valuable support and guidance that inspire me to do this task.

REFERENCES

I. ISBN: 978-3-540-43092-6 (Print) 978-3-540-45587-5 (Online)**Author(s):**Wei Zhang , Dept. of Computer Science, Florida State Univ., Tallahassee, FL, USA. http://metapaper.net/xml/ssax/

II. Microsoft, Visual Studio 2012, http://msdn.microsoft.com/en-us/library/dd831853.aspx

III. Ronald Bourret, XML and Databases, http://www.rpbourret.com/xml/XMLandDatabases.html.

IV. Introduction to XML DOM, http://www.w3schools.com/xml/xml_dom.asp

V. Xerces C++ parser. http://xerces.apache.org/xerces-c/.

VI. Practical Aspects of Declarative Languages. 4th International Symposium, PADL 2002 Portland, OR, USA, January 19–20, 2002 Proceedings by Shriram Krishnamurthy and CR Ramakrishna.

VII. XML Data Management, Addison Wesley.