XAWK

John Cieslewicz [*]
johnc@cs.columbia.edu

Gabriela Cretu [*]
gcretu@cs.columbia.edu

Shi Tak Man [†]
sm2173@columbia.edu

Prashant Puri [‡]
pp2119@columbia.edu

September 27, 2004

[*]First-year Ph.D student in the Department of Computer Science.
[†]Junior in the School of Engineering and Applied Science.
[‡]Masters student in the Department of Computer Science.

1 Introduction

XAWK is a programming language for the intuitive processing of XML files. As its name implies, XAWK is based on the AWK language and the design goal is to make XAWK as easy to use for processing XML files as AWK is for processing space-separated text files.

2 Related Work

AWK was created by Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan. It is an interpreted programming language that processes text files by searching for regular expressions. When a pattern is matched, AWK executes the user-specified instuctions associated with that pattern.

The idea of creating a language similar to AWK that operates on XML files is not new. Peter von der Ahé and Jakob Justsen built XAWK which does the same actions as AWK, but on XML files "in a way native to xml," that is, node-based as opposed to line-based processing [4].

Another approach was made by Stefan Tramm, who built XML GAWK. He replaced AWK's line-by-line file processing with an XML parser, thus processing the XML file token-by-token. XML GAWK matches the regular expression against any element name, element value, attribute name or attribute value [3].

Finally, this work is based, in part, on unfinished work to develop XAWK at Columbia University from November 2003 [1].

3 Language Features

3.1 Robust

Most importantly, XAWK will be a robust language with well-defined behavior. The implementation will provide thorough feedback for errors in user programs and XML input files.

3.2 Ease of Use

XAWK's program structure is very similar to AWK's. A programmer familiar with AWK will be able to write an XAWK program with ease.

In XAWK, the familar AWK pattern expression is replaced by a path expression whose form consists of a subset of the XPath syntax. Expressions refer, therefore, to either a node (an element or text) in the XML parse tree or the attribute of an element [1, page 3].

XAWK supports a subset of the rich features found in AWK. The language inclues standard operators for manipulating numerical and string variables. It also provides associative arrays. Additionally, XAWK includes built-in variables, such as an associative array of attributes on the current element, the name and value of the current element, and the fully qualified path to the current element.

Please see Section 4 for an example of XAWK in action.

3.3 Portable Implementation

Written in JAVA, the XAWK interpreter is portable to all systems with a supported JAVA Virtual Machine implementation.

From the command line, the user supplies a XAWK program file and one or more XML files to be processed. Similar to AWK, the user may also specify the XAWK program commands directly from the command line.

4 Example

XAWK can be used to compute and evaluate information from an XML file with ease. For example, if we have an XML file[1] containing a class roster and grade information, we can calculate the students' grades and print a class report. This can be implemented using the following sample code:

```
#Similar to XAWK, there is a special section for initialization
BEGIN{
  numStudents = 0;
  teacher = "unknown";
  teacher_email = "none";
  studentNames;
  studentGrades;
}

#This path matches a "class" tag in the root document with a child "teacher" tag.
#This small function stores teacher information
/class/teacher/{
  #CA is a built-in associative array of the attributes on the current element
  teacher = CA[name];
  teacher_email = CA[email];
}

#Record the student's name and average
/class/student/{
  studentNames[numStudents] = CA[name];
  problemSetScores;
  numProblemSets;
  projectScores;
  numProjects;

  # This nested path matches tag "assignment" within the /class/student/ path.
  # XAWK matches all "assignment" tags within the context of its parent, "student".
  /assignment/{
    if(CA[type] == "problem set"){
      numProblemSets ++;
      grade = 0;
      # text() matches a text value, i.e. <grade>value</grade>
      /grade/text(){
        grade = CV;
      }
      problemSetScores[numProblemSets-1] = grade;
    }else if(CA[type] == "project"){
      numProjects ++;
      grade = 0;
```

[1]Please see Appendix A for an example of this file.

```
      /grade/text(){
        grade = CV;
      }
      projectScores[numProjects-1] = grade;
    }
  }

  # Now compute the averages
  psAvg = 0;
  for(score in problemSetScores){
    psAvg += score;
  }
  psAvg = psAvg/numProblemSets;

  projAvg = 0;
  for(score in projectScores){
    projAvg += score;
  }
  projAvg = 0;

  studentGrades[numStudents] = psAvg*.40 + projAvg+.60;
  numStudents ++;
}

# The END block contains actions to be executed after the entire
# XML file has been processed.
END{
  print("----------------");
  print("Teacher: " + teacher);
  print("Email: " + teacher_email;
  print("----------------");

  for(n = 0; n < numStudents; n++){
    print((n+1) +"]\t" + studentNames[n]);
    print("\tGrade: " + studentGrades[n]);
  }
  print("----------------");
}
```

5 Implementation Issues

A key decision in the implementation of XAWK is the choice of an application programming interface (API) to use for processing the XML files. There are two standard APIs for processing XML documents with Java, the Simple API for XML (SAX) and the Document Object Model (DOM). For each API there exist different versions and different parsers that support it.

SAX, the Simple API for XML, is “event driven” [2]. SAX parses the XML document and provides content to the client application via callback functions. Because SAX uses call backs, the client application is responsible for maintaining any information about the XML document content. The call back design also makes SAX fast and memory efficient. Since the entire XML file is not stored in the memory, it is possible to process large files. SAX makes one pass over the XML file and, therefore, is not the right API choice when the client application needs access to the whole document at once or random access in the file. In that case, DOM is a better solution.

DOM, the Document Object Model, builds a model of the XML file as a tree in memory [2]. This makes DOM more flexible, but at the same time it makes it depend on the parser classes and, consequently, on the performance of the parser that is used. DOM is not as fast as SAX and needs a lot of memory for the document trees.

Because XAWK processes the XML file in a single pass, the SAX API offers a number of advantages. The better speed and memory efficiency of SAX allows XAWK to process larger XML files more quickly than a DOM API. The single pass nature of the SAX API is also appropriate for XAWK since XAWK processes a file from top to bottom. The main challenge of using the SAX API is the callback interface. The XAWK interpreter must control the advance through the XML document, because, as each path is matched, the interpreter must be able to advance according to the user program. Unfortunately, the SAX API's callback interface places the XML parser in control of advancing through the XML file.

To overcome this challenge, we propose that the XAWK interpreter be composed of two threads. One thread handles the SAX callbacks, assembling the parsed XML file into useable objects that are then placed in a thread-safe queue. The other thread would execute the XAWK user program, extracting XML objects from the queue as needed.

6 Summary

XAWK is a powerful tool for processing XML files. Its familiar AWK-like syntax enhances the language's accessibility, while also providing the programmer with ample flexibillity to process diverse XML files.

A Example XML File

This is a sample XML file that could be processed by the XAWK program in Section 4.

```xml
<class>
  <teacher name = "Stephen Edwards" email ="sedwards@cs.columbia.edu"/>
  <student name = "John Cieslewicz" gender="male">
    <email>johnc@cs.columbia.edu</email>
    <assignment type = "problem set" name = "Problem Set #1">
      <grade>95</grade>
    </assignment>
    <assignment type = "project" name="compiler">
      <grade>82.5</grade>
    </assignment>
  </student>
  <student name = "Gabriela Cretu" gender="female">
    <email>gcretu@cs.columbia.edu</email>
    <assignment type = "problem set" name = "Problem Set #1">
      <grade>96.2</grade>
    </assignment>
    <assignment type = "project" name="compiler">
      <grade/>
    </assignment>
  </student>
  <student name = "Prashant Puri" gender="male">
    <email>pp2119@columbia.edu</email>
    <assignment name = "Problem Set #1" type = "problem set">
      <grade>96</grade>
    </assignment>
    <assignment name = "compiler" type="project>
    </assignment>
  </student>
</class>
```

References

[1] Stephen A. Edwards, Seema Gupta, Miqdad Mohammed, and Peter T. Chen. XAWK Language Reference Manual. 11 2003.

[2] Rusty Harold. Processing XML with java. http://www.cafeconleche.org/books/xmljava/chapters/index.html.

[3] Stefan Tramm and Jürgen Kahrs. XML GAWK. http://homepage.mac.com/stefan.tramm/iWiki/XmlGawkTutorial.html.

[4] Peter von der Ahé and Jakob Justsen. XAWK. http://www.daimi.au.dk/ just/IWS/man.html.