JFlex와 BYacc/J를 이용한 XML Pull Parser 설계

장주현° 노희영
강 원 대 학 교 컴 퓨 터 과 학 과
hc98038@kwnu.kangwon.ac.kr, young@cc.kangwon.ac.kr

Design of Xml Pull Parser Using JFlex and BYacc/J

Ju-Hyun Jang° Hi-Young Roh
Dept. of Computer Science, Kangwon National University

요 약

XML을 파싱하는 기존의 파싱 Model인 Document Object Model은 메모리 내에 트리 구조와 같은 문서의 컨텐츠를 구성하기 때문에 메모리 사용이 많으며 Event 기반의 Push Model은 Consumer의 상태와 관계없이 파싱된 정보를 이벤트 처리 메소드를 이용함으로 처리의 지연, 처리의 지연을 위한 메모리 사용 등의 단점이 있다. 이에 반해 Pull 파싱 Model은 Client가 파싱의 요청을 하는 Model로써 Streaming Data를 파싱 할 시에 적은 지연시간, 메모리의 효율적인 사용, 파싱속도가 신속하다는 장점이 있다. 따라서 본 논문에서는 XML 파서 설계에 있어서 Pull 파싱 모델에 파서 *Generator tool*인 *JFlex*와 *BYacc/J*를 사용하여 기존의 Xml Parser보다 파싱 속도를 향상시키는 Pull 파서의 설계 방법을 제안하고자 한다.

1. 서 론

XML은 SGML과 같이 데이터의 구조와 속성을 정의할 수 있고 HTML의 특징처럼 웹 상에서 문서표현을 쉽게 할 수 있는 장점을 가진 Mark-up언어이다. 또한 문서의 표현 위주가 아닌 자료 위주의 Mark-up언어이기 때문에 자료의 효율적 표현을 위해 많이 사용되어 진다. 하지만 문서의 구조와 속성을 정의함으로 인해 문서의 표현 시에 이를 정확히 반영해야 하는 어려움이 있다. 또한 최근에는 Internet의 발달과 통신의 발달로 인해 mark-up언어가 Personal Computer, Server Computer 들과 같은 장치만이 아니라 PDA, hand-phone등 Wireless network 장치에도 많이 사용되어진다. 이 장치들은 낮은 대역폭, 한정된 메모리, 한정된 저장 매체라는 제한된 상황에 있으며 이러한 Wireless 장치에서도 문서의 정확한 표현은 가능해야 한다[1].

현재 XML(*eXtensible Markup Language*)문서의 표현을 위한 파싱 Model은 여러 Model이 연구되고 있으며 현재 사용하고 있는 Model에는 문서의 정보를 트리 형태로 구현하는 Model (ex. Object model), 문서의 구성 요소, 요소를 하나, 하나를 Event로 처리하는 Event 기반 Model(ex Push)이 있다[1]. 하지만 트리 형태의 구현 Model인 DOM이나 Event 기반의 Push Model은 Streaming되는 Data의 Processing에서의 지연현상 ,메모리의 비효율적 사용이라는 단점을 가지고 있다[1][2]. 이를 극복하기 위해 나온 파싱 Model이 Pull Model[7]이다. Pull Model은 Event 기반의 파싱 Model이라는 점이 Push Model과 유사하지만 파싱의 제어가 Consumer에 있다는 큰 차이를 가지고 있다. 또한 이러한 특징이 Streaming되는 정보의 파싱에서 지연의 감소와 메모리의 효율적 사용에 있어서 뛰어나다는 특징[1]을 가지고 있다. 그리고 파싱의 속도 면에서도 다른 파싱 Model에 비해 상당히 빠른 장점을 가지고 있다. Piccolo Xml for Java Parser 역시 다른 Push Model 파서와 같이 SAX(Simple Api for Xml) API를 기반으로 하여 작성한 파서이지만 기존의 Pull Model 파서 들보다 빠른 파싱속도로 문서를 처리하며 같은 Push Model 파서 에 비해 2~2.5배 정도의 빠른 속도를 보인다.[3] Piccolo Parser가 다른 SAX 구현 파서보다 빠른 이유는 파서의 구현을 JFlex(the Fast LEXical analyzer Generator for java) 와 BYACC/J(Berkely Yet Another Compiler Compiler/Java)를 사용하였다는 점이다. 기존의 파서는 hand-write 방식, 즉 파서의 개발자가 직접 손으로 coding 하는 방식을 사용하였지만 Piccolo Parser는 파서 Generator tool인 JFlex 와 BYacc/J를 사용하여 구문의 빠른 해석이 가능하며 파싱 상태를 정의하고 이를 파싱시에 사용하여 속도 향상을 가져 왔다.

본 논문에서는 이를 Pull Model 파서의 구현 방법에 적용하여 기존의 Pull Model 파서보다 더 빠른 속도를 가진 Pull 파서를 구현하고자 한다. 즉 기존의 Pull Model 파서와 같은 Interface를 구현 하지만 파서 Generator tool을 이용함으로써 좀 더 빠른 속도를 가진 파서의 설계 방법에 대해 제안하고자 한다.

2. 관련 연구

2.1 Parsing Model

XML 문서의 파싱 Model은 문서의 처리 방법에 따라 세 가지 Model이 사용되어 지고 있다. 그 중 첫 번째 파싱 Model이 문서의 요소들을 트리화하여 처리하는 Object Model, 문서의

요소 ,요소를 파서가 Consumer에게 전달하여 Event로 처리하는 Push Model, Consumer가 파서 에게 파싱을 요청하여 이를 기반으로 처리하는 Pull Model이 있다.[1] Object Model은 DOM(Document Object Model) 표준 Model이 있으며 이는 가장 오래된 사용방법으로 문서의 전체를 메모리 내에서 트리화하여 구성함으로 메모리의 효율성이 낮다는 큰 단점을 가지고 있다[1]. Push Model은 SAX(Simple Api for Xml) 표준 Model이 있으며 이 Interface를 통하여 문서의 각 요소, 요소들을 Event 처리한다[1]. Push Model은 파서가 Consumer의 상황에 상관없이 요소 와 정보를 전달함으로 인해 처리되지 못하는 정보들로 인한 지연현상이 생길 수 있으며 또한 Event 처리가 되지 않은 정보들을 저장해야 하므로 많은 메모리를 사용한다.

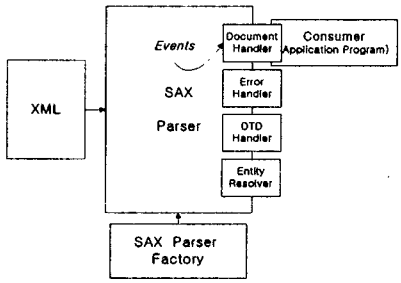

그림 1. Push 파서 구조

이에 반해 Pull Model은 파싱의 요청을 Consumer인 Application Program에서 요청하여 파싱이 되므로 지연현상이 적으며 메모리의 효율성도 뛰어나다. 현재 사용되어지는 파서 들의 bench marking에서도 Pull Model을 구현한 파서 들이 다른 Model의 파서 들보다도 빠른 파싱속도를 보이는 점도 이러한 이유 때문이다[3].

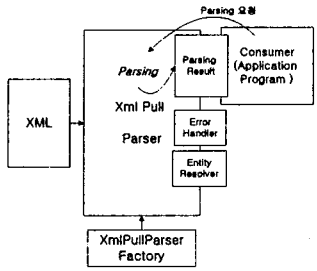

그림 2. Pull 파서 구조

본 논문에서 사용되는 Pull 파서 API인 XMLPull API는 세 개의 Class로 구성되어 있다. XmlPullParserFactory Class는 XmlPullParser의 Instance를 제공하는 Class 이다. XmlPullParser Interface는 XmlParser에 관련된 메소드와 상수에 관련된 정의를 하였다. XmlPullException Class는 입출력과 같은 일반적인 오류를 제외한 파싱과 관련된 예외 처리에 관한 Class 이다.[5] 또한 XmlPullParser Interface의 구현 시에 파싱되는 Event에 대한 상수들이 정의되어 있다. 이 상수들은 파싱된 정보가 문서의 시작을 나타내는지, 시작 태그, 마침 태그, 인지 등 문서의 content에 대한 Type을 상수로 정의하였다. 이는 Consumer가 파싱을 요청하고 파싱된 정보의 Event Type을 판단하기 위해 사용된다. 이 처리는 next(), nextToken(), nextTag() 의 세 메소드를 호출하며 파서는 Event Type에 맞는 값을 반환 해 준다. 응용 프로그램은 반환 값을 통해 Event Type에 대한 정보를 얻어내며 각각의 Event에 적합한 처리를 하여 준다.

2.2 JFLex 와 BYacc/J

Lex & Yacc은 1970년에 벨연구소에서 개발되어 7번째 유닉스 버전이후에 유닉스 표준 유틸리티로 사용되고 있을 정도로 어휘분석과 구문분석에 많이 사용되는 도구이다.

JFlex는 Java기반의 Lex로써 JLex를 향상시킨 scanner이다[6]. Token화와 어휘분석에 사용될 JFlex는 User Code, JFlex Directives, Regular Expression Rules의 세 부분으로 이루어져 있으며 User Code 부분에서는 구현 시에 사용되는 패키지들을 import 하고 JFlex Directives 부분에서는 Class의 정의, 상태의 흐름을 위해 사용될 상태들을 위한 변수들과 전역변수로 사용될 변수들의 초기화를 하였다. Regular Expression부분에서는 Token들의 정규표현식으로 정의를 하여 적절한 Token을 찾아낸다.

BYacc/J는 Token값에 대해서 Xml Pull Parser Interface의 실질적인 구현 응용프로그램으로써 JFlex에게 Token을 요청하고 Consumer(Application Program)에게 파싱된 Token과 Event Type값을 전달하는 역할을 한다. BYacc/J 역시 JFlex와 같이 세 부분으로 이루어지는데 UserCode 부분에서는 JFlex와 마찬가지로 사용되는 패키지들을 import 하며 Directives 부분에서는 Token의 Event Type을 위한 Grammar가 정의되며 Regular Expression 부분에서는 XmlPullParser Interface의 메소드들을 구현한다.

3. JFlex 와 BYacc/J를 이용한 Pull Model 파서 구조

본 논문은 기존 Xml Pull Parser API에 JFlex 와 BYacc/J를 이용하여 아래 그림-3과 같은 XML Pull 파서를 설계하였다

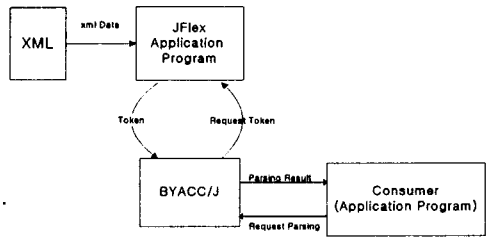

그림 3. Xml Pull 파싱 구조

위에서 파서에 파싱할 문서를 Consumer가 지정해 주고 그 후 BYacc/J에 파싱을 요청하면 BYacc/J는 JFlex에 Token을 요청한다. 이때 JFlex 응용프로그램은 정규표현식으로 Token의 형태를 지정하고 이를 통해 Token을 인식한다. BYacc/J 응용프로그램은 JFlex응용프로그램을 통해 인식된 Token의 Type을 판단하고 이를 Consumer에 전달하는 역할을 한다. 모든 Consumer와 파서 간의 정보 전달을 위한 Interface는 BYacc/J 응용프로그램이 Xml Pull 파서 API 메소드를 구현함으로 이루어진다.

4. Parisng Algorithm

파서 Generator tool을 사용하여 파서를 구현할 시 속도의 향상은 상태에 대한 빠른 대응과 구문의 신속한 해석으로 이루어진다. 파싱시에 사용되는 상태의 기본 변화는 Token의 Event 속성에 따라서 다음 상태를 예상하고 예상된 상태의 Token이 입력되면 상태에 대한 전이를 한다. 만약 원하지 않은 상태가 나오면 이는 well-formed XML 문서가 아니므로 XmlParser Exception을 처리하고 문서의 처리를 종료한다. 이때 파싱시에 사용될 상태들은 상수 값으로 할당되어 있으며 이는 Event 처리 시에 파싱된 Token의 Event Type에 사용된다.

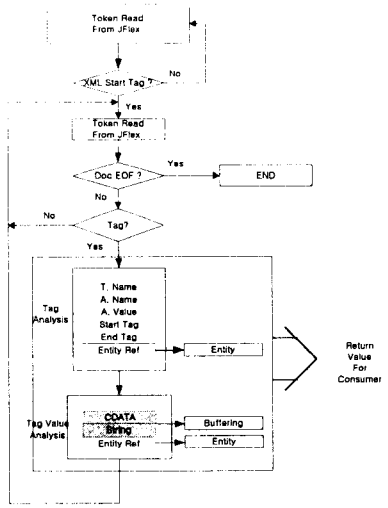

그림 4. BYacc/J에서의 Token 처리

이에 관련 Parisng 절차는 문서를 읽어 들이는 초기 상태에서부터 다음 Token이 Xml 문서의 StartTag인 "<? xml" 로 시작하는 지를 판단하고 StartTag이면 문서의 처리를 위한 초기 상태로 된다. 만약 XML문서에서 파서는 Start Tag를 찾을 때까지 Token을 읽고, 못 찾는다면 Parsing을 종료한다. 그러나 Xml StartTag가 나오면 다음 Token을 기다리는데 값이 Tag형태를 지닌다면 Tag 분석과정을 시행하게 된다.

Tag 분석과정은 Tag를 분석하는 과정으로써 Tag에서 얻어낼 수 있는 값들을 추출해 낸다. 이 과정에서는 Tag가 시작 Tag("<author>")인지 마침 Tag("</author>")인지를 판단하고 Tag의 Name, Tag Attribute의 이름, Tag Attribute Value등의 값들을 분석하며, 분석이 끝나면 Tag Value 분석과정을 시행하게 된다. 예를 들면 분석결과는 표-1과 같다.

<author id=3>

토큰 타입	Tag	Tag_ Name	Tag_Attribute_ Name	Tag_ Attribute_Value
토큰 값	Start_Tag	author	id	3

표 1. Tag 분석 예

Tag 값 분석 과정에서는 Data를 어휘분석을 하지 않고 그대로 표현해 주어야 하는 CDATA, 일반적 값을 의미하는 String, Entity 참조인 EntityReference 형태가 있다. 이 중에 CDATA, String은 빠른 입출력을 위해 Buffering을 하여 준다. 또한 EntityReference는 Entity값을 반환한다. 파싱된 Token과 Token Value는 Token이 인식된 시점에서 Consumer에게 반환되며 반환된 값의 표현을 Consumer의 역할이다. Tag값 분석 과정을 마치게 되면 Tag에 대한 분석을 위하여 Tag 분석과정으로 되돌아간다. 이와 같은 과정은 문서의 끝을 파싱할 때까지 반복되며 문서가 끝에 이르면 파싱이 종료된다.

5. 결론 및 향후 연구과제

본 논문은 효율적이고 빠른 파싱을 위한 파서 에 대해 제안하였다. 이를 위해 Pull Parser Model을 채택하였고 파서 Generator tool인 JFlex 와 BYacc/J를 사용하여 속도향상을 꾀하였다. 본 논문에서 제안한 파서는 메모리가 많고 대역폭이 큰 장치에서의 사용도 적합하지만 hand-phone, PDA와 같은 Wireless 장치에서의 사용에서 더욱 효율적일 것이다. 그 이유는 현재 Wireless장치의 보급의 확대와 많은 응용프로그램의 개발되고 있고 이 장치들의 정보 표현에 있어서 XML을 사용하는 경우가 많기 때문이다. Xml Pull Model 파서는 XML 문서 처리의 지연현상이 적고 메모리의 효율적 사용에 있어 기존 파서 Model 보다 뛰어나 Wireless에는 가장 적합한 파서일 것이다. 하지만 이를 위해서는 파서의 크기를 줄여 Wireless장치에 탑재될 수 있는 정도의 크기로 만드는 방법도 연구해 보아야 할 과제라 생각된다. 제안한 XmlPull Parser API는 Namespace의 사용이 불가능한 것은 아니지만 기본적으로는 지원하지 않고 있다. 또한 파서는 DTD도 사용하지 않는데 이는 파싱속도의 향상에 역 영향을 미치는 또 하나의 요인이기 때문이다. Xml Pull Parser Interface는 Namespace 와 DTD를 지원하지 않음으로 파싱 속도에 대한 향상을 꾀할 수 있을 것으로 판단된다.

참고문헌

[1] http://www.ibiblio.org/xml/

[2] Aleksander Slominski, "Design of a Pull and Push Parser System for Streaming", http://www.extreme.indiana.edu/xgws/papers/xml_push_pull/ ,May, 2001

[3] http://piccolo.sourceforge.net/

[4] http://www.troi.lincom-asg.com/~rjamison/byacc

[5] http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html

[6] http://www.jflex.de/

[7] Dennis M.sosnoski ,"XML documents on the run, Part 1,2,3" ,04.26.2002