A Third Way: The Hierarchical / Streaming XML Parsing Model

Christophe Chardonnet

Technical Consultant, OmniMark

Outline

- Introduction
  - DOM, SAX, OmniMark
  - Event-based vs hierarchy-based parsing
  - Streaming programming model
- Practical example
  - XML to HTML conversion
  - Element rules
  - Dealing with attributes
  - Reordering data

Outline

- Querying the element context
- Dealing with entities
- Handling errors
- Validation
- Demo
- Conclusion

Introduction

XML parsing models

- DOM
  - tree-based parsing
- SAX
  - event-based parsing
- OmniMark
  - hierarchy-based parsing

OmniMark programming language

- XML and text programming language
  - filters, batch conversions
  - CGI, servers
- Streaming programming model
- Rule-based program structure
- Integrated XML and SGML parsers

SAX parsing model

- Beginning of an element
  - one event
- End of the element
  - one separate event

OmniMark parsing model

- Occurrence of an element
  - single event
  - fires a single rule
- Elements can contain nested elements
  - hierarchy of fired rules
  - exact model of the hierarchy of the document

Streaming model

- Data is streamed from a source
- Working process as it flows
- Data streams directly to output
- No buffering of input or output

Technical application

XML to HTML conversion

- Illustration of the hierarchy-based model
- XML instance as input
- HTML as output

XML document

```
<memo>
 <header>
  <from>Barney Rubble</from>
  <to>Fred Flintstone &amp;amp; Dino</to>
  <sent>
   <date year="2000" month="09" day="13"/>
  </sent>
  <subject>Water Buffalo Bowling League standings</subject>
 </header>
 <body>
  <p>These are the current standings in the Loyal Order of Water
   Buffalos bowling league.
  </p>
  <standings>
   <team>
    <team-name>Bedrock Sand and Gravel</team-name>
    <members>
     <person>Fred Flintstone</person>...
```

HTML output


Water Buffalo Bowling League standings - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
MEMO
From: Barney Rubble
To: Fred Flintstone & Dino
Date: 2000-09-13
Subject: Water Buffalo Bowling League standings
These are the current standings in the Loyal Order of Water Buffalos bowling league.
Team
Players
Score
Bedrock Sand and Gravel
Fred Flintstone, Barney Rubble, Bam Bam Rubble, Dino
2854
Rock Vegas Review
Ann Margrock, Siegfried & Rock, Sharon Stone
2387
Hollyrock Hunks
Rock Hudson, Mario van Pebbles, Stoney Curtis, The Rock
1964
Please note that due to damage to the bowling lanes, caused by Fred bowling on tip-toes, a new lodge rule now requires that all contestants weighing > 12 stone must wear bowling shoes.
The next game is HollyRock Hunks vs. Rock Vegas Rockettes at the Bedrock Bowling lanes.
Poste de travail

Initiate XML parsing

- OmniMark is rule-based
  - Program execution begins in a `process` rule
- Initiate the parsing of an XML document
  - `do xml-parse ...`

```
do xml-parse instance scan file ‘doc.xml’
  output ‘<HTML>%c</HTML>’
done
```

XML parsing

- Parsing occurs at the “%c” in a string
- Part before “%c” is output before the parsing:

  <HTML>
- Part after “%c” is output after the parsing:

  </HTML>

“memo” element

- Rule-based
- Root element in XML document is “memo”
  - write a “memo” element rule

```
element “memo”
 output “<BODY>%n<H1>MEMO</H1>%n%c</BODY>”
```

“memo” element

- “memo” element of XML document corresponds to the “BODY” element of HTML
  <BODY> and </BODY> tags around “%c”
- output H1 title for the memo
- “%n” is a linefeed
- stack of element rules is starting to build

“header” element

- At the "%c", parser resumes parsing
- “header” element rule is fired

```
element “header”
 output “<table>%c</table>”
```

“header” element

- Wrapper tags for a table to present the memo header info
- Another call to “%c” fires up the parser again
  - another rule is fired

```
element “from”
 output
  “<tr><td><b>From:</b></td><td>%c</td>”
```

“from” element

- Program has now three-deep element stack (memo, header, from)
- OmniMark doesn’t parse the whole document before processing it
- Part of the output is already generated
- This is the streaming approach to XML processing

Streamed data content

- “from” element doesn’t contain any other elements, only data content
- “%c” streams the data content to the program’s current output

```
<tr><td><b>From:</b></td><td>data content</td>
```

- When parsing of “from” element is finished, element rule is allowed to finish, popping one level off the element rule stack.

Attributes

- “date” element contains attributes
- Attributes are collected into an associative array (shelf)
- Access values using the `attribute` keyword or “%v” escape sequence

```
element “date”
 output “<tr><td><b>Date:</b></td><td>”
 || “%v(day)/%v(month)/%v(year)%c</td>”
```

Reordering Data

- DOM (tree-based) allows access to any part of the document
  - whole document must be in memory
- OmniMark provides a mechanism for reordering data: `referents`

Referents syntax

- Write out a referent instead of a string

```
output referent referent-name
output referent “subject”
```

- At some time during processing, bind the referent to a string value

```
set referent referent-name to StringValue
set referent “subject” to “%c”
```

- Two separate actions
  - You can set a referent and not output it, and you can output a referent and not set it

“subject” element

- Output the subject in the title of the HTML page

```
do xml-parse ...
      output ‘<HTML><HEAD><TITLE>’
      || referent "subject"
      || ‘</TITLE></HEAD>%c</HTML>’
   done
```

“subject” element

- Set the referent in the element rule

```
element “subject”
 output “<tr><td><b>Subject:</b></td><td>”
 || referent “subject”
 || “</td>”
 set referent “subject” to “%c”
```

Querying the Element Context

- OmniMark maintains the context of the current element through the hierarchical stack of element rules
- Let ’s put "td" tags around team names ("team-name" element) in the standings table, not in plain text

Querying the Element Context

- Use element tests (parent, ancestor, open element…)

```
element “team-name” when ancestor is “standings”
   output “<td>%c</td>”

element “team-name” when ancestor isnt “standings”
   output “<b>%c</b>”
```

Dealing with Entities

- Markup characters "<" ">" and "&" must be escaped with text entities "<" ">" and "&".
- Need to find these characters in the data content and replace them
  - "translate" rules

Dealing with Entities

- Translate rules on data content

```
translate "<"
    output   "&lt;"
translate ">"
    output "&gt;"
translate "&"
    output "&amp;"
```

Handling Errors

- OmniMark validates as it parses
- If it finds an error in the XML stream, it fires a "markup-error" rule

```
markup-error
  put #error "Markup error: "
  || #message || " on line "
  || "d" % #line-number || ".%n"
```

Validation

- Validation against a DTD
- From well-formed to validating parsing by changing:

```
do xml-parse instance scan file ‘doc.xml’
```

into

```
do xml-parse instance scan file ‘doc.xml’
```

- "document" keyword activates DTD validation

Demo

- Let ’s run this code

Hierarchy Model: conclusion

- Easy-to-use processing model
  - scalable
- Information on the current context is easy to access
- High performance for big documents
  - minimise data copying
  - minimise memory usage

Hierarchy Model: conclusion

- Coding is simplified
  - less variable
- "process-oriented"
  - code tends to describe the process the program implements in a way that ’s clear and easy to read

Questions ?