JFlex와 BYacc/J를 이용한 JX-PullParser의 구현 및 성능 평가

장주현° 노희영
강 원 대 학 교 컴 퓨 터 과 학 과
jjangju98@naver.com°, rohhy@kangwon.ac.kr

Implementation and Performance Evaluation of JX-PullParser using JFlex and BYacc/J

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

요 약

현재 XML은 HTML의 대체 마크업 언어로써 그 사용이 확대되어 지고, 또한 XML 데이터를 위한 파서 모델과 파서 구현방식에 대한 연구가 진행되고 있다. 그 연구의 결과로 벤치마킹에서 PULL모델이 빠른 파싱 속도를 나타내었고, 파서의 구현 방식에 있어서는 PULL 모델 파서인 piccolo에서 사용한 parser generator tool인 JFlex와 BYacc/J를 사용하는 방법이 기존 파서 구현 방식에서 사용하던 Hand-write 방식보다 파싱이 빠른결과[1]를 내 놓았다. 또한 이 두 방법을 이용하여 기존의 파서 보다 파싱을 위한 시스템 설계를 제안하였다[2]. 본 논문에서는 JX-PullParser 시스템을 구현 하였고, xml 파서 속도 비교 도구인 saxbench 속도 비교 도구를 사용하여 기존 파서보다 빠른 파싱 속도를 보이는 것을 입증하였다.

1. 서 론

XML은 HTML의 대체 마크업 언어로써 HTML의 한계점을 극복할 수 있는 언어로 급부상하였다. 이에 XML의 정확하고 빠른 파싱에 대한 많은 연구가 진행되고 있다. XML의 파싱 모델인 Object, Push, Pull은 XML 파싱 모델의 대표적인 방법으로써 각각의 특징을 지니고 있다[3]. 특히 Pull 모델은 Object방식이 가지고 있는 XML 데이터의 수정, 삽입, 삭제 등의 연산이 불가능하다는 단점을 지니고 있음에도 불구 하고, Push 파서의 개량적인 모델로, 최근 파서의 개발동향인 빠르고 가벼운 파서 모델에 가장 적합한 모델이다[4]. 또한 최근의 산업표준화를 Pull 파서가 가지고 있던 기존의 사용의 불편함을 없앰으로써, 응용프로그래머들은 많은 관심을 가지게 되었다[3]. 또한 Pull 모델의 표준화 이전에 벤치마킹결과에서 가장 빠른 속도를 나타낸 piccolo 파서는 기존의 파서 구현방식과는 다르게 hand-wirte방식이 아닌 parser geneator tool인 JFlex와 BYacc를 사용함으로 기존 파서 보다 월등한 파싱 속도를 나타내었다.

본 논문에서는 Pull 모델의 표준인터페이스인 XMLPULL 인터페이스에 JFlex와 BYacc/J를 이용한 설계를 바탕으로 구현을 하고 saxbench project에서 사용한 속도 비교 도구를 이용하여 기존의 파서들과 속도 비교를 통해 기존 push, pull 모델파서보다 JFlex와 BYacc/J를 이용한 JX-PullParser파서가 빠른 파싱속도를 내는 것을 입증하였다.

2. JX-PullParser의 설계

본 논문에서 사용된 JFlex는 어휘분석 도구로써, BYacc/J는 구문분석 도구로 사용된다. 기존의 파서 구현방식인 hand-write방식에서는 어휘분석단계와 구문분석 단계의 뚜렷한 구분이 없는 것에 비하여 JFlex와 BYacc/J의 사용은 토큰 추출과 추출된 토큰을 통하여 정확한 문법정의가 가능하며 빠른 파싱 속도를 나타낼수 있게 되었다. 아래 그림 1은 JFlex와 BYacc/J를 이용한 JX-Pull 파서의 구조[2] 이다.

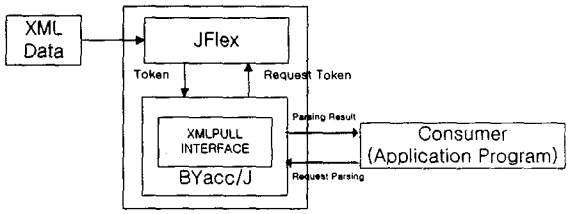

그림 1. JFlex와 BYacc/J를 이용한 JX-Pull 파서 구조

위 그림의 파서의 파싱 과정은 다음과 같다. 응용프로그램이 파싱하고자 하는 문서의 정보를 넘겨준다. 그러면 XMLPULL Interface를 통해 인식된 문서 정보를, JFlex에게 넘겨주고, 이후 응용프로그램이 파싱을 요청한다. JFlex는 문서의 정보를 통해 정의된 문서를 읽는다. JFlex가 문서를 읽어 가면서 token을 인식하고 얻어진 token을 BYacc/J에게 넘겨주면, BYacc는 정의된 문법과 token을 이용하여 이벤트 형을 결정하게 되며, 응용프로그램에게 얻어진 이벤트 형을 넘겨준다. 응용프로그램은 얻어진 이벤트 형을 통하여 다음 작업을 결정하고 다시 파싱을 요청하며, 이 과정은 문서의 끝까지 계속 반복되어 진다.

3. XML PULL 파서의 구현

3.1 Token, 구문정의

본 논문의 구현에서 사용한 Token, 구문정의는 XML spec(second edition)[5]을 기준으로 수정하여 작성하였다.[6] Token 정의는 spec의 EBNF로 정의하였고 구문 정의는 BYacc/J의 구문정의를 따라 정의하였다.

3.2 Non-Validating 파서

XML파서는 Valiating 파서와 Non-Validating파서로 나뉜다. Validaing은 문서의 내,외부 dtd와 XML 문서가 정확한 구조를 갖는지를 판단하여 응용프로그램에게 전달해주는 역할을 하는 파서이다. 본 논문에서 구현한 파서는 Non-Validating파서이며, 이는 최근 XML 파서의 경향이 빠르면서 가벼운 파서이며, 또한 Validating만을 지원하는 도구들이 나와있기 때문이다. Non-Validating파서는 XML문서의 Well-formedness만을 체크하고 이를 위하여 본 논문에서는 tag stack의 push, pop연산을 통하여 well-formedness를 체크하였다.

3.3 XML 선언의 처리

XML 선언은 버전, 문서의 인코딩 방식, 자립성에 대한 정의를 처리하는 부분이다. 본 논문에서는 piccolo와는 다르게 이 부분의 처리를 JFlex의 토큰 처리 부분에서 직접처리 하였다. 이를 위하여 XML 선언에 대한 상태 전이 부분과 토큰 추출 부분을 추가 하였다. 추가된 상태 전이는 아래 그림2와 같다.

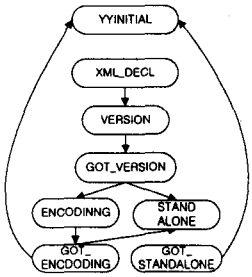

그림 2. XML 선언을 위해 추가된 상태전이

XML 선언은 기본적으로 <? xml 토큰으로 시작하여 version 토큰은 반드시 추출되고 그 후 인코딩 정보와 자립성 정보가 선언된다. 그러므로 위의 그림과 같이 정의하고 상태 전이만을 이용하여 값을 추출해 내었다. 모든 처리 후에는 초기 상태 전이를 하게 된다.

3.4 Namespace의 처리

네임스페이스는 최근의 xml 문서에서 사용되는 빈도는 그리 크지 않다. 그렇기 때문에 해쉬테이블을 사용하지 않고 배열만을 이용하여 작성하였다. namespac의 처리를 위해 1998년 11월에 정의된 namespace spec과 1999년 1월에 정의된 namespace spec에 정의된 접두사를 모두 처리 하였다. 아래 그림 3은 namespace의 처리를 위한 알고리즘이다.

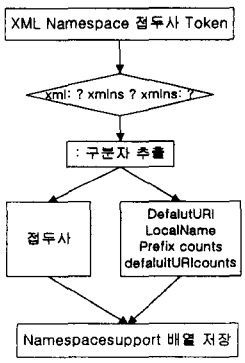

그림 3. namespace 처리 알고리즘

namespace의 처리 과정은 다음과 같다. namespace 접두사 토큰이 추출되면 정의된 접두사의 위치를 검색하고 접두사와 URI등의 정보를 따로 추출하게 된다 추출된 정보는 Namespace support 배열에 저장되며 얻어진 namespace는 응용프로그램의 요청시에 값이 반환된다.

3.5 엔티티 리퍼런스의 처리

본 논문에서 사용한 엔티티 리퍼런스의 처리는 해쉬테이블을 이용하였다. XML spec에 미리 정의되어 있는 엔티티 들은 파싱의 시작과 함께 엔티티를 키로 리퍼런스 값을 값으로 하여 저장된다. 내부 엔티티정의에 있어서도 해쉬테이블을 이용하여 기존에 정의된 엔티티의 유,무를 판단하여 키와 값을 저장하게 된다. XML 문서내에서 엔티티 토큰의 발견시에는 해쉬테이블을 참조하여 값을 반환해 준다. 아래 그림 3은 엔티티 리퍼런스의 처리 알고리즘이다.

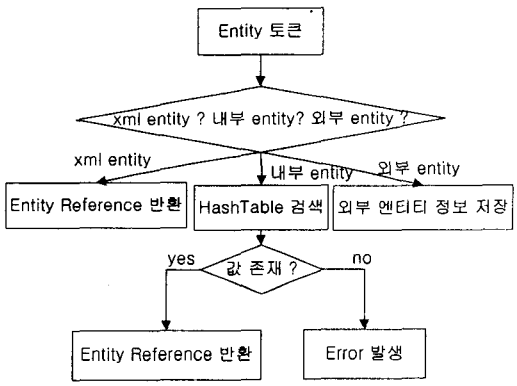

그림 3. Entity 처리 알고리즘

4. 구현환경 및 벤치마킹 결과

본 논문에서 사용한 시스템 환경은 Windows 2000 Server(SP4), 1Gbyte DDRram, 60Gbyte HDD 환경에서 구현하였으며, 벤치마킹에 사용된 문서와 파싱 도구는 saxbench project에서 사용된 것들을 그대로 사용하였다.

또한 JFlex 1.4, BYacc/J 1.1 , XML Pull 1.1.4 버젼을 이용하여 구현하였으며 Token정의, 문법정의를 위한 XML spec은 1.0(second edition)을 기준으로 하였고, namespace 처리를 위한 스펙은 1999년 1월 14일에 재정된 스펙을 기준으로 하여 구현하였다.

벤치마킹에 사용된 saxbench의 과정은 아래 그림4와 같다.

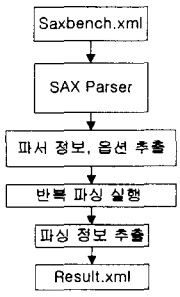

그림4 saxbench 과정

먼저 벤치마킹을 위한 파서 설정과 파싱의 반복횟수, namespace의 사용유무, 객체의 재사용여부, 파싱될 문서를 xml로 정의하고 정의된 xml은 sax 파서로 파싱되며, 그 후에 지정된 파서를 옵션에 따라 실행하게 된다. 정의된 반복횟수만큼의 파싱을 하고, 시작 시간과 끝 시간의 차이를 반복횟수만큼으로 나눔으로써 평균 파싱시간을 구하게 된다. 파싱 결과는 실행 창과 result.xml문서를 통해서 확인할 수 있다.

벤치마킹에 사용된 파서는 본 논문에서 구현한 JXP파서와 Pull파서의 종류인 MXP와 Pull파서 기반의 모바일 장치 파서인 KXML 기존 파서중 가장 빠른 파싱 속도를 보인 Piccolo 파서를 사용하였고 사용된 문서들은 네종류로 나뉠수 있다. 첫번째는 soap 처리를 위한 벤치마킹문서로 크기에 따라 세 개의 파일로 구성되고, 두 번째는 GNU-licence를 가진 랜덤 xml 생성 도구를 이용하여 생성한 랜덤 xml문서로 파일의 크기에 따라 두 개의 파일이 있으며, 세 번째는 문서에서 태그가 차지하는 비율이 80%인 경우와 마지막은 반대로 태그의 값이 80%인 경우의 벤치마킹을 하였다.

아래 그림은 saxbench 도구를 통해서 측정한 파싱 속도의 그래프이다.

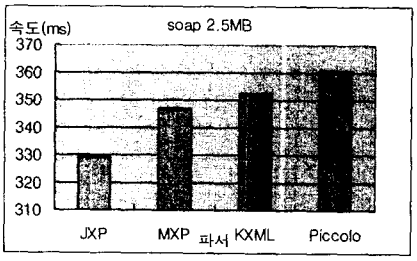

그림 5. soap 문서의 파싱 속도 비교

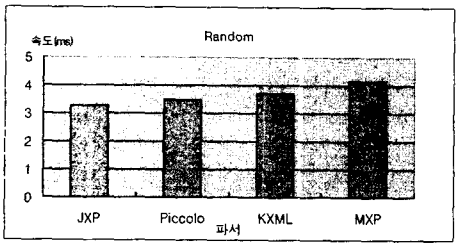

그림 6. Ramdom 문서의 파싱 속도 비교

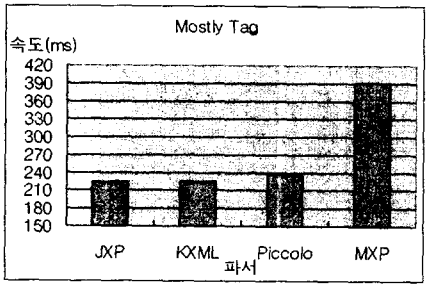

그림7. Mostly Tag의 파싱 속도 비교

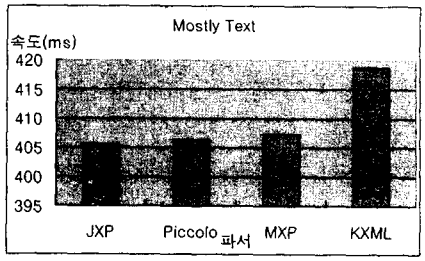

그림 8. Mostly Text 문서의 파싱 속도 비교

그림 5,6,7,8을 살펴보면 파싱 속도비교에서 JXP가 다른 파서에 비해 빠른 속도를 나타냄을 알 수 있다.

5. 결론 및 향후연구과제

본 논문에서는 XML 파서의 빠른 파싱 속도를 위하여 Pull 모델을 채택하고, 구현방식에 있어 JFlex와 BYacc/J를 사용하는 시스템을 구현 및 벤치마킹 하여 기존의 파서보다 빠른 파서의 설계 및 구현을 입증하였다.

본 논문에서 사용한 pull 모델은 빠른 속도뿐 아니라 메모리 사용에 있어서 기존의 파싱 모델보다 효율적인 모델이다. 따라서 향후에는 구현된 파서와 기존 파서들간의 메모리 사용량에 대한 연구가 필요하고, 또한 본 논문에서 구현한 XML 스펙인 second edtion이후인 2004년 2월 4일에 나온 third edtion과의 호환성에 대한 연구도 향후 연구과제로 남는다.

참고문헌

[1]Yuval Oren, http://piccolo.sourceforge.net/bench.html
[2] 장주현,노희영 JFlex와 BYacc/J를 이용한 Xml Pull Parser 설계, 한국정보과학회 추계 학술발표논문집, 제 30권 제1호(B), 31 ~ 33
[3]Dennis M. Sosnoski ,http://www.javaworld.com/javaworld/jw-02-2002/jw-0208-xmljava_p.html
[4]Dennis Sosnoski , http://www.ibiblio.org/xml/
[5]http://www.w3.org/TR/2000/REC-xml-20001006
[6]장주현,노희영 XML PULL Parser를 위한 JFlex와 BYacc/J의 적용방안,한국정보과학회 추계학술발표회 제30-2권 제1호 259-261